cadence®

Placing the Elements

Product Version 23.1
September 2023

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

Contents

1

APD: Generating Standard Components

You create the symbols that represent the various components in your design after setting up the design parameters. Allegro X Advanced Package Designer (APD) provides several design utilities to create symbols for a die, a BGA, and a plating bar, if necessary. For information about the die stackup, refer to the *Allegro User Guide: Preparing the Layout.*

Defining Dies

The die vendor usually supplies the **D**ie **I**nformation **E**xchange (DIE) file in ASCII form. The DIE file contains die size, pin size and location, and the logical data associated with each pin. The DIE file structure consists of BLOCK data. A BLOCK is a collection of die descriptions. While the DIE file format may contain much data about the die (signal integrity characteristics, thermal characteristics, and so on) only the pin size, shape, and location, and the die size are used to create the die symbol. The default unit in a DIE file is microns (um) unless you change the default user units using *Setup – Design Parameters* (*Design* tab). Only units of meters, inches, or mils are allowed.

To add a die to your design, you must add a die *symbol* that the tool can understand. All related mechanical and net information for the die is imported into the Symbol Editor to create a symbol native to APD. You can create a die symbol in several ways:

- *Generate – Die Generator* (`die generator` command)
  The Die Generator Wizard presents a series of dialog boxes to guide you through the process of experimenting with different die configurations and generating symbols for wire bond or flip-chip dies. The Die Generator dialog boxes graphically represent the results of your pin arrangement and numbering choices. Without opening the symbol and padstack editors, you can define your die and manipulate padstack data before actually creating the appropriate symbol. However, for special customization, you must use the padstack or Symbol Editor.
- *Generate – Die Text-In Wizard* (`die text in` command)
  The bare die information regarding pin location, pin size, pin shape, and die dimensions is in a spreadsheet format. Exporting this information from the spreadsheet to a text file, which the *Die Text-In Wizard* imports, is the most common method for creating the bare die symbol.
  To know more about the die text-in file, see .
- *File – Import – D.I.E. Format* (`die in` command)

You can also obtain the bare die information if the silicon vendor can produce the industry standard DIE file. Thus, you can create the die symbol by importing the DIE file, which automatically generates die symbols from DIE files. When importing a DIE file, APD looks for a file with the extension `.die`.

- Padstack or Symbol Editor for special customization.
  You can also create or edit the die symbol within the Symbol Editor. To edit the pin padstacks use Padstack Editor.

Using the Die Generator

Use the Die Generator Wizard to define a die in relation to the padstacks and pin arrangement and numbering. The Die Generator Wizard's dialog boxes accept the data required to create a symbol. Using the Die Generator Wizard lets you set dimensions and component origin; generate the component outline; and generate, save, import, search for, and change padstacks.

You can also generate the following pin arrangements automatically:

- Full Matrix (for flip-chip die only)
- Perimeter Matrix with a specified number of rings
  With a perimeter matrix pin arrangement, you can further specify:
  - Separate staggering options for core and perimeter pins
  - Separate padstacks for core and perimeter pins

For a flip-chip die, you can specify a rectangular core area and separate pin pitches for core and perimeter pins.

With the Die Generator, you can preview the component that will be generated using the settings in the Wizard's dialog boxes. You can edit the dialog boxes' settings until your die meets your requirements by clicking *Back*.

Once you are able to preview the die, it is instantiated into your design. The generator creates a device file, `packagename_die.txt`, in your working directory, and the tool writes the symbol information to the database. Consequently, you can no longer edit the symbol; you must delete the symbol from the database before using that reference designator again.

To generate a die, perform each of the following procedures, using the `design wizard` command:

- Define die outline and attachment method
- Define the pin arrangement
- Define the padstack information

- Define the pin numbering scheme
- Preview the symbol prior to instantiation into your design

You can also view and edit die properties. For example, previously, if you incorrectly entered the information about how a die was mounted in the component, you had to delete the die or re-import it. Now, you can view and edit the setting that describes how the die is mounted in the component: flip-chip or wire bond.

Using the Generate – Die Text-In Wizard

If bare die information regarding pin location, pin size, pin shape, and die dimensions is in a spreadsheet format, you can export this information from the spreadsheet to a text file, and then choose *Die Text-In Wizard* (`die text in` command) to create the bare die symbol.

⚠ Use only alphabetical characters to name column headers in the import file. Numbers or special characters are not supported in column header names.

To generate logical connectivity, choose *Die Text-In Wizard* after you create a die component and pins. The *Die Text-In Wizard* also creates the device file for the die symbol used during the third-party netin process.

The *Die Text-In Wizard*:

- Generates die symbols, nets, and properties by importing an ASCII spreadsheet of die pin information.
- Manipulates the spreadsheet information in the Die Text-In Wizard to modify individual pin values.
- Places columns of data in a standard format.

Symbol View

You can import a DIE file into the Symbol Editor as well as the Layout Editor. The process is the same except there is no logical information created in the database, that is, the reference designator)

Also note that whatever the symbol drawing is open, the layout editor replaces the drawing name with the die symbol name as found in the DIE file (`die_name`). In this example `anyname` is replaced with `388die`.

Die File Import Comparison

Data Created	Layout View	Symbol View
Device file (`388die.txt`)	Yes	Yes
Die pin padstack (`die.pad`)	No *	Yes
Source symbol file (`388die.dra`)	No *	Yes
Compiled symbol file (`388die.psm`)	No *	Yes
Partial Netlist file (`388die_nl.txt`)	No	Yes
Logical Component (*U1*)	Yes	No

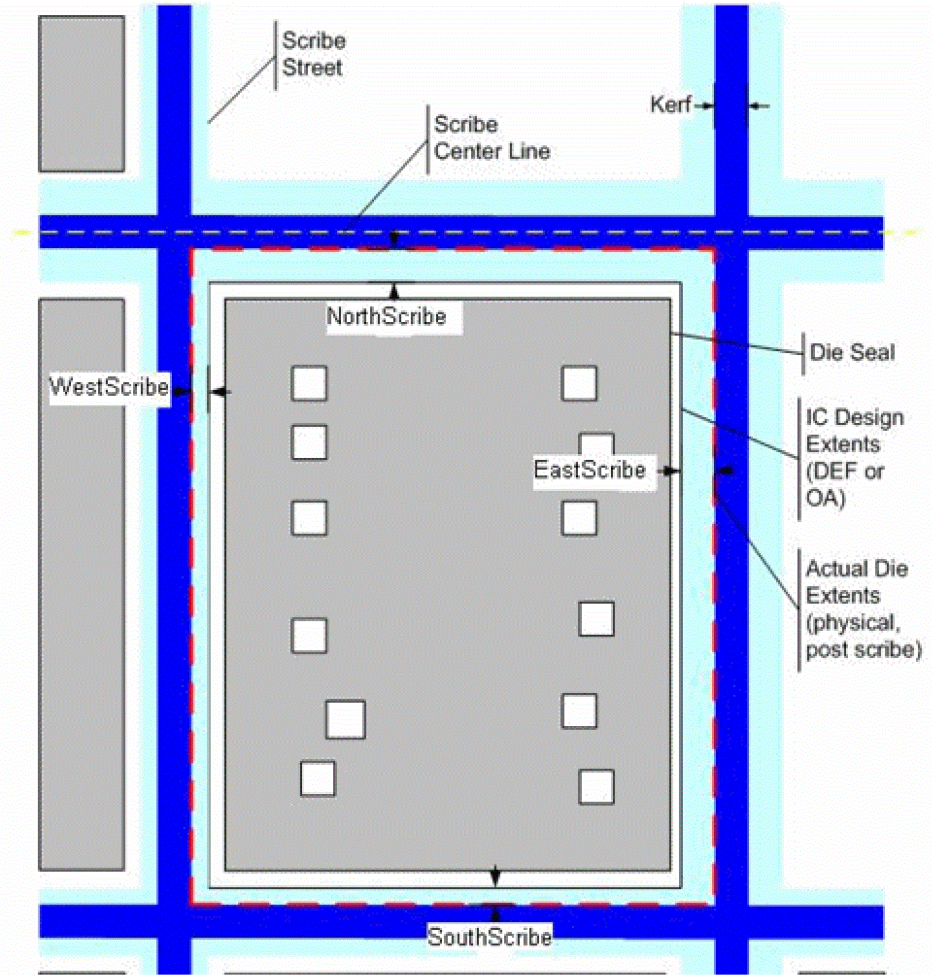
Use the *File – Export – Libraries* command from the Layout View to generate the symbol data (`.pad`, `.dra`, and `.psm`).

- die generator
- die text in
- die in
- Padstack Editor
- design wizard
- die properties
- Die Text File Format Specification

About Die Scribe Lines

During manufacturing, dies are generally created in batches where many dies are created on one wafer board. When the dies are laid out on the wafer, a space is left between each die boundary so that the scriber can cut the individual dies. This space (scribe area) is wide enough for the saw blade to pass through without affecting the components on the die. When a saw cuts, it removes a channel on the material it is cutting. The following figure shows the scribe area between dies on a wafer board.

Scribe Area on Wafer Board



Previously, when you created a co-design die in an IC tool (or imported the die using OA or DEF), its physical extents were based on the design extents from the IC design tool, not the actual physical qualities of the die. Yet, you need to consider these actual physical extents when placing dies in a component substrate layout. You must use them for any measurement, clearance, assembly, or placement rule checks that you perform in the component substrate design. This is particularly important in situations such as measuring the clearance between the 3D path of a bond wire and the edge of the die, or the distance between closely-spaced components. If you do not use the actual die size (including the scribe area), you may place components too closely together in the layout design and, at the time of manufacturing, conflict with the placement of other components.

⚠ When using the `die generator`, `die text in`, `die in`, and `place manual` commands, it is assumed by the package layout tool that the symbol already includes the scribe dimensions and any optical shrink of the geometries.

Scribe Lines Feature

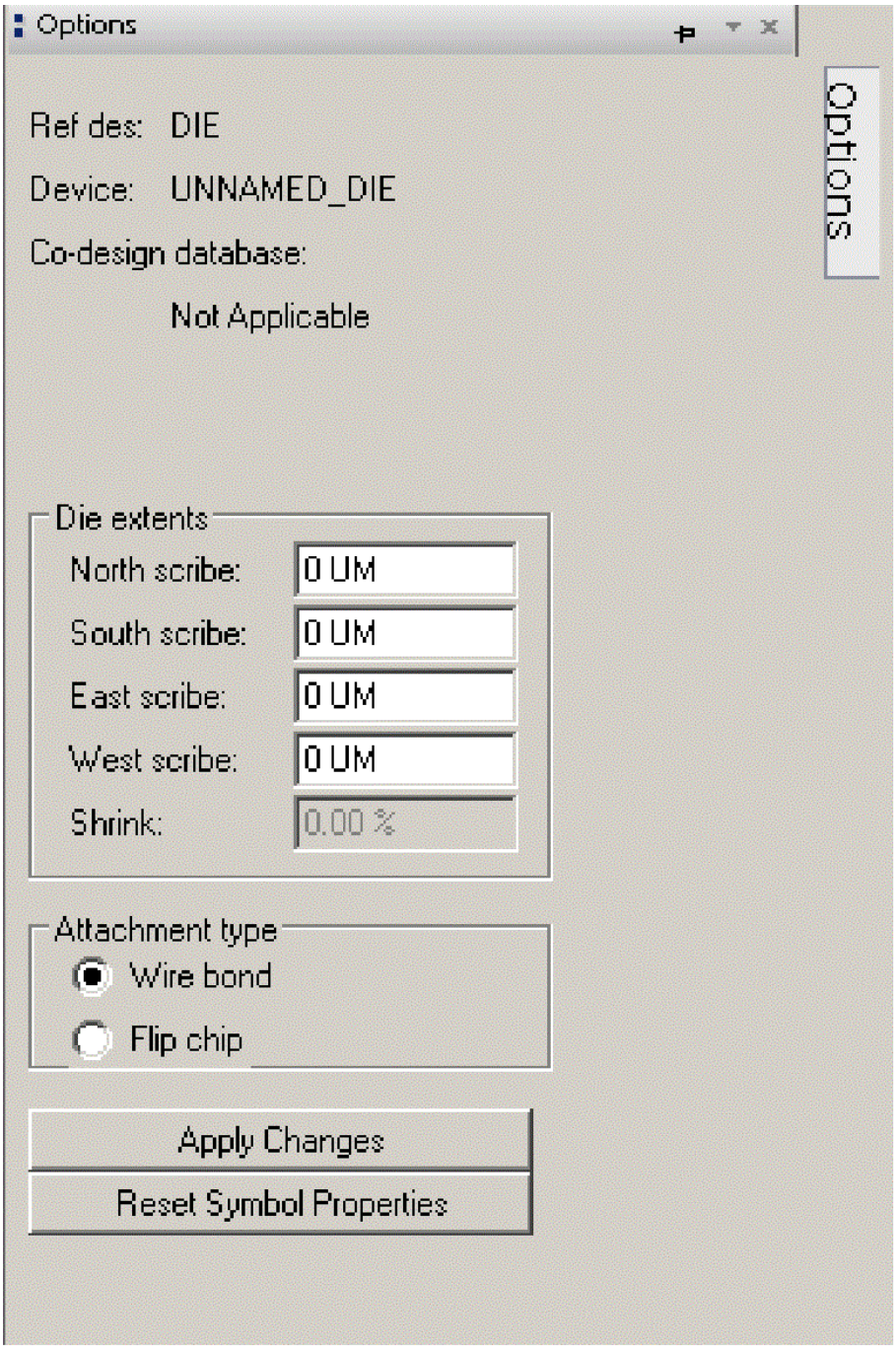
The Scribe Lines feature shows the physical extents of the actual manufactured die. This includes the scribe area outside the design extents that is part of the wafer scribe or sawing process. Scribe lines are added after the optical shrink if applied. Depending on the method or source of the data you use in generating the symbol representation of the die in the component, the source data may or may not include the scribe line information in the boundary it uses to represent the die.

The `die text in`, `def in`, `oa in`, and `add codesign die` commands let you add the size of the scribe area when you create the die in the package design so that you achieve the true dimensions of the finished die. If you define a scribe area with one of these commands, the log file for the specified command records this information. Errors are reported in the console window.

If you have already placed your die and did not include a scribe line, you can add a scribe line using the `die properties` command. This command also lets you view and edit the current scribe values assigned.

The DFA tools use the DFA_PLACE_BOUND_TOP/BOTTOM shapes during interactive placement with the `place manual` command to actively prevent you from placing components too close together. Scribe lines help to provide the die components with the most accurate physical placement boundaries, which the DFA tools require.

Die Properties Parameters



Related Topics

- `die text in`
- `def in`
- `oa in`
- `add codesign die`
- `die properties`

About Die Shrink

As technologies improve, die designs undergo optical shrinking during the manufacturing process. This involves the shrinking of all the geometries in the IC using a newer fabrication process to create a smaller, faster, and cheaper die. It is important to note that the chip is otherwise unchanged and the IC design database is unaffected. You apply the shrink factor when you import the die into the component layout tool.

You can apply a die shrink value when you run the die `text in`, `def in`, and `add codesign die` commands. The shrink value is recorded in the log file of the command you specified when you set the parameters, and errors are reported in the console window.

To view the shrink value assigned, run the `die properties` command. By default, the pre-shrink definition of the die will be retained in the design for future reference and viewing. This allows you to revert to the original definition and see the die as it exists in the IC design space.

Both, the original definition and the updated definitions, will appear in the `ALT_SYMBOLS` list for the component. If you do not want to save the original definition,
enable the *icp_shrink_nosave_original_symdef* environment variable.

> To remove the original die definition from the design to reduce design size, first remove the *symbd_no_del* property using SKILL. It is also recommended to keep a backup of the original die definition DRAs before deleting them from your design.

The `oa out` and `def out` commands cannot be used to export a shrunken die. To export a shrunken die, use the `die text out` command. Exporting a shrunken die to a library component (`.dra` or `.psm`) removes any recorded information about the shrink that was applied.

Related Topics

- die text in
- def in
- add codesign die
- Scribe Lines Feature
- die properties

Defining a BGA

You can create a BGA symbol using the BGA Generator Wizard or the BGA Text-In Wizard. For procedural details on these wizards, see the following topics in the *Allegro PCB and Package Physical Layout Command Reference*.

- *Add – BGA Generator* (`bga generator` command)
- *Add – Die Text-In Wizard* (`bga text in` command)

The BGA Generator Wizard

The BGA Generator Wizard lets you experiment with different component configurations and generate the component without using the symbol and padstack editors to create a padstack. For customization, however, you must use the padstack or Symbol Editors.

Choose *Generate – BGA Generator* (`bga generator`) command to display the BGA Generator Wizard. The Wizard's dialog boxes accept the data required to create a symbol. The Wizard lets you set dimensions and component origin; generate the component outline; and generate, save, import, search for, and change padstacks. You can also generate the following pin arrangements automatically:

  - Full Matrix
  - Perimeter Matrix
  - Perimeter Matrix with core pins

  With a *Perimeter Matrix* pin arrangement, you can further specify:

- Separate staggering options for core and perimeter pins

- Separate padstacks for core and perimeter pins

You can specify a rectangular core area and separate pin pitches for core and perimeter pins for either *Full* or *Perimeter Matrix* pin arrangements.

When the BGA Generator Preview dialog box appears, the BGA is instantiated into your design. You can edit the dialog boxes' settings until your component meets your requirements by clicking *Back.*

The generator creates a device file, `packagename_bga.txt`, in your working directory, and the tool writes the component information to the database. Consequently, you can no longer edit the component; you must delete the component from the database before using the component name again unless you use the BGA Editor.

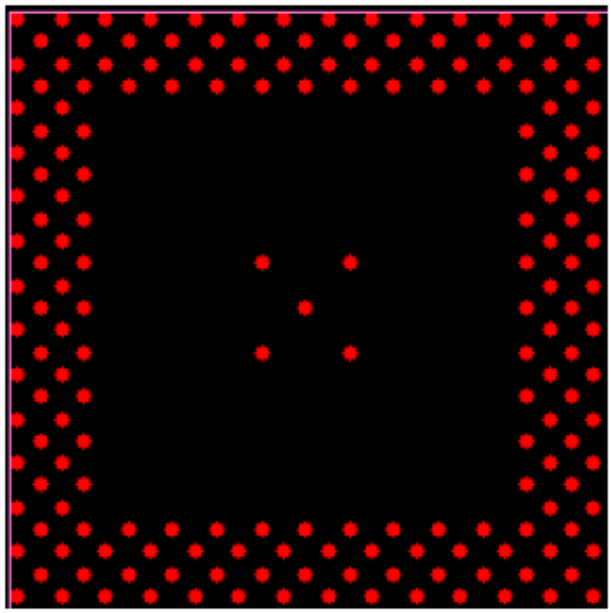
Sample Parameters for Creating Staggered Perimeter Matrix with Staggered Core Pins

For a staggered perimeter matrix pin arrangement with staggered core pins, setting the following parameters results in an arrangement as shown in the following figure.

X coordinate	2200
Y coordinate	1700
Height	700 mil
Width	700 mil
Perimeter Matrix	enabled
Outer Rings	4
Stagger Outer	enabled
Core columns	3
Core rows	3
Stagger core	enabled
Pin Pitch horizontal	25 mils
Pin Pitch vertical	25 mils

Core multipliers	2
Edge spacing	12.5 mils
Number of Pins horizontal	27
Number of Pins vertical	27

Staggered Perimeter Matrix with Staggered Core Pins



The BGA Text Wizard

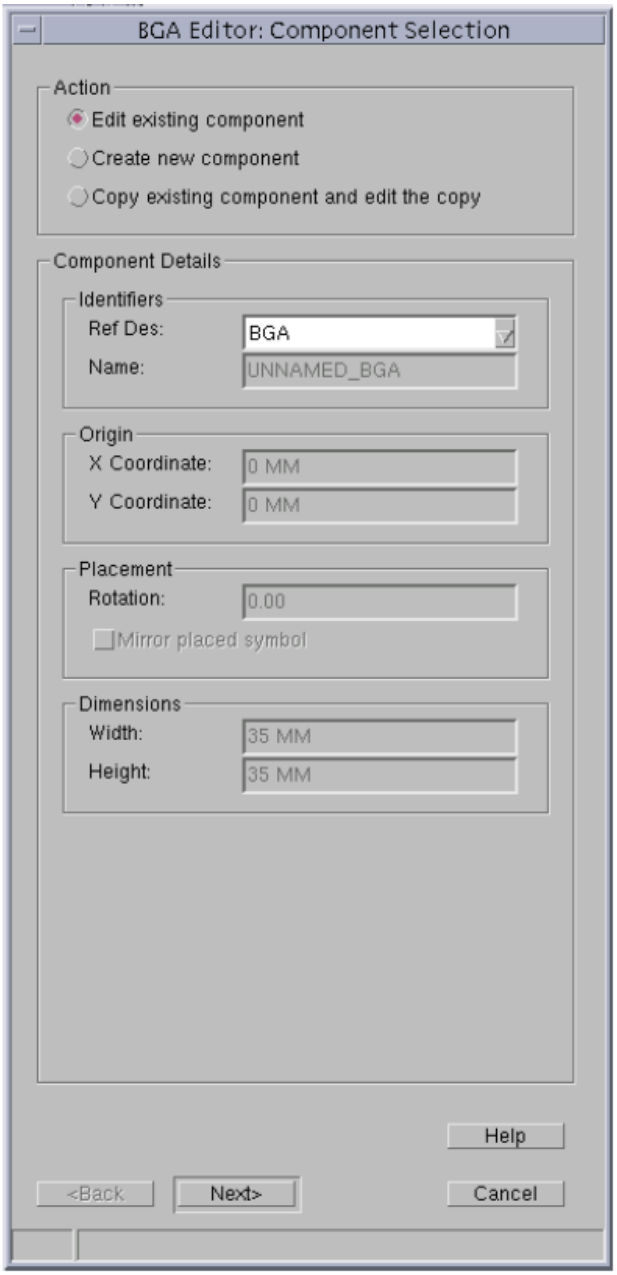
If the symbol information for the BGA is in a text file (saved from a spreadsheet), then use the **BGA Text-In Wizard** to create the BGA symbol. Use the `bga text in` command to use this wizard.

The BGA Editor

You can edit a BGA symbol previously created with *Add – BGA Generator* (`bga generator` command), *Add – BGA Text-In Wizard* (`bga text in` command), or other method. Running the BGA Editor (*Edit – BGA)* lets you edit the symbol to represent the specific requirements of the current design without leaving the tool environment. This section provides an overview of the BGA Editor.

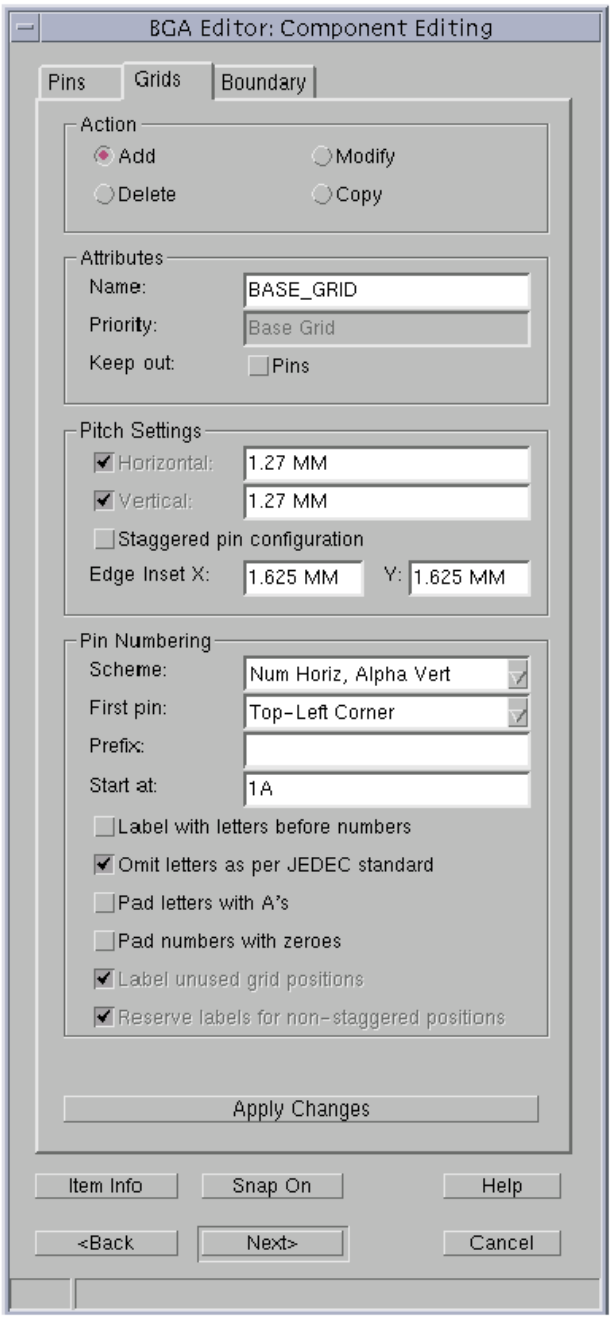
The initial symbol you create with the BGA Generator provides a default pin arrangement, reference designator, and pin-labelling scheme. Using the BGA Editor, you can then edit the existing BGA by adding new pins to the arrangement, deleting existing pins, copying existing pins from one location to another, swapping pins, or moving existing pins from one location to another. You can also change the currently designated padstack, pin use, or net assignment when adding, copying, or modifying pins. For example, core pins in a perimeter BGA may require different size padstacks for better conductance and heat transfer than padstacks used for signal connections on the perimeter pin array. In addition to the changes you can make to an existing BGA, you can also choose to create a new BGA or copy the existing one, then edit the copy. The following figure illustrates the Component Selection dialog box that allows you these options.

BGA Editor: Component Selection Dialog Box



Following component selection (existing, new, or copy) in the initial dialog box, the BGA Editor requires that you define the grid name and priority attributes, pin numbering scheme, starting pin location, JEDEC naming standards use, staggered pin configuration settings, and horizontal and vertical pitch that determine the grid settings. The following figure illustrates the Component Editing dialog box where you define these parameters in the *Grids* tab. Other controls are available in the *Pins* and *Boundary* tabs of this dialog box.

BGA Editor: Component Editing Dialog Box



The BGA Editor automatically:

- Assigns names to pins as you add, move, or copy them to new locations. The BGA Editor bases the new name on the current pin numbering scheme settings and the position within the grid structure.

- Adjusts pins to lie directly on the nearest grid location to maintain consistency of the specified pin pitch and grid structure for the symbol being edited.

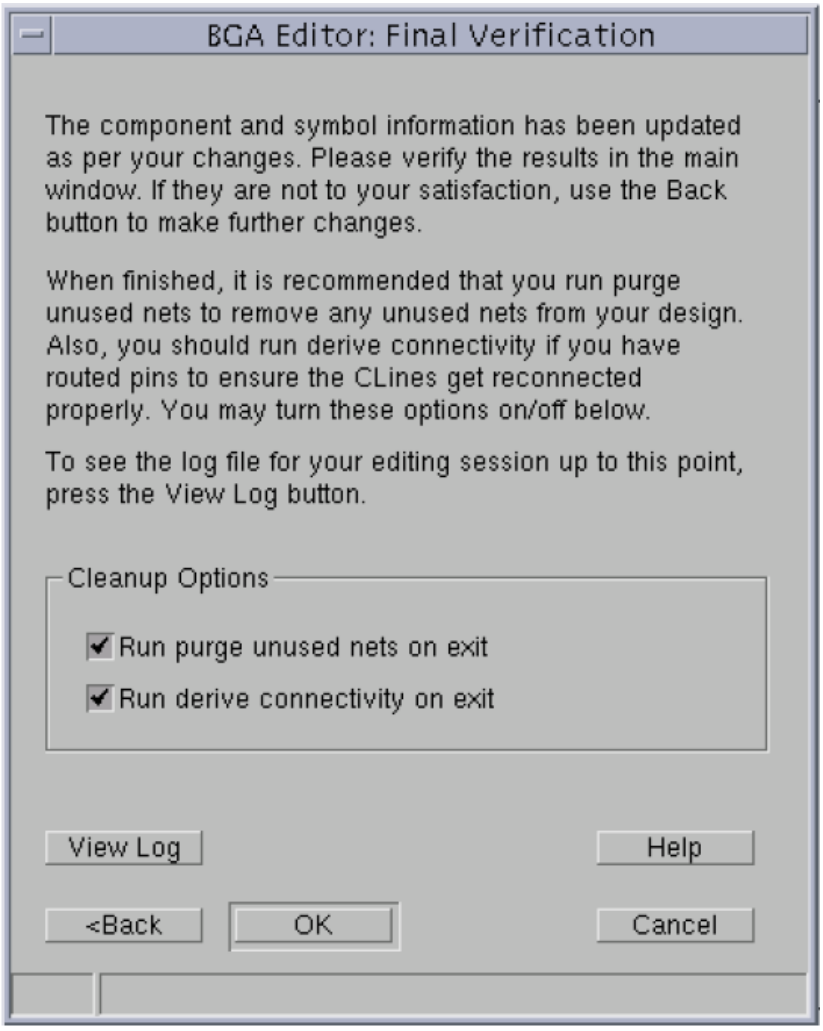
The BGA Editor color codes pins you are editing based on pin use. The default color scheme is based on colors defined in the Color/Visibility dialog box:

Power	Red
Ground	Green
Signal	Yellow
Unspecified/No Connection	Blue

Changing values in the *Pin Numbering* section of the Component Editing dialog box renames all pins in the symbol to match the modified settings. Existing text labels on the borders of the BGA symbol update to reflect these changes only when you finish editing.

When you apply your edits, the BGA Editor presents a final verification dialog box, as shown in the following figure. Cadence recommends that you purge any unused nets and derive connectivity if any routing has occurred. Although available as separate commands in *Logic – Purge Unused Nets* (`purge unused nets` command) and *Tools – Derive Connectivity* (`derive connectivity` command), these functions run automatically if checked in the verification dialog box.

BGA Editor: Final Verification Dialog Box



Capabilities

The following list describes the general capabilities of the BGA Editor:

- Adds, deletes, copies, moves, swaps, and modifies pins
- Allows mirroring and rotation of copy and move selections of items within a group
- Allows grid setting changes (numbering pattern, pitch, offset, and so on) during the active editing session
- Allows definition of multiple grid areas

- Supports pin coloring based on their current pin use
- Displays a heads-up window that provides information on elements over which the cursor is placed. Simultaneously highlights the element that is the object of the display, if desired.
- Supports all pin numbering schemes, including sequential and customized
- Adds a pin number prefix to all pin numbers within the same grid
- Allows pin numbering schemes that ignore unused grid points
- Supports rotated pins
- Supports border pin text creation
- Automatically generates pin number text on every pin in a symbol, offset by a user-defined value
- Provides warnings and back-out options when a grid modification operation might result in pins beings shifted, deleted, or renumbered
- Allows restriction of new pins within a grid
- Allows creation of new symbols and components during an active editing session

Guidelines and Constraints

The BGA Editor is useful for editing BGA symbols only and does not afford full Symbol Editor capability. For other types of symbols, such as a die or a non-BGA type component, use the Symbol Editor mode.

⚠ To edit dies, run the *Edit – Die* (`die editor`) command to invoke the Die Editor.

To ensure that you obtain the optimal results while using the BGA Editor, be aware of the following operating conditions:

- Symbols that you edit with the BGA Editor must have an I/O component class.
- You can edit only one component at a time.
- You cannot edit a symbol if two symbol instances of the same symbol definition exist, because modifications to one symbol instance would propagate to all symbol instances.
- Components that you are editing for the first time and which were *not* created using the BGA Generator initialize with a customized numbering pattern and the grid disabled. This condition does not apply to components created with the BGA Generator.

- You cannot create new nets. You can only place pins on nets that existed prior to invoking the BGA Editor.
- You cannot assign the name of a new pin except in customized mode. The labelling scheme determines all pin names, which the BGA Editor creates, assigns, and maintains.
- Attributes you attach to a pin are not recreated if you delete the pin, then add it back during the editing session *unless* you undo (*Oops*) the delete action.
- You can undo (*Oops*) only the last operation that you performed.
- You cannot add metal, text, keepouts, or lines.
- Pin numbers must be unique, otherwise symbol and component generation fail.
- All pin numbering schemes except *Customized* are controlled by the editor. User-control of numbering values is available only through the *Pin Numbering* section of the Component Editing dialog box.
- You cannot display or hide pin number text on a per-pin or per-grid basis, only at the component level.
- Border text is not available when your pin numbering scheme is set to *Customized*, *Sequential*, if the pin pattern skips unused spaces, or if you use multiple grids.
- Metal connected to pins in the component that you are editing is not affected if you move the pins; that is, not move as a unit with the pins.
- You cannot modify the symbol outline.
- You cannot add mechanical pins.
- You cannot add connect pins at off-grid locations.
- You cannot add more that one pin to the same location.
- Rotated symbols spin back to 0 degrees during the editing session. When you have completed, the rotation value is returned.
- Mirrored symbols are unmirrored during editing.

BGA Editor Use Models

This section describes how you may use the BGA Editor in some common circumstances. These scenarios give only a glimpse of the many use models that you can create pairing good design practices and the flexibility of the BGA Editor.

⚠ These flows are provided as examples only. The procedures associated with them provide instructions on running the editor for these scenarios only.

These flows apply to an existing BGA symbol under one or more of the following conditions:

- The BGA was created with the BGA Generator.
- The BGA was previously edited with the BGA Editor.

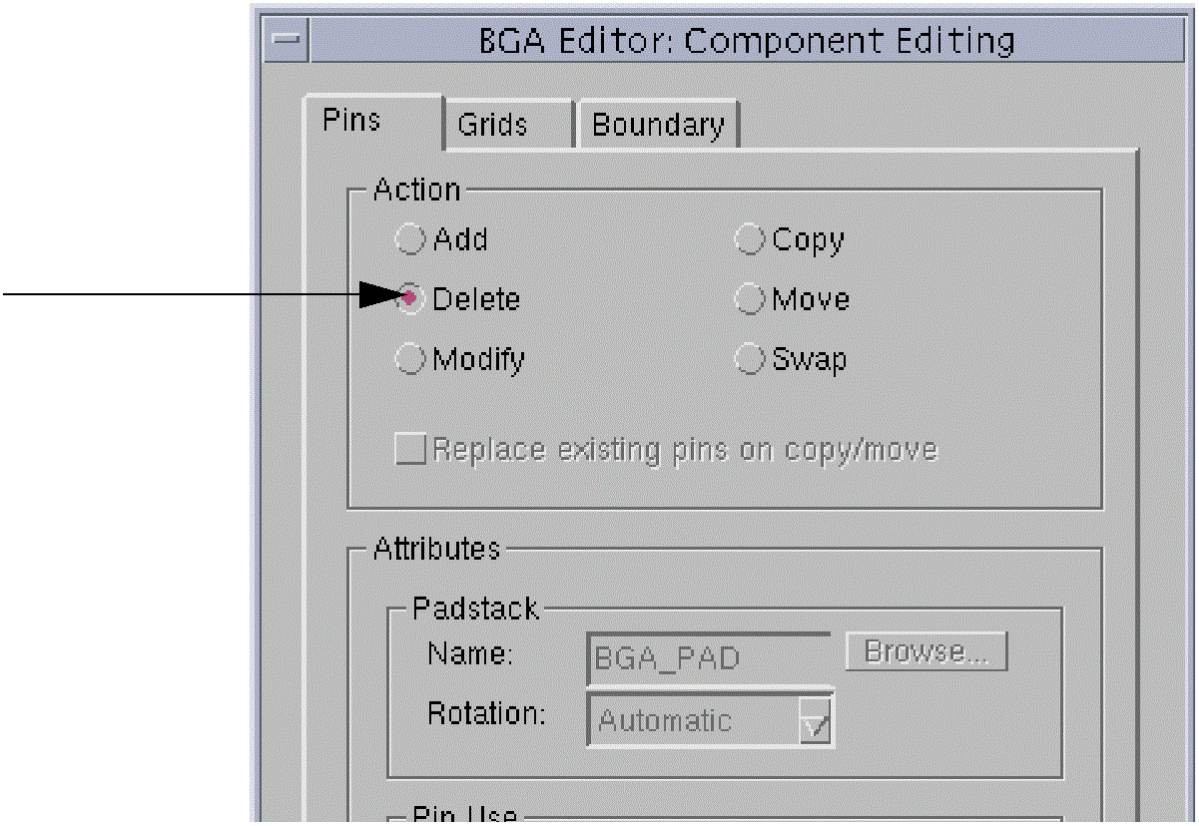
This ensures that a grid structure is in place at the time of the editing session. If not, you must create one.

Goal	Open the file containing the BGA that you want to edit.
User Action	Run *Edit – BGA* (`bga editor` command).
Result	**a.** You are prompted to select the BGA that you want to edit. If the design contains only one BGA, this occurs automatically.
	b. The BGA Editor opens in pin editing mode.

Deleting Balls From the Grid Array

Goal	Remove some balls from the corner areas to improve routability.
User Action 1	Click *Next* in the Component Selection dialog box.
Result	The editor advances to the component editing phase.
User Action 2	Click *Delete* in the *Action* frame of the Component Editing dialog box, as shown in the following figure.

Delete Pins Selection



User Action 3	Either pick individual pins, go into the Temp Group mode, or window around the balls that you want to delete.
Result	The selected pins disappear from the Design Window.
User Action 4	To complete the goal of deleting pins, press *Next* in the dialog box.
Result	The Final Verification dialog box appears, presenting you with further options as described in the dialog box.

Changing the Padstack of the Core Balls

This flow assumes an existing BGA with defined grid is open in the Design Window, and that the BGA Editor is open in pin-editing mode.

Goal	Change the padstack assignment of the core balls in the BGA.

User Action 1	Click *Next* in the Component Selection dialog box.
Result	The editor advances to the component editing phase.
User Action 2	Click *Modify* in the *Action* frame of the Component Editing dialog box.
Result	The *Attributes*, *Pin Use,* and *Net* frames of the dialog box become active.
User Action 3	Window around the core balls to select them for editing.
Result	The pins are highlighted and the *Attributes*, *Pin Use*, and *Net* frames of the dialog box are updated with information based on your selection. (Double asterisks ** in the list fields of the dialog box indicate multiple object types in your selection group.)
User Action 4	In the *Attributes* frame of the Component Editing dialog box, change the padstack, and click *Apply Change*s.
Result	The display of core balls changes to reflect your selection.
User Action 5	To complete the goal of changing padstacks, press *Next* in the dialog box.
Result	The Final Verification dialog box appears, presenting you with further options as described in the dialog.

Assigning a New Net to a BGA Ball

This flow assumes an existing BGA with defined grid is open in the Design Window, and that the BGA Editor is open in pin-editing mode.

Goal	Change the net assignment of a ball in the BGA.

User Action 1	Click *Next* in the Component Selection dialog box.
Result	The editor advances to the component editing phase.
User Action 2	Click *Modify* in the *Action* frame of the Component Editing dialog box.
Result	The *Attributes*, *Pin Use,* and *Net* sections of the dialog box become active.
User Action 3	Click *Item Info* in the dialog box.
Result	The Item Information "heads-up" window appears.
User Action 4	Make sure that you set the item type to *Pins*, then place your cursor over a pin.
Result	The pin number, net, padstack, and location of the pin appear in the window.
User Action 5	Click on the pin to select it for editing. Then, in the *Net* frame of the Component Editing dialog box, make your change and click *Apply Change*s.
Result	The net attachment of the ball changes to reflect your selection.
User Action 6	Confirm that the change occurred by placing the cursor over the ball and reading the new net name in the heads-up window.

Onscreen Error Messages

Onscreen messages from the BGA Editor appear as the event happens. Messages output to the Design Window indicate events requiring user confirmation.

Log File Information Messages

Messages written to the `bga_editor.log` file chronicle the most recent editing session.

Related Topics

- bga text in
- bga generator
- Working with Symbols
- bga editor
- purge unused nets
- derive connectivity
- die editor
- Temp Group

Using the New Design Wizard (Package Layout)

To generate a prototype design for a die or component or both, you can run the New Design Wizard. The wizard guides you through a series of steps that accepts input of required data from various sources and creates a prototype that you can then analyze for signal integrity, size requirements, and other considerations. Routing capability is not included in the wizard.

⚠ The wizard may not accommodate your specific design requirements. It is designed to produce a relatively simple component design; therefore it may not be practical for sophisticated packaging needs.

The New Design Wizard lets you prototype a design using new data that you specify in the Die Generator or Tiling Generator phase of the process, or from existing data that you import from OpenAccess (OA), LEF/DEF, Die text, or design settings from an existing user-defined template drawing. There are two primary design paths that you can follow:

- Die-to-component flow

- Component-to-die flow

Related Topics

- design wizard
- New Design Wizard (Package Layout)

New Design Wizard (Package Layout)

Certain assumptions, inherent in the New Design Wizard, concerning the kind of die or component that you want to make include:

- Die-pin geometry that is either rectangular, square, or circular
- Single-size peripheral pins
- Single-size core pins
- Flip-chip cores containing checkerboard-patterned alternating power and ground bumps
- Round, uniform-size BGA peripheral balls
- Round, uniform-size core balls

Additionally, the following are not supported:

- Multi-chip packages
- Packages with ball-up orientation
- Signal balls in the BGA core
- Importing of GDSII (Stream), DXF, and DIE format files

Die and Component Technology Files

The New Design Wizard creates a new die and component based on values established for your die fabrication technology and component type. To help you enter these values during the wizard process, Cadence provides two generic ASCII text files that you can populate with your data and which the wizard then reads during die and component creation.

`.chipTechnology.txt`

The `chipTechnology.txt` file contains the parameters of your die fabrication technologies. The file information includes:

- Die technology name
- Pin pitch
- Staggered or not staggered
- Pin height and width
- Distance from die edge to pin edge
- Die type (flip-chip or wire bond)

`PkgTypeParms.txt`

The `PkgTypeParms.txt` file contains a fixed list of component types that you can configure by editing the file manually when the wizard is inactive. Changes that you make to component types are applied during subsequent sessions. The file information includes:

- Component type name
- Technology file name
- Ball size and pitch
- Component outline-to-ball clearance
- Die type (flip-chip or wire bond)

Both files are tab-delimited for easy conversion to a spreadsheet, though they work equally well in text format.

⚠ This information is used only for new designs that you are creating with your tool, not as data that you are bringing in from LEF/DEF, Die Text-In, or OpenAccess (OA) files.

The default location of the files is defined by the `TECHPATH` variable, which you can view or modify using the User Preferences Editor. The `TECHPATH` variable resides in the Design_paths category. See the *Setup – User Preferences* (enved) command for details on using this feature.

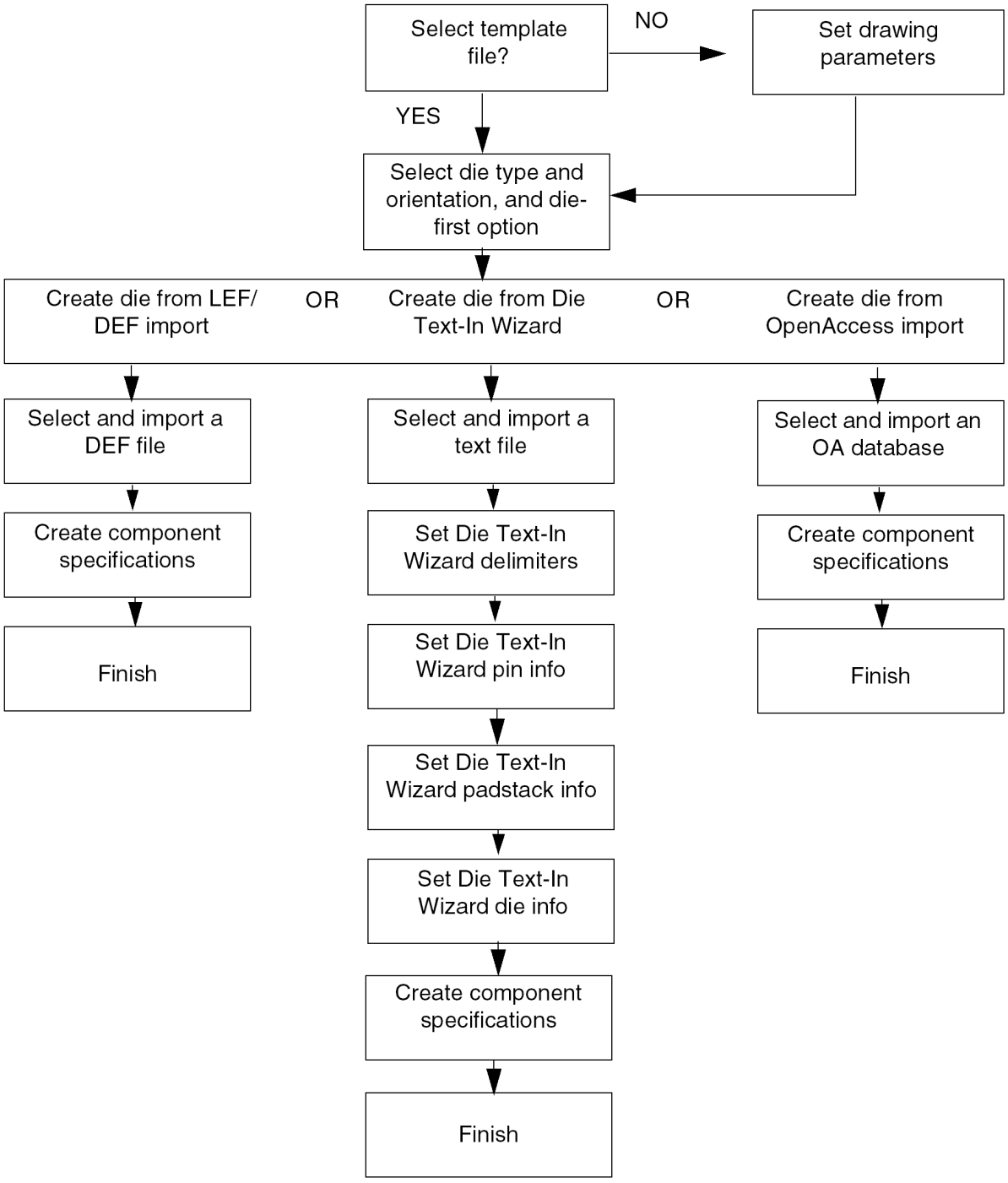
Design Flows

The two primary design paths that you can follow using the wizard are die-to-component (generating the die first) or component-to-die (generating the component first). Additionally, if you create a design using new data, rather than importing existing data from outside your tool, the wizard provides different flows for flip-chip and wire bond dies. The following charts illustrate the various flows, based on your selection of the options provided by the wizard.

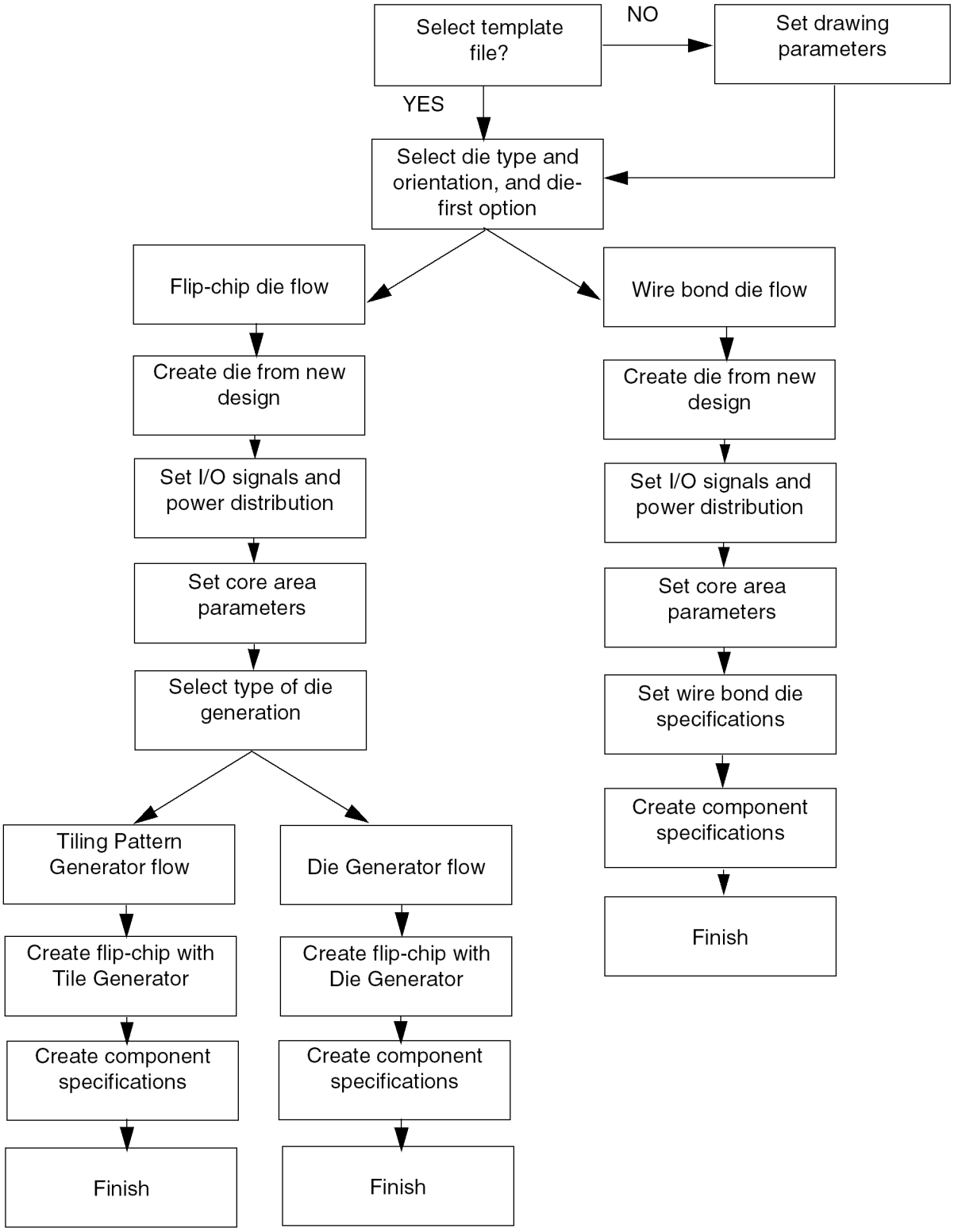
- Die-to-Component Design Flow: From External Data
- Die-to-Component Design Flow: From New Data
- Component-to-Die Design Flow: From External Data
- Component-to-Die Design Flow: From New Data

Die-to-Component Design Flow: From External Data

This figure shows the flow when creating a die from LEF/DEF, Die Text-In, or OA files.

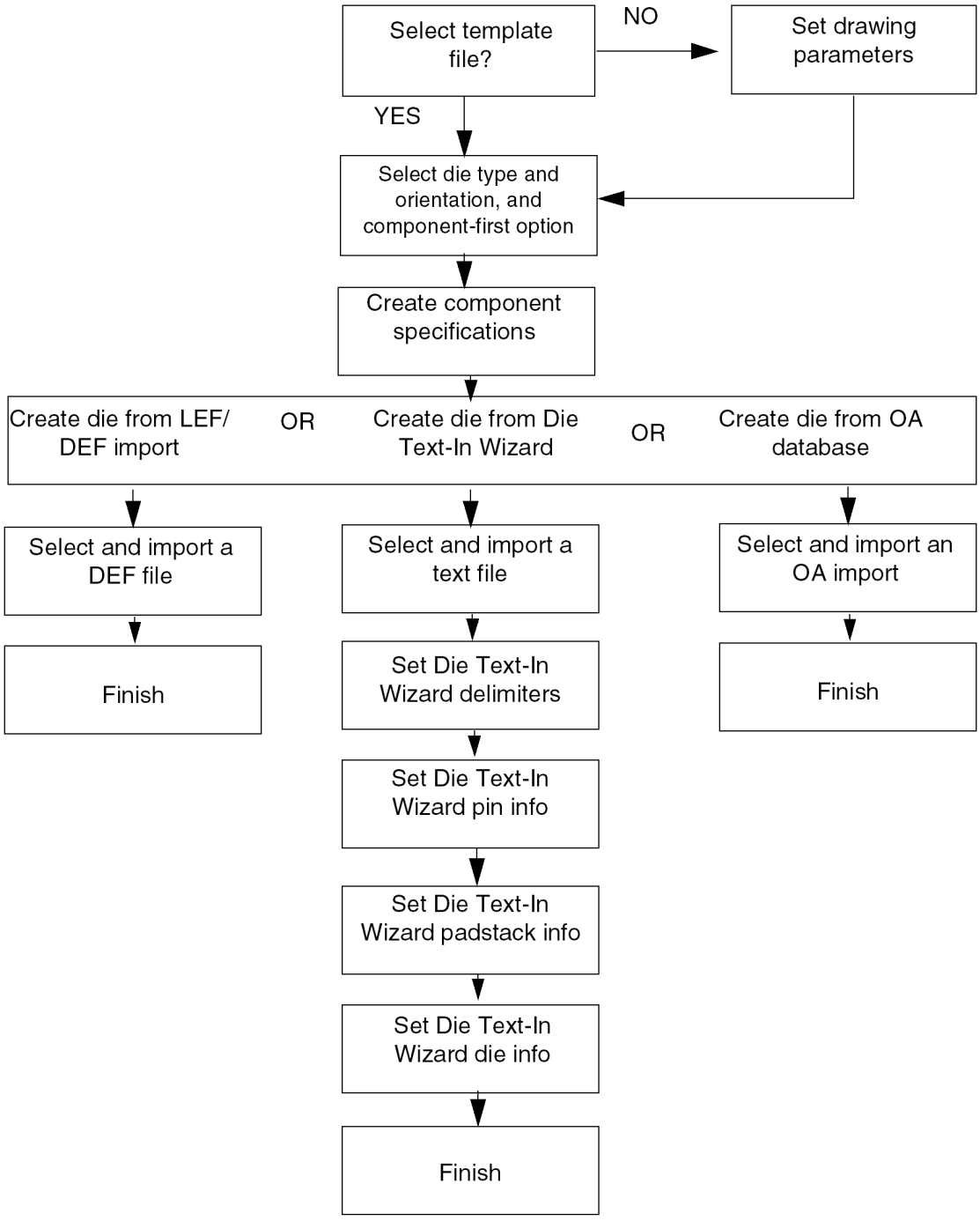


Die-to-Component Design Flow: From New Data

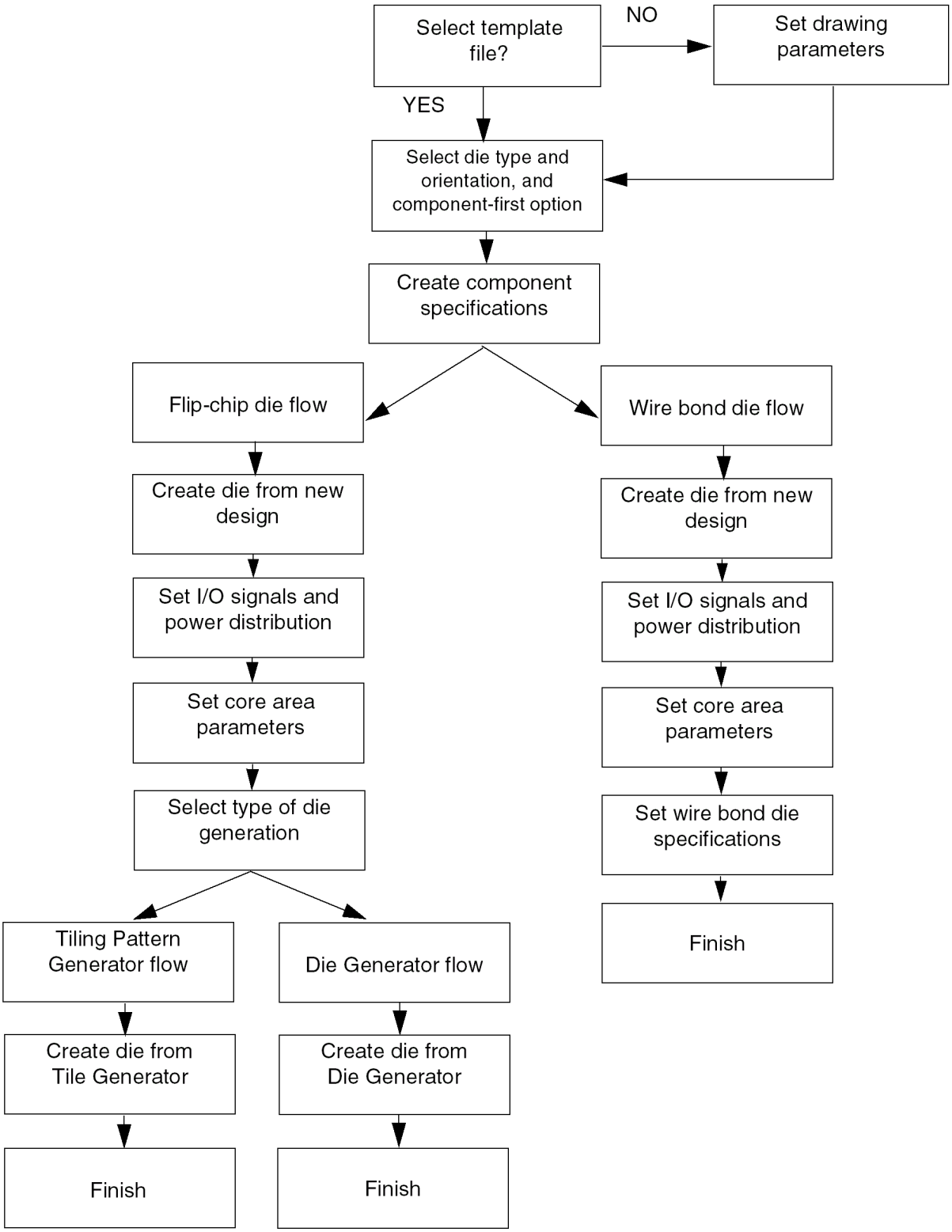

Select template file?
NO
Set drawing parameters
YES
Select die type and orientation, and die-first option
Flip-chip die flow
Wire bond die flow
Create die from new design
Set I/O signals and power distribution
Set core area parameters
Select type of die generation
Tiling Pattern Generator flow
Die Generator flow
Create flip-chip with Tile Generator
Create flip-chip with Die Generator
Create component specifications
Create component specifications
Finish
Finish
Create die from new design
Set I/O signals and power distribution
Set core area parameters
Set wire bond die specifications
Create component specifications
Finish

Component-to-Die Design Flow: From External Data

This figure shows the flow when creating a die from LEF/DEF, Die Text-In, or OA files.



Component-to-Die Design Flow: From New Data


Select template file?
NO
Set drawing parameters
YES
Select die type and orientation, and component-first option
Create component specifications
Flip-chip die flow
Wire bond die flow
Create die from new design
Create die from new design
Set I/O signals and power distribution
Set I/O signals and power distribution
Set core area parameters
Set core area parameters
Select type of die generation
Set wire bond die specifications
Tiling Pattern Generator flow
Die Generator flow
Finish
Create die from Tile Generator
Create die from Die Generator
Finish
Finish

OpenAccess Database

APD requires an interface to the OpenAccess (OA) database so that the Cadence I/O Planner (IOP) and IC tools can communicate with one another. As a packaging tool for ICs, APD requires a physical description of the IC outline, and the I/O pad sizes and locations. IOP also requires a physical description of the IC outline and I/O pad sizes and locations. Additionally, it needs the I/O cell placement (location and orientation) information. The OA interface provides a method to import this die information from IC design tools into APD.

⚠ Currently, this feature supports only die information and how the die interacts with the component.

OA is an open database. The OA data format is a binary format representing the IC library, technology, and design data. These binary files are presented in a hierarchical directory structure. For example, each cell (macro) definition is represented as a separate binary file in the library directory structure.

You can install OA in one of two ways. You can install the shared libraries and binaries as part of the Cadence product installation. For additional information, see the *OpenAccess Installation and Configuration Guide*. Alternatively, you can download OA from the Si2 website, `www.openeda.si2.org`. In either case, all the tools in your flow should use the same OA installation.

Before you import an OA database, you should have received a directory structure and files from the IC designer. You must set up the LEF Library Manager to correctly map to the Library Exchange Format (LEF) files supplied by the IC designer. This allows APD to interpret the cells used by the IC design. If you do not have LEF files, use the `oa2lef` tool to convert the library cell into LEF format. You can find this tool in the area where you installed the runtime version of OA. The tool is located in the `OA_HOME/bin`. directory.

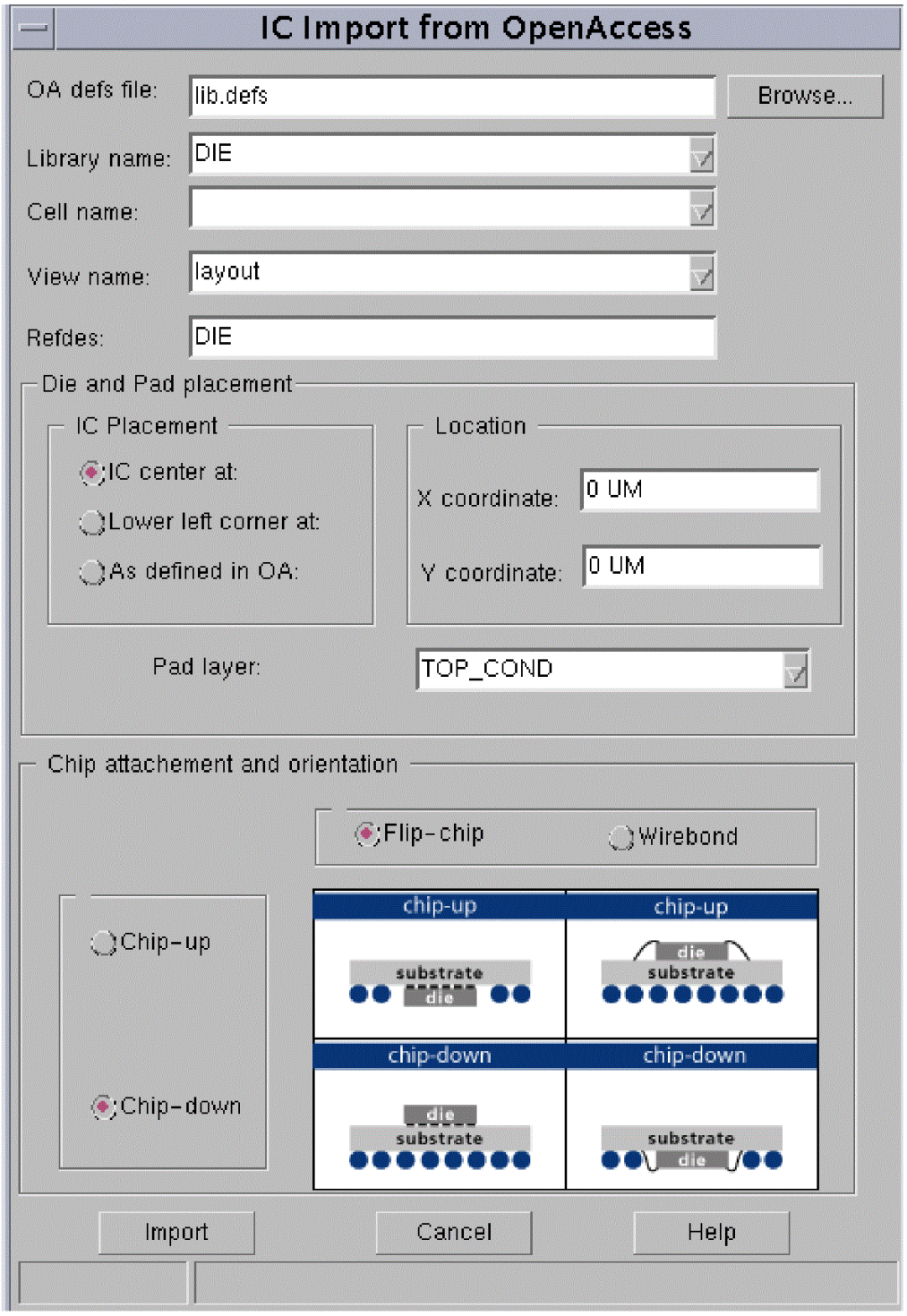
If you do not have an OA database, but you do have a DEF file, use the `def2oa` tool to convert the DEF file to an OA database. You can find this tool in the area where you installed the runtime version of OA. The tool is located in the `OA_HOME/bin`. directory. If you have a DEF file that contains pin/bump and driver information, use the `def in` and then the `oa out` commands to ensure that your OA database contains all the information from the DEF file.

Also be sure to check the `lib.defs` file. If the file contains absolute paths to the library files, update the `lib.def`s file to use relative paths so that the tool can access the library files.

IC Import

You use the `oa in` command to import data from an OA database into APD. The following figure shows the IC Import from OpenAccess dialog box that appears when you run this command.

IC Import from OpenAccess Dialog Box



Log File

During the importing of information from an OA database, the tool generates the following messages and logs them to the `oaImport.log` file.

Message	Additional Information
`Duplicate Macro name found. 'Macroname' defined in 'filename1' and 'filename2'. The macro defined in 'filename1' was used.`	This error appears in the log file if the tool finds duplicate macro names in a library. APD records the macro name and the LEF files in which they are found. It uses the macro in the LEF file that appears first in the library list.
`Before importing an OA database file, The tool displays a message indicating the path and filename of the file to be imported.`	This allows you to associate a specified file with the error message.
`The drawing size is <width> X <height>.`	Informational only.
`This version of OA database is not supported. Convert the database using the oa20to22 tool or similar tool and try again.`	You can find this tool in the area where the you installed the runtime version of OA. The tool is located in the `OA_HOME/bin` directory.
`The OA database has non-recoverable errors. Check the database file and try again.`	Run the `oa2def` tool. You can find this tool in the area where you installed the runtime version of OA. The tool is located in the `OA_HOME/bin` directory. If the `oa2def` tool does not generate errors, the database is correct.
`The IC size is <width> X <height>.`	Informational only.
`The drawing size is smaller than the IC size.`	Increase the drawing size and start over again.

`Macros lacking PINs were encountered but ignored. Only Macros containing PINs are candidates for processing.`	Verify that the macros are correct. You may want to discuss this matter with the IC designer. If you are familiar with the Condensed Macro Library (`.cml`) and LEF files, check the files.

Command Line Messages

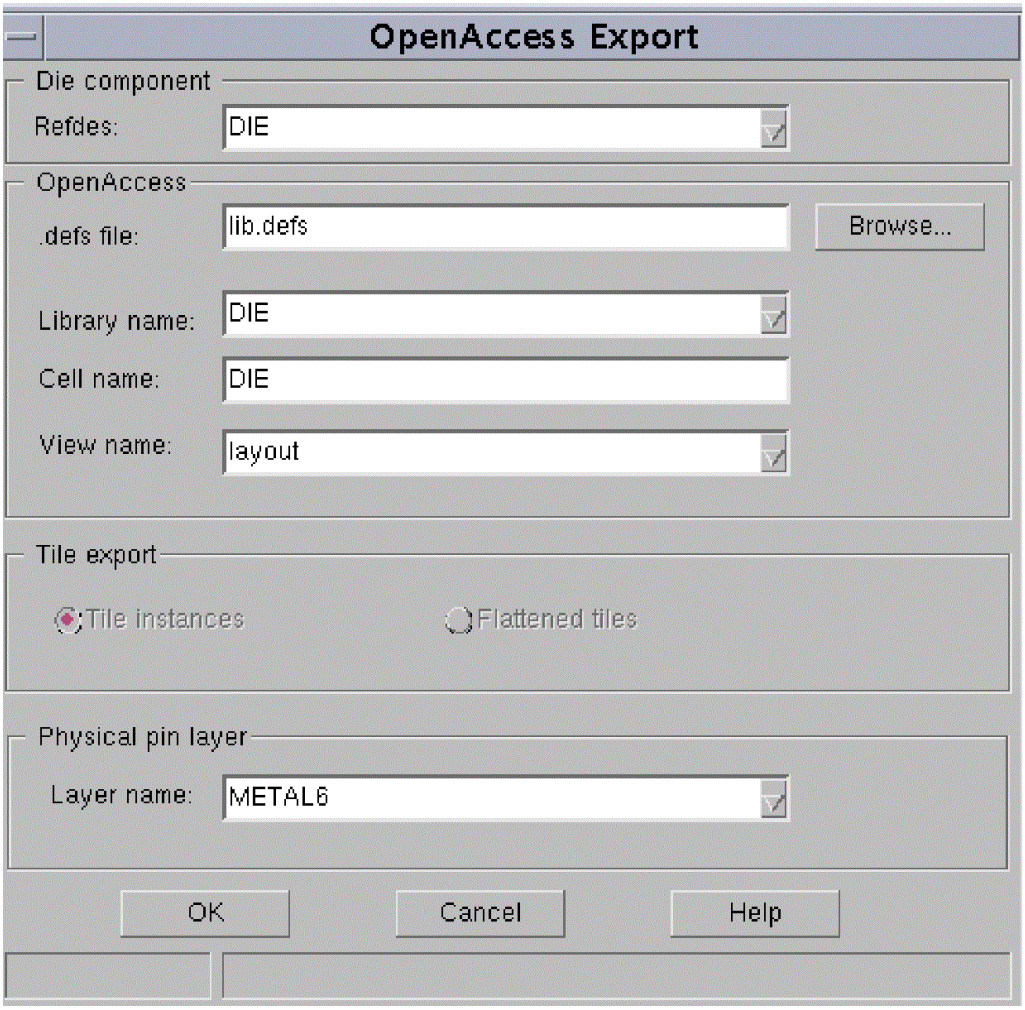
APD generates the following messages at the console window prompt:

```
This version of OA database is not supported. See Log File for details.

The OA database has non-recoverable errors. See Log File for details.

The drawing size is smaller than the IC size.
```

IC Export

You use the `oa out` command to export die data from APD to an OA database. The following figure shows the OpenAccess Export dialog box that appears when you run this command.

OpenAccess Export Dialog Box



Log File

During the export of die information to the OA database, APD generates messages and logs them to the `oaExport.log` file. Information includes the die name that you are exporting, cell name, view name, library name, and specified CML files. The tool also records the date and time.

APD generates the following messages in the log file and at the console window prompt:

```
Warning: No IC Component found in the Design. There is nothing to export. This error
occurs if there are no IC symbols in the design.

There is an error in writing OA data. The directory or files may be write protected.
```

Related Topics

- oa in
- oa out

2

APD: Generating Co-Design Die

Co-design of silicon, package, and board becomes essential as each piece of the system increases in complexity. In many cases, it costs more to develop the package for a given die than it does to fabricate the die itself. As a result, it may be important to know early in the process that the die is constructed to minimize the cost of the package later on. You need to be able to access the package design to ensure that the die can be manufactured and packaged, while meeting all high-speed electrical constraints.

If you can view the die in its proposed package in an early feasibility state, you can change the die pin layout before the final IC design. By understanding the packaging requirements and how the arrangement of the pins affect the package design, the IC designer can help minimize package costs. By looking at the Virtual System Interconnect (VSIC) model for the chip and package together early in a co-design process, you can rule out earlier design options that cannot meet the electrical or high-speed signal integrity constraints, thus enabling a faster time to market for the entire project. At the same time, you need to consider the floor plan and I/O placements inside the IC. When the feasibility phase ends and the project enters the design stage, you can use the information from the preliminary feasibility as a reference or for seeding the production design of both the IC and package.

A co-design die is a die designed with its end package to ensure that the combination of die and package meets all design requirements while at the same time minimizes the overall cost of production. Layout and IC tools work together to support co-design.

The ideal tool for co-design allows the designer to:

- Visualize the IC in the context of the package and see where a change to one impacts the design of the other.
- Manage the system-level netlist for the overall design with respect to the local netlists at the package and individual IC level. This involves the mapping of signal names from the master netlist to the specific names at each level of the design.
- Estimate the timing and signal integrity of connections from driver models, through the IC RDL routing and the package routing, to the driver pin of another IC.
- Update die pin interface change requests from package to IC and IC to package.

About Co-Design with APD

APD support two types of co-design environments:

- Concurrent (dynamic)
- Distributed

Concurrent Co-Design Environment

In the Concurrent Co-Design environment, you use Cadence I/O Planner (IOP), an IC layout tool that provides an environment in which you can edit the IC layout at the I/O cell level and above. When you edit a die in APD in the concurrent co-design environment, the actions are validate in real-time with IOP. APD interacts either indirectly with IOP by importing a co-design die (an existing OpenAccess database), or directly, by involving an active IOP session and bringing in designs from DEF or Verilog sources.

Part of the functionality added by this feature supports the front-to-back flow using System Connectivity Manager (SCM) for logic design and the APD for physical layout design.

In addition, changes to various commands support forward and backward annotation of logic changes between the front-end logic design tool, System Connectivity Manager (SCM), and the layout tool for both co-design dies and the actual package BGA. To support the concurrent co-design environment, the die and BGA import and export mechanisms support logical pin name to physical pin number mapping and pin swap. Also, the logic manipulation commands of the layout tools make physical pin number assignments using swapping logical pin names, rather than by reassigning nets to logical pins.

Distributed Co-Design Environment

In addition to Concurrent Co-Design, APD supports *Distributed Co-Design*.

Distributed Co-Design is the process of creating or updating a co-design die using a die abstract file. The process typically involves a APD design owned by a designer on one computer network and an IC design owned by a different owner on a different network, perhaps even in a different company. The die abstract file lets designers exchange die information between Innovus and APD. APD can add a co-design die to the layout by reading in the die abstract file generated by Innovus. Then after editing the co-design die in APD, APD designer can export a die abstract file with the updated die information, which, in turn, is imported into EDI Systems. Thus there is no direct interaction with I/O Planner. Once you create a co-design die using a die abstract file, you can only update the die using a die abstract file.

The die abstract file contains information on cell libraries, netlist, port names, pins, and driver

instances. It is attached to the APD database during an add co-design operation. The file does not contain any die shrink information. When updating the co-design in the Die Editor, the die abstract file, stored in the database, is replaced with an updated one.

Distributed co-design is available on all Cadence supported platforms.

⚠ Distributed Co-Design is not supported in SCM.

Related Topics

- Co-Design Die Creation
- Distributed Co-Design Environment

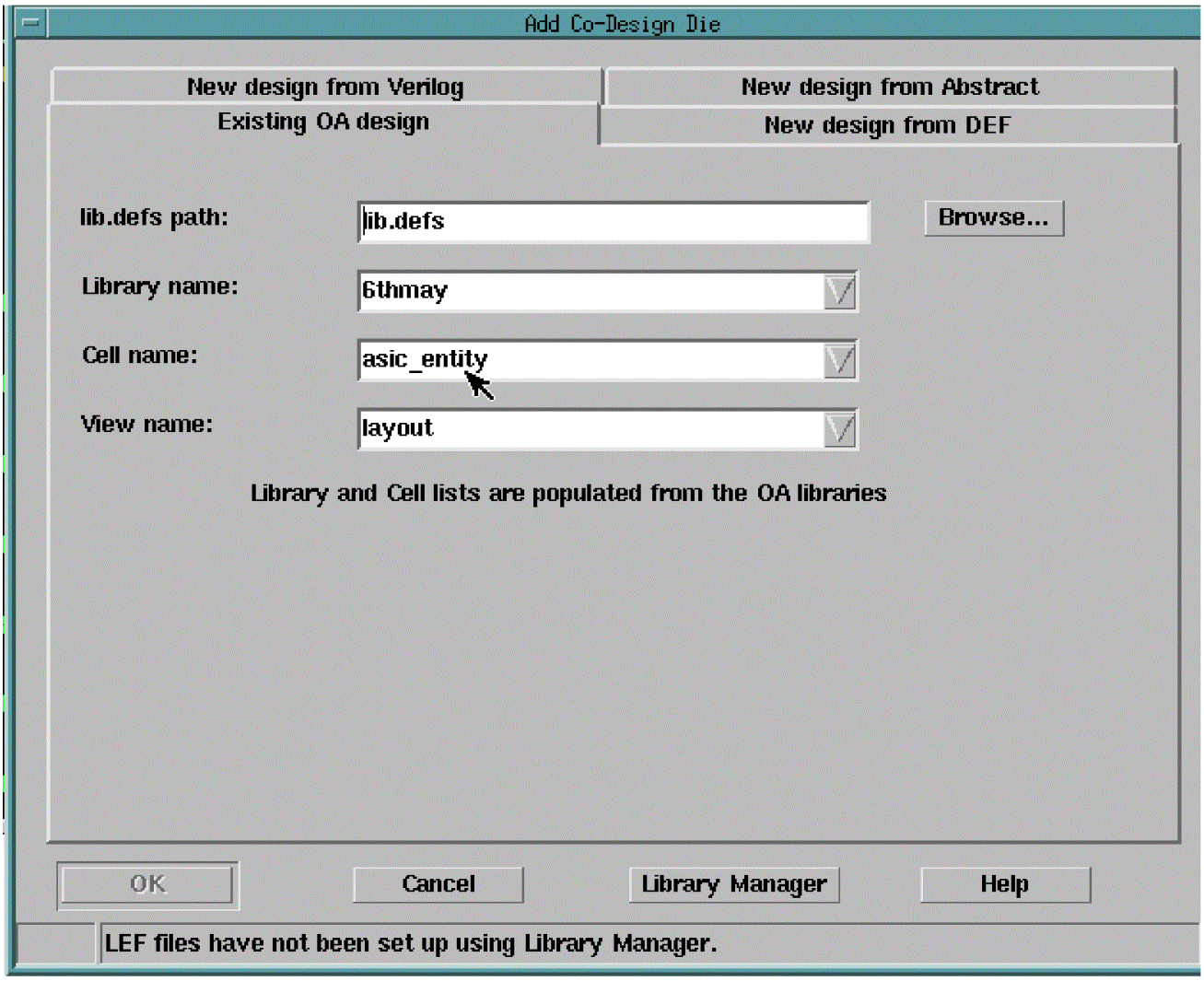
Co-Design Die Creation

shows the Place Co-Design dialog box, which

Concurrent Co-Design Environment

From APD, choose *Add – Co-Design Die* (`add codesign die` command) from the menu to create or add existing co-design dies to a package. The Add Co-Design Die dialog box appears as shown in the following figure.

Add Co-Design Die Dialog Box for Concurrent Co-Design



You can choose an OpenAccess (OA) database which already exists on disk, or specify a new OA database for a new co-design die. This co-design die is then directly imported into the active `.mcm` design without opening an IOP window. You specify the OA database by locating an OA library definition file, and then choosing a library, cell and view from the library list. The specified cell view must contain an IC layout written by IOP.

> ⚠ When adding or editing a co-design die, if your DRC constraints values are not properly configured, this could result in many DRC errors being flagged, which may not be actual errors. You can set the `codesign_delay_drc_checks` variable under Codesign in the IC_Packaging category of the User Preferences Editor to delay DRC checking.

You can also specify DEF or Verilog files to create co-design dies. If you specify either of these files.

Distributed Co-Design Environment

To add a co-design die in the Distributed Co-Design Environment, the layout designer needs a die abstract generated by EDI Systems.

Using EDI Systems, the IC designer loads the design using LEF or Verilog files, places the I/O drivers, and creates a bump array and assigns nets to the bump array if the die is a flip chip die. Finally the designer generates the die abstract file. To generate the die abstract file from EDI systems, use the `writeDieAbstract` command with the syntax:

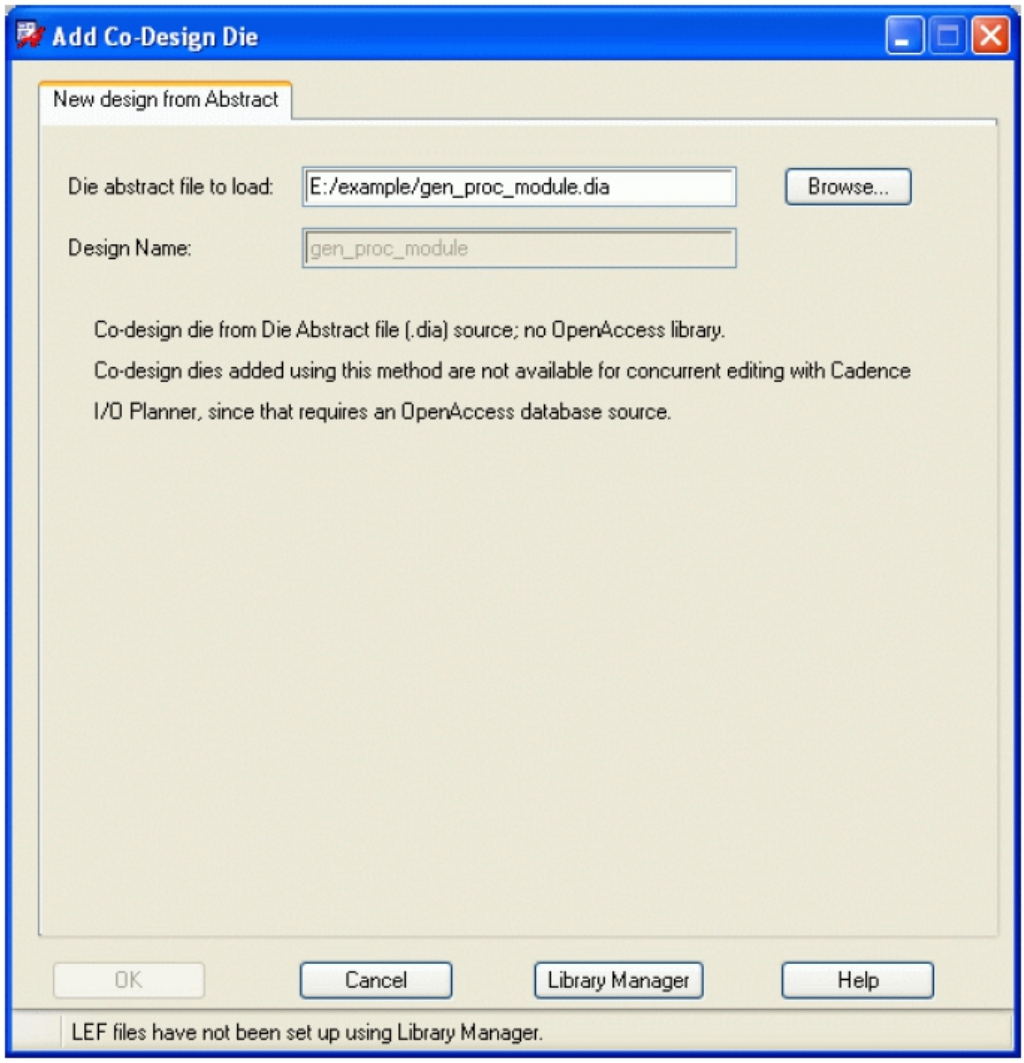
```
writeDieAbstract die_file_name
```

For example, to create a die file named `my_die_abs.dia`:

```
writeDieAbstract my_die_abs.dia
```

When adding a the co-design die in APD, the layout designer imports the die abstract file and places it in the layout design. The Add Co-Design Die dialog box appears as follows.

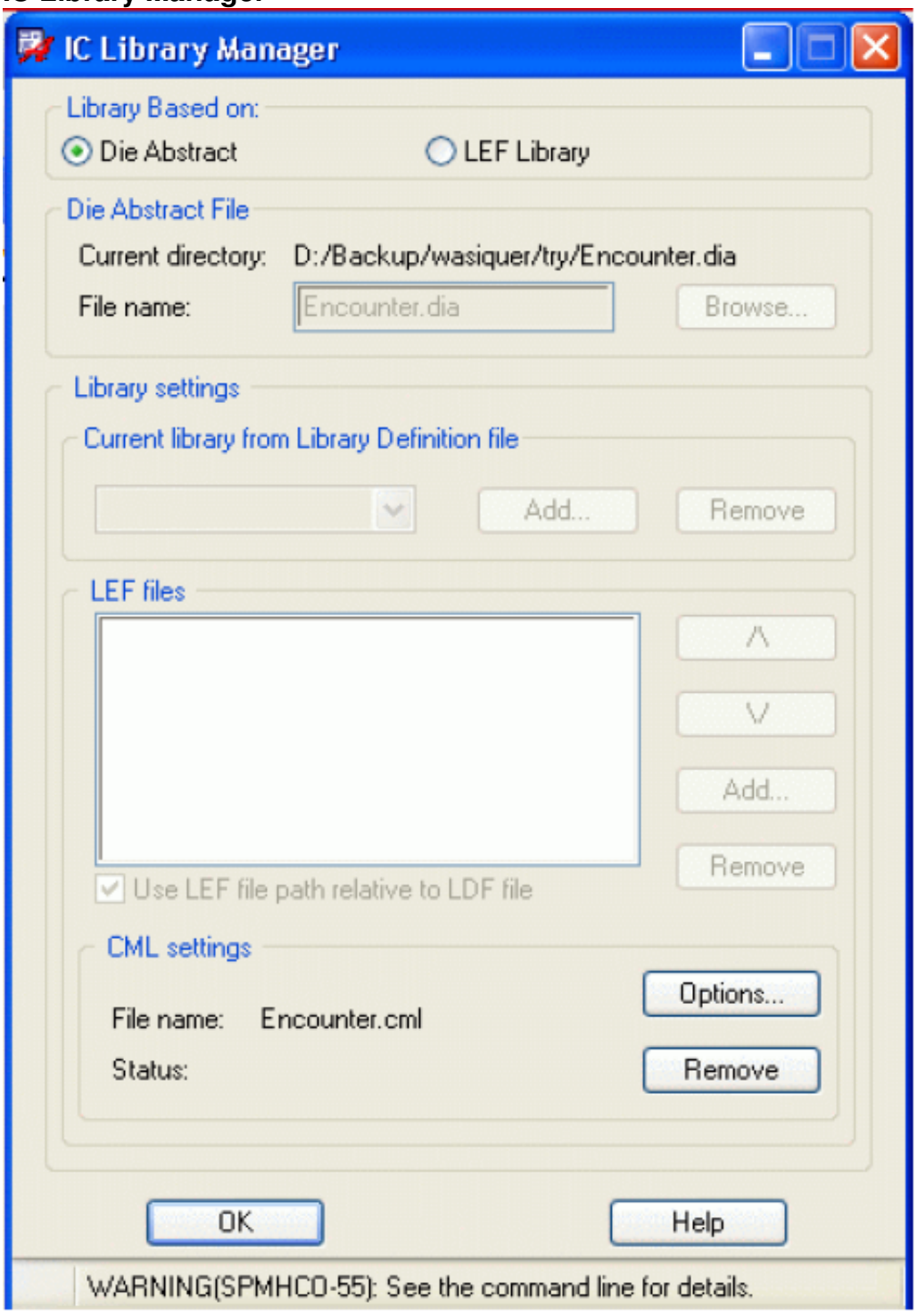
Add Co-Design Dialog Box for Distributed Co-Design



⚠ With the support of OpenAccess (OA) on the Windows platform, you can also import an existing OA design in the Distributed environment. no

The die abstract file contains all library information and can be used directly. The file contains information for all layers for each pin of each macro. You can also add CML files if needed. The CML files created from the *IC Library Manager* can be generated either using LEF files or from the die abstract file itself. To do so, open the IC Library Manager dialog box, and specify the library settings. You need to add a Library Definition file if you select the *LEF Library* option.

IC Library Manager



⚠ The Library Settings and LEF Files sections are not available if you select Die Abstract in the Library Based on section.

Die Tasks in the I/O Planner

If you specify DEF or Verilog files to create co-design dies, the I/O Planner opens.

You can:

- Place I/O drivers
- Set die size
- Create a bump array if the die is a flip-chip die
- Assign nets to the bump array if it is a flip-chip die.
- Update the package

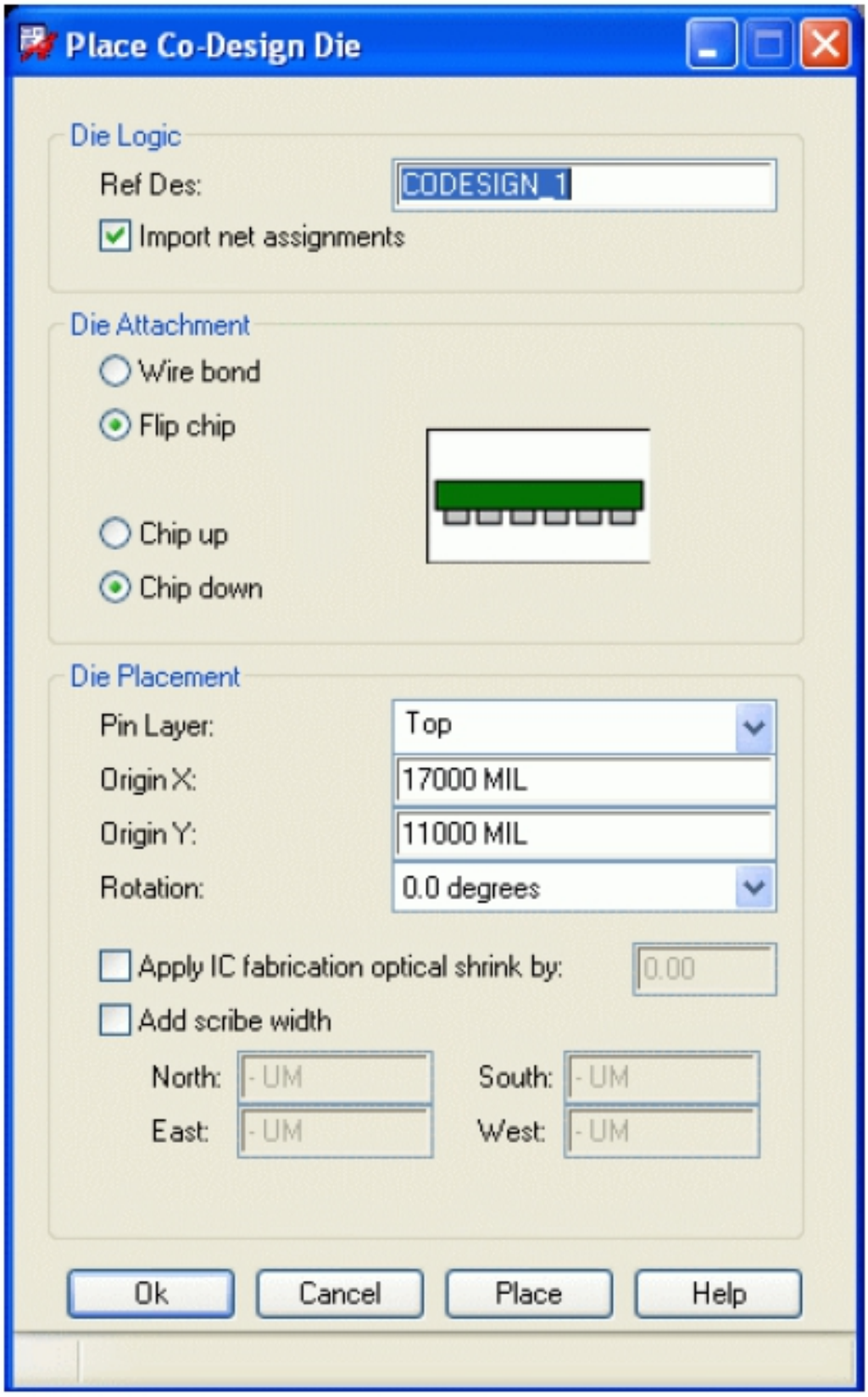
See the *Cadence I/O Planner Application Note* for additional information.

Placement in APD

Once you add a co-design in either a Concurrent Co-Design or Distributed Co-Design environment, you place the co-design in APD. the figure below shows the Place Co-Design dialog box, which lets you place an existing OA design as a co-design die or place a new co-design die.

You can also apply scribe lines and an optical shrink to the imported die when you place the die.

Place Co-Design Dialog Box


Place Co-Design Die
Die Logic
Ref Des:
CODESIGN_1
Import net assignments
Die Attachment
Wire bond
Flip chip
Chip up
Chip down
Die Placement
Pin Layer:
Top
Origin X:
17000 MIL
Origin Y:
11000 MIL
Rotation:
0.0 degrees
Apply IC fabrication optical shrink by:
0.00
Add scribe width
North:
- UM
South:
- UM
East:
- UM
West:
- UM
Ok
Cancel
Place
Help

When importing a co-design die (whether directly by bringing in an existing IOP database, by running `update package` from the IOP for the first time, or by a die abstract file), APD generates the `import_codesign_die.log` file. It generates this file anytime a co-design die is added or updated in the layout database.

Related Topics

- Co-Design Die Editing

Co-Design Die Editing

Package-Driven I/O Planning

With Release 16.3, the Die Editor provides the Package-Driven I/O Planning capability when you edit a co-design die. You access this capability when you run the Die Editor (choose *Edit – Die* or run the `die editor` command). Package-Driven I/O Planning includes a set of commands that lets you view the I/O drivers of a co-design die (wire bond or flip-chip) in APD. You can perform pin and driver operations within APD, and then have those operations confirmed or adjusted by the I/O Planner, which has access to IC process technology information. This minimizes the need to learn IOP by providing many die I/O planning capability within the package design environment. This also maximizes package routability by optimizing the die I/O bump array and pad ring.

The Package-Driven I/O Planning capability is available in both co-design environments in Release 16.3:

- Concurrent Co-Design Environment
  Available with Innovus Digital Implementation System, Innovus Advanced Node GXL, Allegro X Advanced Package Designer
- Distributed Co-Design Environment
  Available with any IDIS installation, Allegro X Advanced Package Designer

When using the Package-Driven I/O Planner capability, it is assumed that you initially placed and sized your die with I/O Planner, that is, there is at least a placed extents rectangle in the design, before you can edit this type of co-design die in APD.

APD depends on I/O Planner to generate the exact coordinates of objects in the die. Any coordinates generated by APD are approximations used to issue commands, but the final coordinates are generated by I/O Planner. Drivers shown outside the die extents in I/O Planner are *unplaced* and are not displayed when the Die Editor is running. However, they do appear in the list of unplaced I/O driver cells and you can place them in APD.

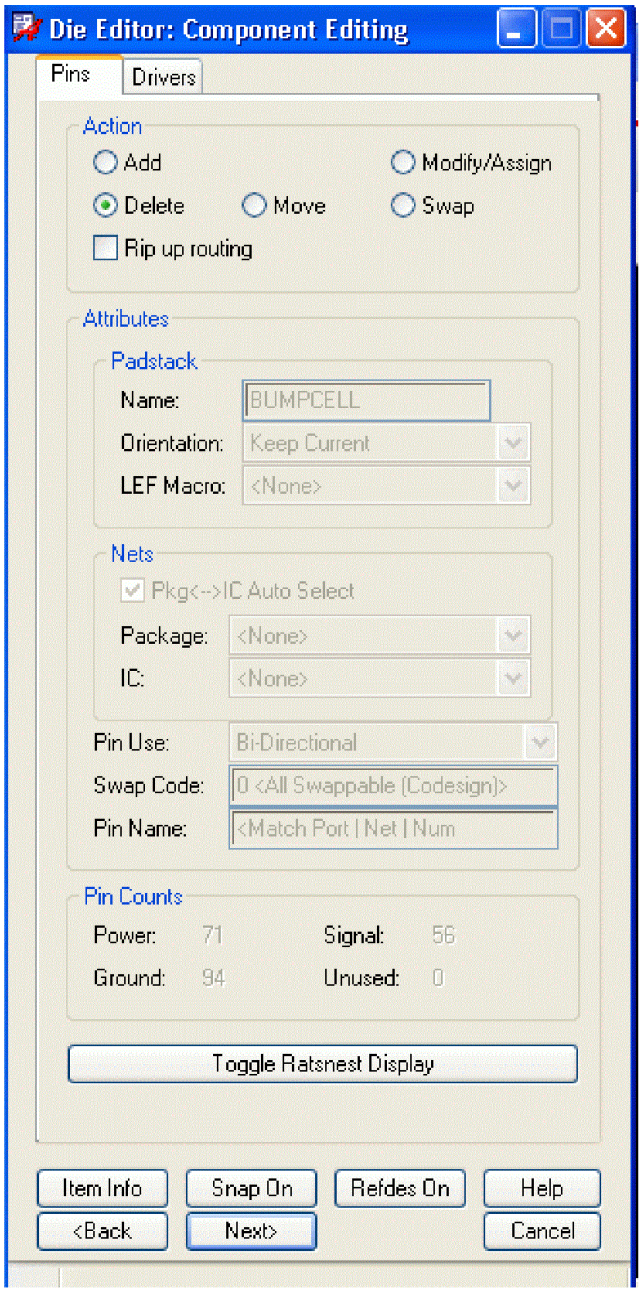
⚠ APD relies on I/O Planner for adherence to the IC design rules.

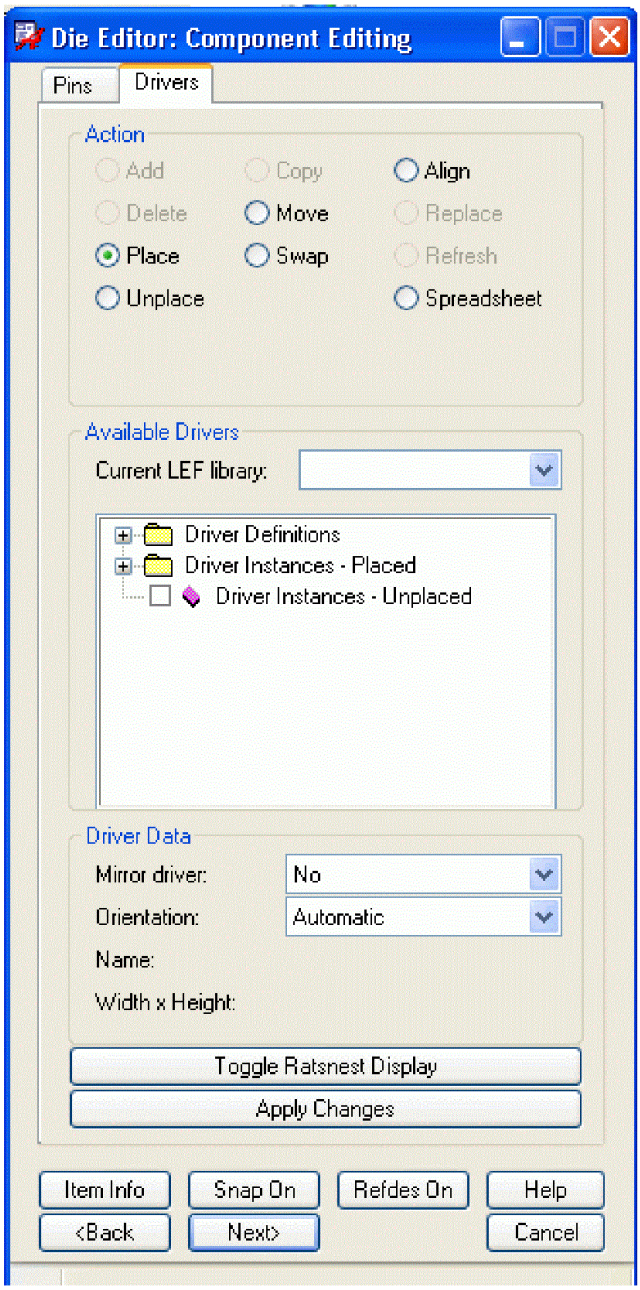
Running the Die Editor

Once you run the Die Editor and select the co-design die for editing, the Die Editor opens with the *Pins* and *Drivers* tabs activated (The *Drivers* tab is available only when editing a co-design die). The APD Design Window is updated with a display of I/O objects just as they appear in the I/O Planner.

⚠ You can click the Library Manager button to specify library information for the source file. Otherwise, library information specified during previous edit or add operation will be used.

Pins and Drivers Tabs Available with Package-Driven I/O Planning

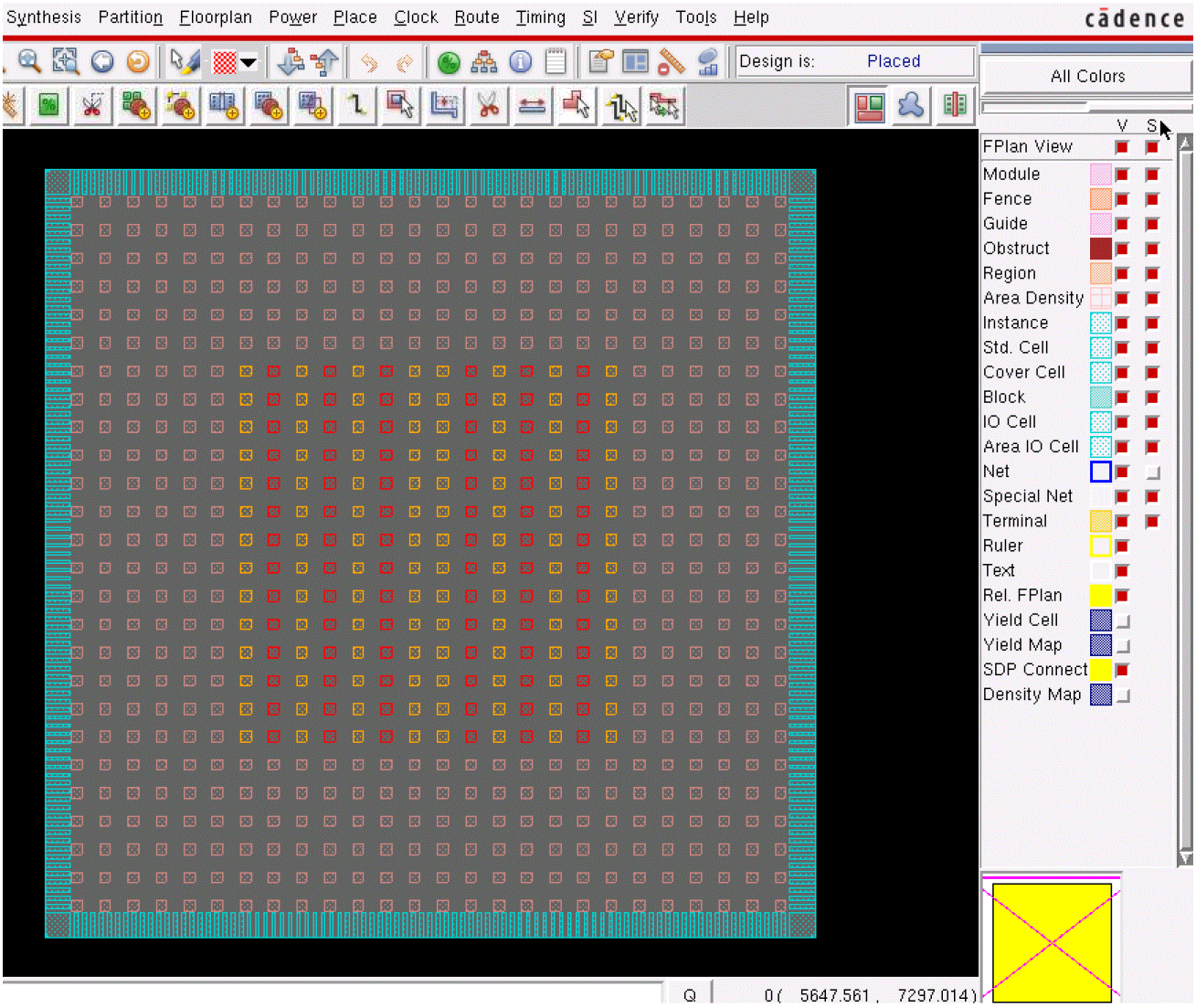




In a Concurrent Co-Design environment, the I/O Planner Window also appears. If the I/O Planner was already running, the Die Editor associates with I/O Planner already running. You can use the I/O Planner window to perform RDL routing.

⚠ If you make changes in IOP while using Die Editor in APD in the Verilog ECO flow, you must synchronize the changes from IOP. Use the `updatePackage` command in IOP to synchronize before exiting the Die Editor in APD.

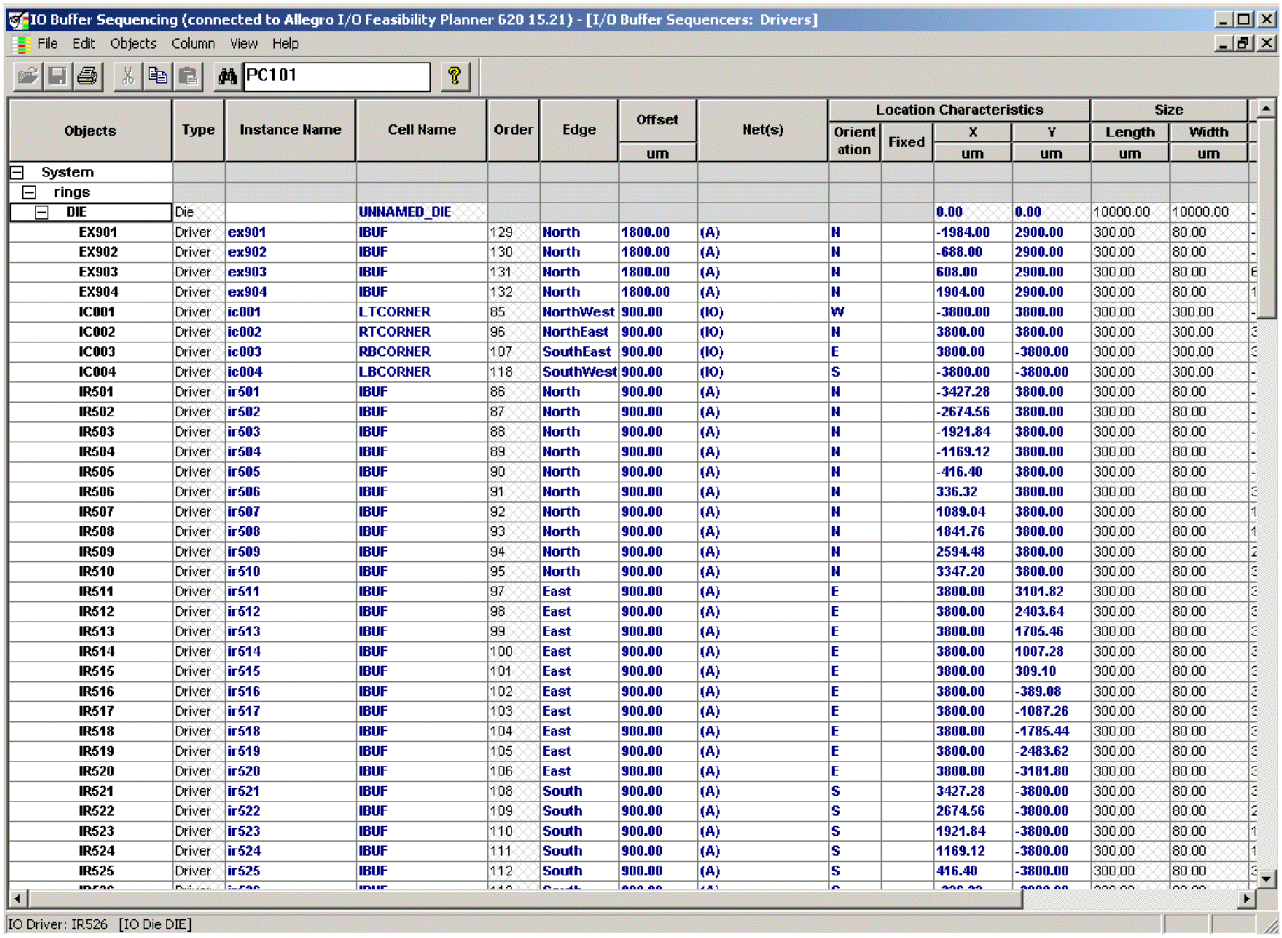
I/O Planner Window – Concurrent Co-Design Environment



Modifying Pins and I/O Objects in APD

You can add, move, swap, and delete pins in APD. You can also place, unplace, move, swap, and align I/O drivers. Additionally, you can display the built-in
I/O Buffer Sequencing Spreadsheet from the *Drivers* tab of the Die Editor; this spreadsheet lets you manipulate the I/O drivers.

I/O Buffer Sequencing Spreadsheet



Objects	Type	Instance Name	Cell Name	Order	Edge	Offset	Net(s)	Location Characteristics				Size	
								Orientation	Fixed	X	Y	Length	Width
						um				um	um	um	um
System													
rings													
DIE	Die		UNNAMED_DIE							0.00	0.00	10000.00	10000.00
EX901	Driver	ex901	IBUF	129	North	1800.00	(A)	N		-1984.00	2900.00	300.00	80.00
EX902	Driver	ex902	IBUF	130	North	1800.00	(A)	N		-688.00	2900.00	300.00	80.00
EX903	Driver	ex903	IBUF	131	North	1800.00	(A)	N		608.00	2900.00	300.00	80.00
EX904	Driver	ex904	IBUF	132	North	1800.00	(A)	N		1904.00	2900.00	300.00	80.00
IC001	Driver	ic001	LTCORNER	85	NorthWest	900.00	(IO)	W		-3800.00	3800.00	300.00	300.00
IC002	Driver	ic002	RTCORNER	96	NorthEast	900.00	(IO)	N		3800.00	3800.00	300.00	300.00
IC003	Driver	ic003	RBCORNER	107	SouthEast	900.00	(IO)	E		3800.00	-3800.00	300.00	300.00
IC004	Driver	ic004	LBCORNER	118	SouthWest	900.00	(IO)	S		-3800.00	-3800.00	300.00	300.00
IR501	Driver	ir501	IBUF	86	North	900.00	(A)	N		-3427.28	3800.00	300.00	80.00
IR502	Driver	ir502	IBUF	87	North	900.00	(A)	N		-2674.56	3800.00	300.00	80.00
IR503	Driver	ir503	IBUF	88	North	900.00	(A)	N		-1921.84	3800.00	300.00	80.00
IR504	Driver	ir504	IBUF	89	North	900.00	(A)	N		-1169.12	3800.00	300.00	80.00
IR505	Driver	ir505	IBUF	90	North	900.00	(A)	N		-416.40	3800.00	300.00	80.00
IR506	Driver	ir506	IBUF	91	North	900.00	(A)	N		336.32	3800.00	300.00	80.00
IR507	Driver	ir507	IBUF	92	North	900.00	(A)	N		1089.04	3800.00	300.00	80.00
IR508	Driver	ir508	IBUF	93	North	900.00	(A)	N		1841.76	3800.00	300.00	80.00
IR509	Driver	ir509	IBUF	94	North	900.00	(A)	N		2594.48	3800.00	300.00	80.00
IR510	Driver	ir510	IBUF	95	North	900.00	(A)	N		3347.20	3800.00	300.00	80.00
IR511	Driver	ir511	IBUF	97	East	900.00	(A)	E		3800.00	3101.82	300.00	80.00
IR512	Driver	ir512	IBUF	98	East	900.00	(A)	E		3800.00	2403.64	300.00	80.00
IR513	Driver	ir513	IBUF	99	East	900.00	(A)	E		3800.00	1705.46	300.00	80.00
IR514	Driver	ir514	IBUF	100	East	900.00	(A)	E		3800.00	1007.28	300.00	80.00
IR515	Driver	ir515	IBUF	101	East	900.00	(A)	E		3800.00	309.10	300.00	80.00
IR516	Driver	ir516	IBUF	102	East	900.00	(A)	E		3800.00	-389.08	300.00	80.00
IR517	Driver	ir517	IBUF	103	East	900.00	(A)	E		3800.00	-1087.26	300.00	80.00
IR518	Driver	ir518	IBUF	104	East	900.00	(A)	E		3800.00	-1785.44	300.00	80.00
IR519	Driver	ir519	IBUF	105	East	900.00	(A)	E		3800.00	-2483.62	300.00	80.00
IR520	Driver	ir520	IBUF	106	East	900.00	(A)	E		3800.00	-3181.80	300.00	80.00
IR521	Driver	ir521	IBUF	108	South	900.00	(A)	S		3427.28	-3800.00	300.00	80.00
IR522	Driver	ir522	IBUF	109	South	900.00	(A)	S		2674.56	-3800.00	300.00	80.00
IR523	Driver	ir523	IBUF	110	South	900.00	(A)	S		1921.84	-3800.00	300.00	80.00
IR524	Driver	ir524	IBUF	111	South	900.00	(A)	S		1169.12	-3800.00	300.00	80.00
IR525	Driver	ir525	IBUF	112	South	900.00	(A)	S		416.40	-3800.00	300.00	80.00

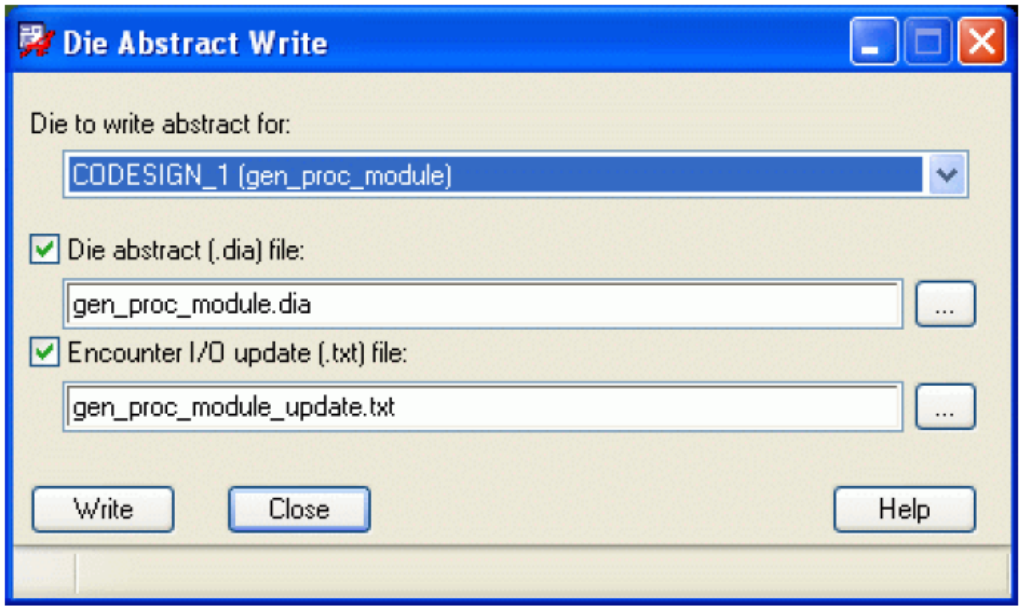
As shown in the previous image, the spreadsheet shows information for all drivers shown in the Design Window, as well as any unplaced drivers. You can edit any item that is not on a hashed (criss-crossed) background. For example, you can change the *Order, Edge, Nets, Offset,* and Location Characteristics fields, all shown in blue in the spreadsheet. When you make changes in the I/O Buffer Sequencing Spreadsheet, they are immediately reflected in the Design Window and also in the *Display Tree*, (located in the *Drivers* tab of the Die Editor: Component Editing dialog box) which lists all placed and unplaced driver instances.

Updating I/O Planner with Changes from APD

When you are running in a *Concurrent Co-Design* environment, as you make changes in APD using the Package-driven I/O Planning commands, the changes are automatically updated in I/O Planner.

When you are running these commands in *a Distributed Co-design* environment, APD updates the die abstract file for transfer of data from APD to I/O Planner. You use the *File – Export Die Abstract* (`die abstract export` command) to transfer this information (as follows).

Die Abstract Write Dialog Box



Updating APD from I/O Planner

When you are in a Concurrent Co-Design environment you might need to update APD, for example, if you perform RDL routing tasks in I/O Planner. You use the *Resync* button in the *Drivers* tab of the Die Editor to update APD. This action ensures that the I/O Planner and APD are synchronized.

Related Topics

- I/O Buffer Sequencing Spreadsheet Window
- die abstract export

Reviewing an Updated Die Abstract File

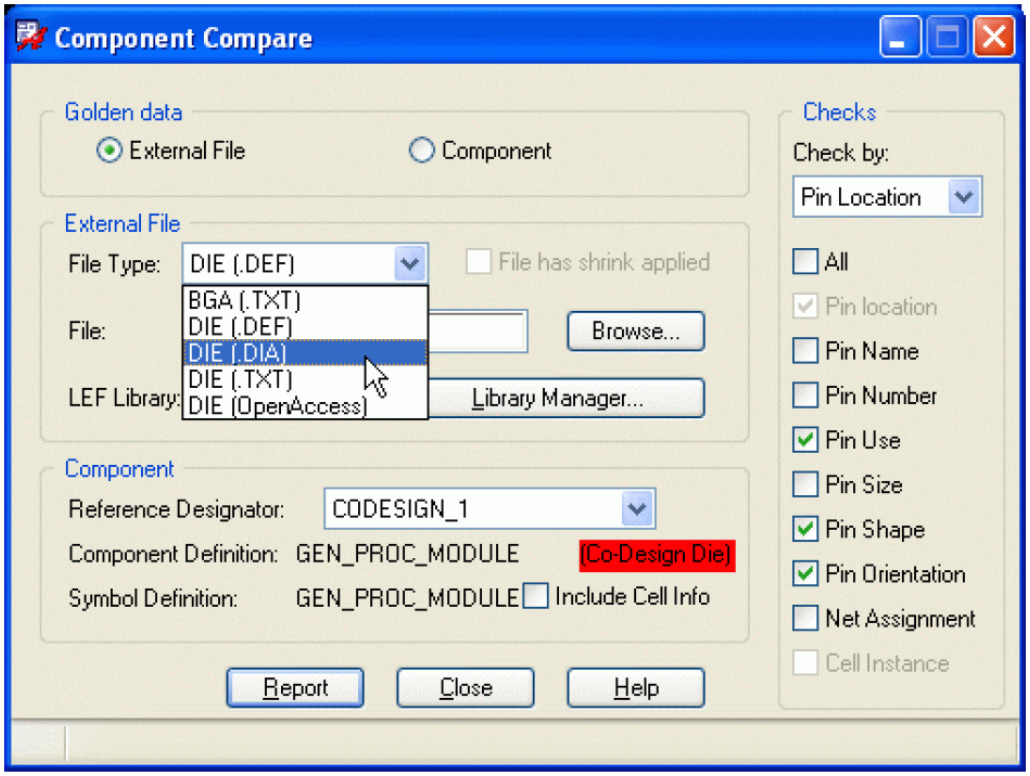
The updating of a co-design die with new information may take several iterations. If you are in a Distributed Co-Design environment, the IC designer and the package designer use the die abstract file to transfer changes during the modification process. When the IC designer modifies the co-design die, the package designer may want to review the differences before updating the package design.

You can use the Die Abstract Compare (`diaCompare`) utility or the Component Compare (`compare comp`) utility to compare dia files before updating a die with a new abstract file.

You can use the Die Abstract Compare utility (`diaCompare`) to compare two die abstract files. The `<Cadence_Installation>/tools/pcb/bin/diaCompare` utility is independent of the tools being used and can be run from the command prompt. You can run the `diaCompare` command without any parameters to get a list of options and their description. For additional information see the dia compare command.

The Component Compare feature (`compare comp` command) is available to compare an updated die abstract and the existing die information before importing the updated die abstract into the layout design.

Component Compare Dialog Box

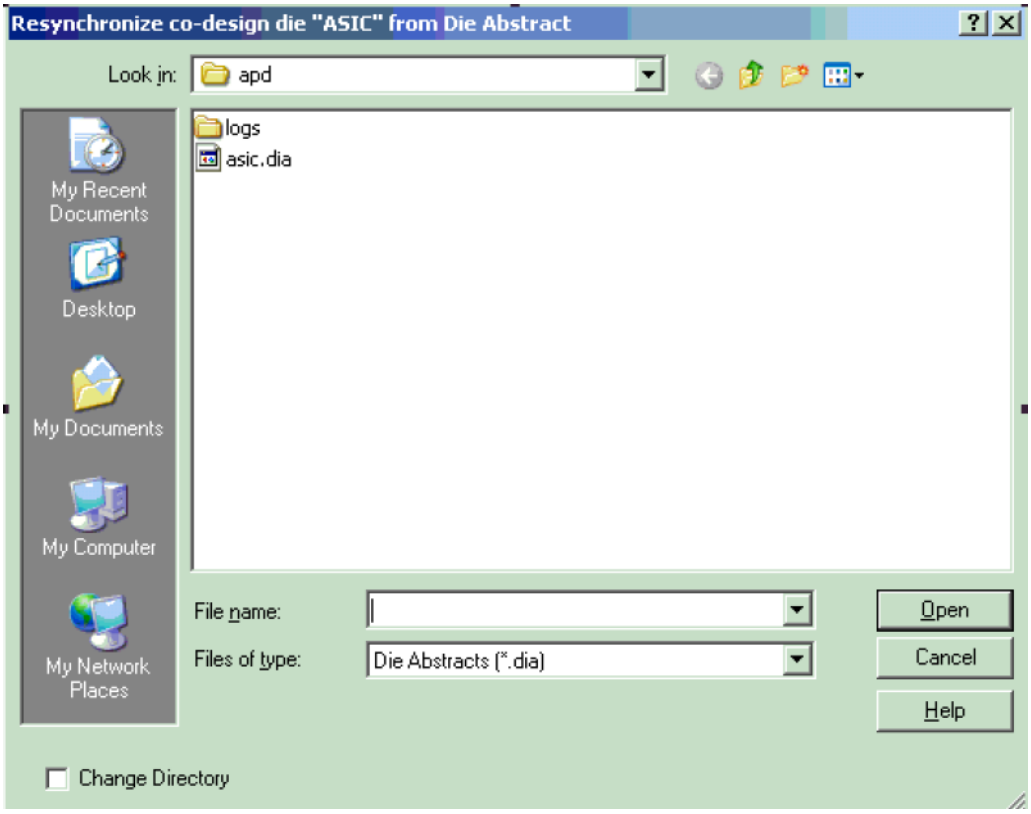


For additional information, see the compare comp command.

Updating a Co-Design Die in a Distributed Environment

To update a co-design die (created in a Distributed Co-Design environment) in APD with new information from the IC designer, you need to import the updated die abstract file.

Select a die abstract file (`dia`) from which to import an ECO to the die under edit. The design name referenced within the die abstract file must match the design name of the current die.



Saving an OA-Based Co-Design Die

ICP products that support Open Access (OA)-based, IO Planner (IOP) co-design dies as part of a package design flow now have automated *Save As* (save as command) operations on the OA databases as part of the overall APD *Save As* process. The tool makes a copy of the OA cell view for each OA-based co-design die in a package design in the same directory where the original OA library is located. To retain the original cell and view names, it copies the cell view to a new OA library and assigns it an auto-generated name (or when appropriate, reuses an existing OA library name). It ensures that the name of the copy is referenced in the parent `lib.defs` file. It also updates the internal OA library reference for each co-design die in the new package to point to the new OA library.

You can make backup copies of a package design and the OA libraries referenced from the dies contained within the package. You can also restore a design from a backup design by performing a package *Save As* operation from a previously backed-up design to the original design.

If you previously saved the backup design, you can restore the original design from that backup, as follows:

- Open the backup design, `myPackage_backup.mcm`.
- Choose *File – Save As*, browse to the original design in the `work` directory, and select it as the save-as target
- Confirm to overwrite the original design when prompted.
  When you confirm the overwrite, the tool saves the `myPackage_backup.mcm` as `myPackage.mcm` in the work directory. Then it saves `OALib_myPackage_backup` library as `myOALib`. It sets the co-design die in `myPackage.mcm` to reference `myOALib` library.

Scripting between Layout Tools and IOP

This section describes the scripting capability between APD and IOP. IOP is a Tcl-based product. It writes a log file (`.cmdlog`) of all commands processed during an IOP session. You can use this file as a source for creating a Tcl script file that you then use as input to IOP at launch.

With this capability, you can:

- Pass Tcl script commands and source files to IOP upon IOP launch.
- Force the layout tool into a wait state, while awaiting an update request from IOP.
- Continue script replay in the layout tool upon completion of the IOP die instance update.

A typical recorded session may include:

1. Opening a specific `.mcm` design.
2. Performing any pre-IOP operations, if necessary.
3. Launching an IOP edit session with the *Edit – Die* command.
4. Updating IOP edits to the existing OA-based die instance in the design.
5. Performing any post-IOP operations in the layout tool, if necessary.

After you finish scripting, unset the `IOP_INIT_CMD` environment variable so that you can use IOP again. Otherwise, IOP fails to load a die in that console window.

```
% unsetenv IOP_INIT_CMD
```

3

APD: Working with RF PCells

As a designer, you would have specified the dimensions of the RF components in the schematic. However, when designing the layout, you may need to make some changes. You can perform these changes using the RF Module option in using APD. You can also make the changes directly in the layout by changing the shapes by hand. However, this method is not recommended.

⚠ The footprints of the RF components are created at runtime by invoking several SKILL functions. You can modify the SKILL functions as required to customize the footprint generation.

Related Topics

- SKILL Functions for PCELLS.

Placing RF Pcells

RF Pcells are placed like other components through the *Place – Manually* menu option in APD. Components are brought in from Virtuoso Schematic or can be added directly through *Logic – Edit Part List* menu option.

⚠ You must properly define the Allegro environment variable called `pcell_lib_path`. This environment variable is used for searching the pcells. This must be set in the Allegro environment file shared across front end and back end.

Editing PCells

You can edit the PCells either by using the RF Module option, graphical editing or manually modifying the shapes.

Irregular edit occurs when you modify the shape of the footprint in the APD editor. The Edit RF Properties dialog box displays this information in the title bar and the changes made to the properties will not be annotated to the design.

You need to use the update pcell symbols command to update the symbols of the shapes that are irregularly edited.

The command syntax is as follows:

```
update_pcell_symbols(ALL|SELECTED)(PROP_MODIFIED|IRREGULAR_EDITED|ALL)
```

where

Parameter	Description
ALL|SELECTED	Depending on your choice, updates all the symbols or the selected ones.
PROP_MODIFIED|IRREGULAR_EDITED|ALL	The PROP_MOIDIFIED option updates the footprint with the modified property value. The IRREGULAR_EDITED option updates the footprint with the property values that have not been changed, effectively undoing the irregular edit. The ALL option regenerates the footprint as per the current property values.

Example

```
update_pcell_footprint ALL PROP_MODIFIED
```

1. Choose RF Module – Graphic Edit.
2. Select the RF component whose shape you want to modify.
3. Drag one of the handles and resize as required.
4. Double-click to complete resizing.
5. Right-click and select *Done*.

SKILL Functions for PCELLS

A PCELL component should have property PCELL_TYPE=TRUE in the component definition. It comes from the `chips.prt` file in front end. The value of the JEDEC_TYPE property in `chips.prt` file for a PCELL component is a SKILL file name (extension.il or .ile is assumed with the value), instead of the usual `.psm/.dra` file. The SKILL file contains various functions which are used to create the footprint on the fly.

⚠ In the names of the functions mentioned below, *componentName* must be replaced by the value of the JEDEC_TYPE property.

Functions for Creating Footprint

l_List _sip*componentName*GetPcellParameters()

`This function returns a list containing information about the parameters.`

Example

```
procedure(_sipROUNDSPIRALGetPcellParameters()

(prog (l_lPropList)

    l_lPropList = _sipCreatePCellParamList("ROUNDSPIRAL")

    return(l_lPropList)

))
```

_sip*componentName*CreateFootprint(t_packageName ...)

This function creates the footprint in real time based upon the arguments. This function is called from APD when the component is being placed. This function will be called with name of the component symbol to be created, and the remaining arguments are the values of those parameters which have type as "physical" or "physical_layer".

Example

```
procedure( _sipROUNDSPIRALCreatefootprint(packageName layer width  space innerdia
numturns  )

  prog(( HSTC )

    HSTC = 0

    width = _sip_sklAPDStripUnit(width)

    space = _sip_sklAPDStripUnit(space)

    innerdia = _sip_sklAPDStripUnit(innerdia)

    if( HSTC == 1 then

      hstc_init()

      ROUNDSPIRAL_hstcfootprint(?layer layer ?package packageName ?width width ?spacing
space ?innerRadius innerdia ?numturns float(numturns) )

    else

    ROUNDSPIRAL_rfcreatefootprint(?packageName packageName ?layer layer ?width width ?
space space ?innerdia innerdia  ?numturns numturns )

    )

  )

)
```

Functions for Regular Edit and Analysis

l_List _sip*componentName*GetProperty()

This function returns a list containing information about the parameters. The format of the list is shown in the following example:

```
  l_lPropList = list(

list ("W" "20 microns" "Physical" "Yes" "Width" "string")

    list("Temp" "40" "Simulation" "No" "Temperature" "")

  )
```

Each element in the top level list is a list containing information about a parameter. The first element of this list is the parameter name followed by the default value, type (can be physical, simulation, layer, physical_layer and so on), whether to annotate the value on the schematic instance, parameter description, and type of the parameter value (optional).

If the type of a parameter is "physical" or its type is "physical_layer", then that parameter is required for creating the footprint. For all such parameters, the last parameter specifying the type of the value is also required. All the parameters of these types are passed to _sip*componentName*createFootPrintFunction().

Example

```
procedure(_sipROUNDSPIRALGetProperty()
( prog ( l_lPropList )

 l_lPropList = list(

     list("W"   "20 UM"  "Physical" "Yes" "Width" "string")
     list("S" "10 UM"  "Physical" "Yes" "space" "string")
     list("Ri"  "50 UM"  "Physical" "Yes" "Inner Radius" "string")
     list("N"   "4"   "Physical" "Yes" "Number of Turns" "int")
     list("F" "1Ghz"  "Electrical" "Yes" "Frequency")
           list("Temp" "40" "Simulation" "No" "Temperature")
     list("APDLAYER_A" "TOP_COND" "layer" "Yes" "Metallic Layer" "string")
           list("APDLAYER_B" "DIELEC_LAY" "layer" "Yes" "Dielectric Layer" "string")
                                list("PCELL_VERSION" "v1.0" "Simulation" "Yes" "Version")
              )

      return(l_lPropList)
 )
)
```

l_list _sip*componentName*Analyze(l_list)

This function is called from both Virtuoso Schematic Editor and cdnsip for analysis. It gets the list of parameters information list with type as "physical" "layer" "physical_layer". It returns the list of electrical parameter after analysis.

Example

```
        -


procedure(_sipROUNDSPIRALAnalyze(l_lPhyPropLst)
( prog (returnList l_lBasePhyPropLst l_lLayerList  l_lLayerData  l_lLayerName
 t_lLibPath t_lTechDgnUnit t_lErr)

         l_lBasePhyPropLst = l_lPhyPropLst

        l_lPhyPropLst = _sipPCellSimulationList(l_lBasePhyPropLst)
 l_lLayerList = _sipFilterPropertyList(l_lBasePhyPropLst "Layer")


        l_lLayerData = _sipGetTechLayerDataEx(l_lLayerList "APDLAYER_B")

        if((l_lLayerData == nil) then
                    return(nil)
        )

        l_lLayer = _sipGetLayerPropData(l_lLayerData "layerDielectricConstant" "ER")
        if((l_lLayer != nil) then
             l_lPhyPropLst = append( l_lPhyPropLst list(l_lLayer))
        else
             sprintf(t_lErr "\n Dielectric Constant missing for layer %s  \n" l_lLayerName)
             println(t_lErr)
             return(nil)
        )

 l_lLayer = _sipGetLayerPropData(l_lLayerData "layerThickness" "t")
        if((l_lLayer != nil) then
             l_lPhyPropLst = append( l_lPhyPropLst list(l_lLayer))
        else
             sprintf(t_lErr "\n Layer Thickness missing for layer %s  \n" l_lLayerName)
             println(t_lErr)
             return(nil)
        )


 t_lLibPath = _sipGetLibPath("roundspiral")
 if((t_lLibPath != nil) then
  t_lTechDgnUnit = _sipGetTechDesignUnit()
  returnList = _sipRF_DllFnExec( t_lLibPath "Analyse" t_lTechDgnUnit l_lPhyPropLst)
 )
 return(returnList)
```

```
))
```

t_name _sip*componentName*BitmapName()

This is an optional function which is needed only if the graphic view of the component is to be shown in the Edit RF Properties panel in cdnsip. It returns the name of the bmp which should be present in the BMPPATH defined using TELENV file.

```
procedure(_sipROUNDSPIRALBitmapName()
prog(()
  return("rfsip_roundspiral.bmp")
))
```

Complete Example: Creating a Footprint for a Component with jedec Type Set to roundspiral

Following is a complete example of how to create a footprint for a component with jedec type set to roundspiral.

```
; Author --
 ;    Cadence Design Systems
 ;
 ; Function --
 ;            1) _sipROUNDSPIRALGetProperty
 ;            2) _sipROUNDSPIRALAnalyze
 ;          3) _sipROUNDSPIRALGetIconList
 ;          4) _sipROUNDSPIRALBitmapName
 ;          5) _sipROUNDSPIRALGetPcellParameters
 ;          6) _sipROUNDSPIRALCreatefootprint
 ;
 ; Copyright, 1993, Cadence Design Systems, Inc.
 ; No part of this file may be reproduced, stored in a retrieval system,  ; or transmitted in any
form or by any means --- electronic, mechanical,  ; photocopying, recording, or otherwise ---
without prior written permission  ; of Cadence Design Systems, Inc.
 ;
 ; WARRANTY:
 ; NONE. NONE. NONE.
 ; Use all material in this code at your own risk.  Cadence Design Systems  ; makes no claims about
any material in this archive.  These examples may  ; not function or may only function in specific
instances. We'd like to hear  ; what you think of our approach to this, and how we can improve it.
 ;
 ; RESTRICTIONS:
 ; All software contained within this archive is in the public domain or  ; the author has given us
permission for redistribution.  Some packages  ; have explicit copyrights and notices concerning
their redistribution.
 ; Please carefully read all documentation with any package on this tape.
 ;
 ; CHANGE LOG:
 ; Initial Version 1.0 date 04/05/2005
 ;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
procedure(_sipROUNDSPIRALGetProperty()
( prog ( l_lPropList )

 l_lPropList = list(

     list("W"   "20 UM"  "Physical" "Yes" "Width" "string")
     list("S" "10 UM"  "Physical" "Yes" "space" "string")
     list("Ri"  "50 UM"  "Physical" "Yes" "Inner Radius" "string")
     list("N"   "4"   "Physical" "Yes" "Number of Turns" "int")
     list("F" "1Ghz"  "Electrical" "Yes" "Frequency")
           list("Temp" "40" "Simulation" "No" "Temperature")
     list("APDLAYER_A" "TOP_COND" "layer" "Yes" "Metallic Layer" "string")
           list("APDLAYER_B" "DIELEC_LAY" "layer" "Yes" "Dielectric Layer" "string")
                              list("PCELL_VERSION" "v1.0" "Simulation" "Yes" "Version")
              )

      return(l_lPropList)
 )
)

;*****************************************************************************************************
*

procedure(_sipROUNDSPIRALAnalyze(l_lPhyPropLst)
( prog (returnList l_lBasePhyPropLst l_lLayerList  l_lLayerData  l_lLayerName
 t_lLibPath t_lTechDgnUnit t_lErr)

          l_lBasePhyPropLst = l_lPhyPropLst

        l_lPhyPropLst = _sipPCellSimulationList(l_lBasePhyPropLst)
 l_lLayerList = _sipFilterPropertyList(l_lBasePhyPropLst "Layer")



        l_lLayerData = _sipGetTechLayerDataEx(l_lLayerList "APDLAYER_B")

        if((l_lLayerData == nil) then
                    return(nil)
        )

        l_lLayer = _sipGetLayerPropData(l_lLayerData "layerDielectricConstant" "ER")
        if((l_lLayer != nil) then
             l_lPhyPropLst = append( l_lPhyPropLst list(l_lLayer))
        else
```

```
            sprintf(t_lErr "\n Dielectric Constant missing for layer %s  \n" l_lLayerName)
            println(t_lErr)
            return(nil)
       )

 l_lLayer = _sipGetLayerPropData(l_lLayerData "layerThickness" "t")
       if((l_lLayer != nil) then
            l_lPhyPropLst = append( l_lPhyPropLst list(l_lLayer))
       else
            sprintf(t_lErr "\n Layer Thickness missing for layer %s  \n" l_lLayerName)
            println(t_lErr)
            return(nil)
       )


 t_lLibPath = _sipGetLibPath("roundspiral")
 if((t_lLibPath != nil) then
  t_lTechDgnUnit = _sipGetTechDesignUnit()
  returnList = _sipRF_DllFnExec( t_lLibPath "Analyse" t_lTechDgnUnit l_lPhyPropLst)
 )
 return(returnList)
))

;*****************************************************************************************
*

procedure( _sipROUNDSPIRALGetIconList(@key (packageName "sinductor") (layer "ETCH/TOP") (width
500.0)
           (space 2000.0) (innerdia 0.0) (numturns 5.0)
         )
      prog( (  xc yc llx lly urx ury n outer dl iconList lPointList lPointListTmp )
               xc = 0.0
               yc = 0.0
               n = numturns
               outer = 2.0 * n * ( width + space ) + innerdia
               llx = xc - outer / 2.0
               urx = xc + outer / 2.0
               lly = yc - outer / 2.0
               ury = yc + outer / 2.0
               lPointListTmp = inductor(?packageName packageName ?layer layer ?urx urx ?ury ury ?
llx llx ?lly lly ?width width
               ?space space ?pointlistgenerator 1)
```

```
                for(i 0 (length(lPointListTmp)-1)

                   if((mod(i 3) == 0) then
                          lPointList  =append(lPointList list(list(round(nth(0 nth(i
lPointListTmp)))
                                                                  round(nth(1 nth(i lPointListTmp)))
                                      )))
                    )
                )

                dl = dlMakeDisplayList()
                dl.penTable = dlMakePenTable(5)
                dlSetPenTable( dl   dl.penTable)
                dlSetPenColor(1 hiMatchColor(nameToColor("white")) dl.penTable)
                dlSetPenColor(2 hiMatchColor(nameToColor("red")) dl.penTable)
                dlSetPenColor(3 hiMatchColor(nameToColor("grey")) dl.penTable)
                dlSetPenColor(4 hiMatchColor(nameToColor("blue")) dl.penTable)
                dlSetPenFillStyle(1 "SolidFill" dl.penTable)
                dlSetPenFillStyle(4 "SolidFill" dl.penTable)
                dlClearDisplayList(dl)

                dlAddPolygon(dl 4 lPointList)

                iconList = dlDlistToIcon(dl 200 200 3)

                return(iconList)
                return(lPointList)
   )
 )

procedure(ROUNDSPIRAL_icongenerator()
 _sip_rfIconGenratorFn("sipROUNDSPIRALGetIconList")
 )

;*****************************************************************************************
*

procedure(_sipROUNDSPIRALBitmapName()
prog(()
  return("rfsip_roundspiral.bmp")
))

;*****************************************************************************************
```

```
*

procedure(_sipROUNDSPIRALGetPcellParameters()
(prog (l_lPropList)
 l_lPropList = _sipCreatePCellParamList("ROUNDSPIRAL")
 return(l_lPropList)
))

;*****************************************************************************************************
*

procedure( _sipROUNDSPIRALCreatefootprint(packageName layer width  space innerdia numturns  )
  prog(( HSTC )
    HSTC = 0

    width = _sip_sklAPDStripUnit(width)
    space = _sip_sklAPDStripUnit(space)
    innerdia = _sip_sklAPDStripUnit(innerdia)

    if( HSTC == 1 then
       hstc_init()
       ROUNDSPIRAL_hstcfootprint(?layer layer ?package packageName ?width width ?spacing space ?
innerRadius innerdia ?numturns  float(numturns) )
    else
        ROUNDSPIRAL_rfcreatefootprint(?packageName packageName ?layer layer ?width width ?space
space ?innerdia innerdia  ?numturns numturns )
    )
 )
)

;*****************************************************************************************************
*

procedure( ROUNDSPIRAL_rfcreatefootprint(@key (packageName "sinductor") (layer "ETCH/TOP") (width
50.0) (space 2000.0) (innerdia 0.0) (numturns 5.0) )

      let( (  xc yc llx lly urx ury n outer  )
                xc = 0.0
                yc = 0.0
                n = numturns

                outer = n * (width + space)  + innerdia
```

```
                llx = xc - outer
                urx = xc + outer
                lly = yc - outer
                ury = yc + outer

                inductor(?packageName packageName ?layer layer ?urx urx ?ury ury ?llx llx ?lly lly ?
width width ?space space ?inner innerdia ?n n)
 )

)




;*********************************************************************************************
*

procedure( ROUNDSPIRAL_hstcfootprint(@key (deviceType 33) (layer "ETCH/TOP") (package "ind5_15")
(netname "") (padName1 nil)
     (padName2 nil) (numturns 5.0) (innerRadius 0.0) (spacing 10) (width 15) (flipMode 0))
   _rfCreateMSINDSymbolDef(deviceType layer package netname padName1 padName2
       numturns innerRadius spacing width flipMode)
)


;*********************************************************************************************


procedure( inductor(@key (packageName "sinductor") (llx -15000.0) (lly -15000.0) (urx 15000.0) (ury
15000.0) (width 50.0)
    (space 2000.0) (layer "ETCH/TOP")  (pointlistgenerator 0) (inner 500.0)(n 5)
     )
      prog( (l_extents pin1 pin2 l_pinData symdef textOrient P1 P2 path1  x y nq ns nx xo chCos
               chSin c r twopi d points outer points2 x1 y1 xc yc x2 y2 x3 y3 nc x2p y2p x3p y3p
             )
               xc = (llx + urx)/2
               yc = (lly + ury)/2


               ;inner  =0.0 ; inner diameter
               ; build the spiral shape:
               ; compute the number of turns:
               ;n = (outer - inner) / 2. / (width + space)
```

```
            ; compute number of quarter turns and sides per quarter turn:

            outer  = urx - llx ; outer diameter
            nc = 72 ; number of chords per 360 degrees
            nq = fix( n * 4)
            ns = fix( nc / 4)
            ; calculate some intermediate parameters:
            twopi = 2 * 3.1415926
            d = twopi / nc ;nc is number of chords per 360 degrees

            chCos = cos( d)
            chSin = sin( d)
            c = (width + space) / nc
            r = outer * .5
            ; build the point lists:
            x = r
            y = 0.
            points = nil
            x2 = nil
            y2 = nil
            x3 = nil
            y3 = nil
            for( i 1 nq
                nx = ns
                for( j 1 nx
                    xo = x
                    x = (chCos * xo - chSin * y) * (1. - c / r)
                    y = (chCos * y + chSin * xo) * (1. - c / r)
                    x1 = x + xc
                    y1 = y + yc
                    if(x2 == nil && y2 == nil then
                        x2 = x1
                        y2 = y1
                    )
                    points = cons( x1:y1 points)
                    r = r - c
                    if((i == (nq-1)) && (j == (nx-1) && ((space*width*inner) > 0)) then
xo = x
x = (chCos * xo - chSin * y) * (1. - c / r)
y = (chCos * y + chSin * xo) * (1. - c / r)
x1 = x + xc
y1 = y + yc
if(x2 == nil && y2 == nil then
```

```
      x2 = x1
      y2 = y1
  )
  points = cons( x1:y1 points)
  r = r - c
                      )
                   )
              )



              x3 = x1
              y3 = y1
              c = (width + space) / nc
              r = (outer * .5) + width
              x = r
              y = 0.
x2p = 0
y2p = 0
x3p = 0
y3p = 0
              points2 = nil
              for( i 1 nq
                 nx = ns
                 for( j 1 nx
                     xo = x
                      x = (chCos * xo - chSin * y) * (1. - c / r)
                      y = (chCos * y + chSin * xo) * (1. - c / r)
                      x1 = x + xc
                      y1 = y + yc
                      if(x2p == 0 && y2p == 0 then
                          x2p = x1
                          y2p = y1
                      )
                      points2 = cons( x1:y1 points2)
                      r = r - c
 if((i == (nq-1)) && (j == (nx-1) && ((space*width*inner) > 0)) then
        xo = x
  x = (chCos * xo - chSin * y) * (1. - c / r)
  y = (chCos * y + chSin * xo) * (1. - c / r)
  x1 = x + xc
  y1 = y + yc
  if(x2p == 0 && y2p == 0 then
```

```
        x2p = x1
        y2p = y1
    )
    points2 = cons( x1:y1 points2)
    r = r - c
  )

                    )
                )
  x3p = x1
  y3p = y1
            points = append(reverse(points2) points)
            points = cons(x2:y2 points)

            if( (pointlistgenerator == 1) then
                  return(points)
            else
                ; create the symbol definition

  if( (rexMatchp("/" layer) == 't)
  then
   layer_break=parseString(layer "/")
   conducting_layer=cadr(layer_break)
  else
   conducting_layer=layer

   )

  sprintf(psname "%s" conducting_layer)

  if( (axlLoadPadstack(psname) == 'nil) then
   pad_list = cons(make_axlPadStackPad(
   ?layer conducting_layer, ?type 'REGULAR, ?figure 'CIRCLE, ?figureSize width/2:width/2) nil)
   axlDBCreatePadStack(psname nil pad_list t)
  )
                  P1=list(x2+ (x2p-x2)/2 y2 + (y2p-y2)/2)
                  P2=list(x3 +  (x3p-x3)/2 y3 + (y3p-y3)/2)
                pin1=make_axlPinData(?rotation 0.0 ?origin P1 ?padstack psname ?number "1")
                  pin2=make_axlPinData(?rotation 0.0 ?origin P2 ?padstack psname ?number "2")
                  l_pinData=list(pin1 pin2)
                  l_extents=list((llx-width-space-1000.0):(lly-width-space-1000.0)
(urx+width+space+1000.0):(ury+width+space+1000.0))
                  ; symbol definition skeleton that contains only the pin information
```

```
                symdef=_axlDBCreateSymDefSkeleton(list(packageName "PACKAGE") l_extents l_pinData)
                if((symdef == nil) then
                       printf("symbol definition could not be created \n")
              else
                  printf("symbol definition created \n")

            )
           textOrient=make_axlTextOrientation(?textBlock "1" ?rotation 0.0 ?mirrored nil ?justify
"center")
  /* Refdes in REF DES/ASSEMBLY_TOP */

         axlDBCreateText("RFU*" P1 textOrient "REF DES/ASSEMBLY_TOP" symdef)
         path1=axlPathStart(points)
         axlDBCreateShape(path1 t layer nil symdef)
                ;axlWritePackageFile(symdef)

          )
 )
)

;*************************************************************************************************
*

procedure( inductor_caller(@key (packageName "sinductor") (layer "ETCH/TOP") (width 50.0) (space
2000.0) (innerdia 0.0)
    (numturns 5.0)
     )
      let( (  xc yc llx lly urx ury n outer  )
                xc = 0.0
                yc = 0.0
                n = numturns
                outer = 2.0 * n * ( width + space ) + innerdia
                llx = xc - outer / 2.0
                urx = xc + outer / 2.0
                lly = yc - outer / 2.0
                ury = yc + outer / 2.0
                inductor(?packageName packageName ?layer layer ?urx urx ?ury ury ?llx llx ?lly lly ?
width width ?space space)
 )

)

;*************************************************************************************************
```

```
*

procedure( hstc_init()
if( isCallable('axlDBCreateText) then
 printf("Loading HSTC code\n")
 if( getShellEnvVar("HSTC_SKILL_DIR") then
  t_ghstcSkillPath = getShellEnvVar("HSTC_SKILL_DIR")
 else
  t_ghstcSkillPath = getShellEnvVar("pcell_lib_path")
 )
 if( t_ghstcSkillPath == nil then
  t_ghstcSkillPath="."
 )
 loadi(strcat(t_ghstcSkillPath "/msindcomp.ile"))
))

;*****************************************************************************************************
*

procedure( inductor_cline(@key (packageName "sinductor") (llx -15000.0) (lly -15000.0) (urx 15000.0)
(ury 15000.0) (width 50.0)
    (space 2000.0) (layer "ETCH/TOP")  (pointlistgenerator 0)
     )
      prog( (l_extents pin1 pin2 l_pinData symdef textOrient P1 P2 path1  x y n nq ns nx xo chCos
               chSin c r twopi d points inner outer points2 x1 y1 xc yc x2 y2 x3 y3 nc x2p y2p y3p
              )
                xc = (llx + urx)/2
                yc = (lly + ury)/2
                inner  = 0.0 ; inner diameter
                outer  = urx - llx ; outer diameter
                nc = 72 ; number of chords per 360 degrees
                ; build the spiral shape:
                ; compute the number of turns:
                n = (outer - inner) / 2. / (width + space)
                ; compute number of quarter turns and sides per quarter turn:
                nq = fix( n * 4)
                ns = fix( nc / 4)
                ; calculate some intermediate parameters:
                twopi = 2 * 3.1415926
                d = twopi / nc ;nc is number of chords per 360 degrees
                chCos = cos( d)
                chSin = sin( d)
                c = (width + space) / nc
```

```
                r = outer * .5
                ; build the point lists:
                x = r
                y = 0.
                points = nil
pointLists = nil
                x2 = nil
                y2 = nil
                x3 = nil
                y3 = nil
                for( i 1 nq
                    nx = ns
                    for( j 1 nx
                        xo = x
                        x = (chCos * xo - chSin * y) * (1. - c / r)
                        y = (chCos * y + chSin * xo) * (1. - c / r)
                        x1 = x + xc
                        y1 = y + yc
                        if(x2 == nil && y2 == nil then
                            x2 = x1
                            y2 = y1
                        )
                        points = cons( x1:y1 points)
                        r = r - c
                    )
                    pointLists = cons( points pointLists)
                )
                        points = cons( x1:y1 points)
                x3 = x1
                y3 = y1
                c = (width + space) / nc
                r = (outer * .5) + width
                x = r
                y = 0.
x2p = 0
y2p = 0
y3p = 0
                points2 = nil
                for( i 1 nq
                   nx = ns
                   for( j 1 nx
                       xo = x
                        x = (chCos * xo - chSin * y) * (1. - c / r)
```

```
                    y = (chCos * y + chSin * xo) * (1. - c / r)
                    x1 = x + xc
                    y1 = y + yc
                    if(x2p == 0 && y2p == 0 then
                        x2p = x1
                        y2p = y1
                    )
                    points2 = cons( x1:y1 points2)
                    r = r - c
                )
            )
y3p = y1
          ;points = append(reverse(points2) points)
          ;points = cons(x2:y2 points)

          if( (pointlistgenerator == 1) then
                return(points)
          else
              ; create the symbol definition

if( (rexMatchp("/" layer) == 't)
then
 layer_break=parseString(layer "/")
 conducting_layer=cadr(layer_break)
else
 conducting_layer=layer

 )

sprintf(psname "%s" conducting_layer)

if( (axlLoadPadstack(psname) == 'nil) then
 pad_list = cons(make_axlPadStackPad(
 ?layer conducting_layer, ?type 'REGULAR, ?figure 'CIRCLE, ?figureSize width/2:width/2) nil)
 axlDBCreatePadStack(psname nil pad_list t)
)
                P1=list(x2  y2 + (y2p-y2)/2)
                P2=list(x3  y3 + (y3p-y3)/2)
              pin1=make_axlPinData(?rotation 0.0 ?origin P1 ?padstack psname ?number "1")
                pin2=make_axlPinData(?rotation 0.0 ?origin P2 ?padstack psname ?number "2")
                l_pinData=list(pin1 pin2)
                l_extents=list((llx-1000.0):(lly-1000.0) (urx+1000.0):(ury+1000.0))
                ; symbol definition skeleton that contains only the pin information
```

```
                symdef=_axlDBCreateSymDefSkeleton(list(packageName "PACKAGE") l_extents l_pinData)
                if((symdef == nil) then
                       printf("symbol definition could not be created \n")
              else
                  printf("symbol definition created \n")

            )
           textOrient=make_axlTextOrientation(?textBlock "1" ?rotation 0.0 ?mirrored nil ?justify
"center")
  /* Refdes in REF DES/ASSEMBLY_TOP */

        axlDBCreateText("RFU*" P1 textOrient "REF DES/ASSEMBLY_TOP" symdef)
        path1=axlPathStart(points, width)
               ;foreach( path pointLists
                   axlDBCreatePath( path1 layer nil symdef)
               ;)
               ;axlWritePackageFile(symdef)

          )
 )
)

;*****************************************************************************************************
*
```

4

Overview of Placing Elements

Allegro layout editors provide manual and automatic tools for placing elements and swapping pins, functions (gates, inverters, or logical elements within a packaged component), and components. You can also place associations of elements.

⚠ Placement is not available in all Allegro X PCB Editor products.

Manual and Automatic Placement

With manual placement, you can place elements individually or place elements of the same type during one pass. You can use alternate symbols for components as you place them on a design. Alternate symbols are different representations of a particular package. You define the alternate symbols as property values in device files for package or, if using Concept, in a Concept schematic.

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 floorplan, 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

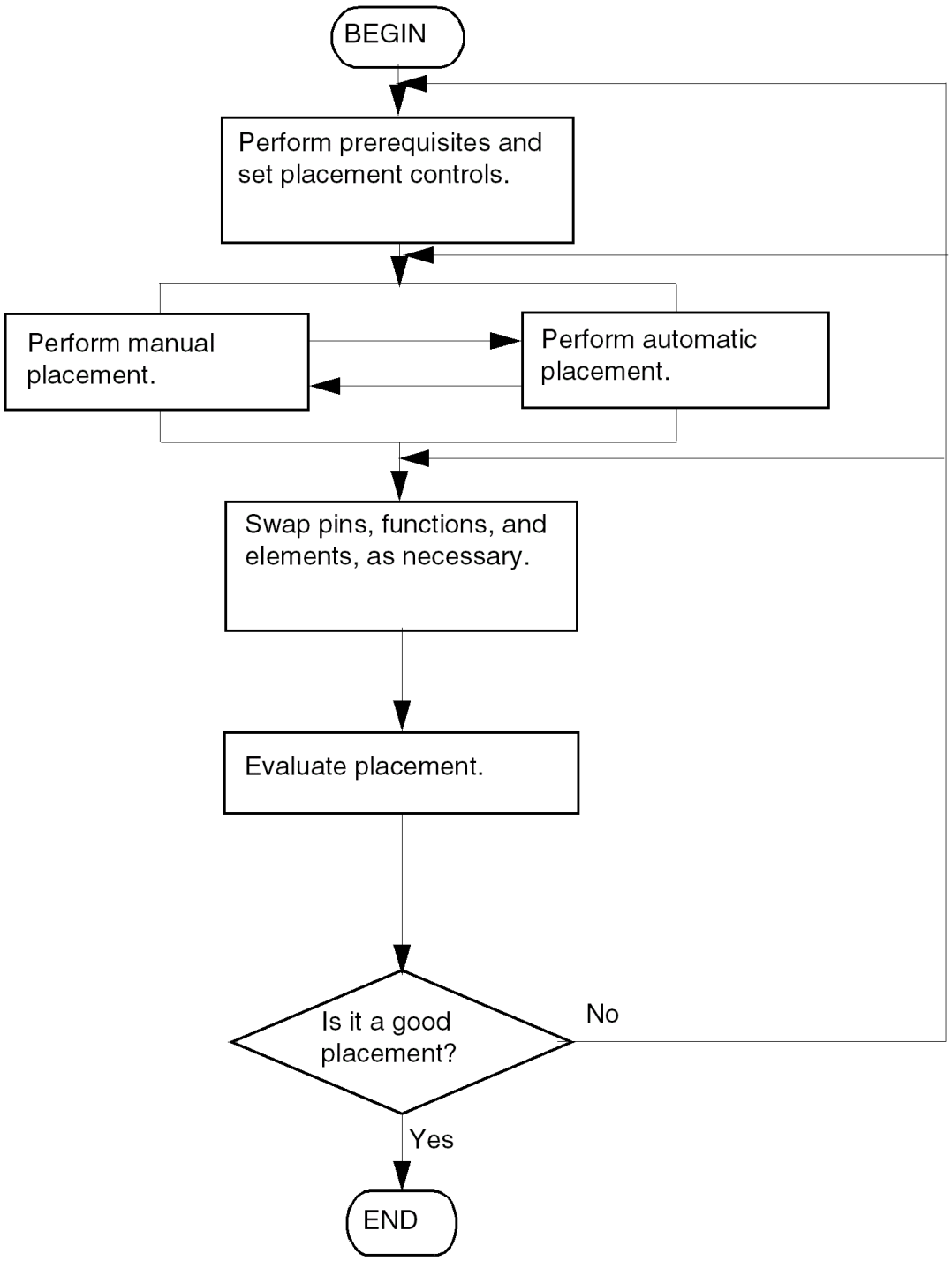
You can use any combination of placement and swap tools. For example, component placement typically involves the following activities:

- Determining design requirements
- Creating the items required for placement processing (such as grids or package keepin and keepout areas)
- 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
- Swapping pins, gates, and parts as necessary

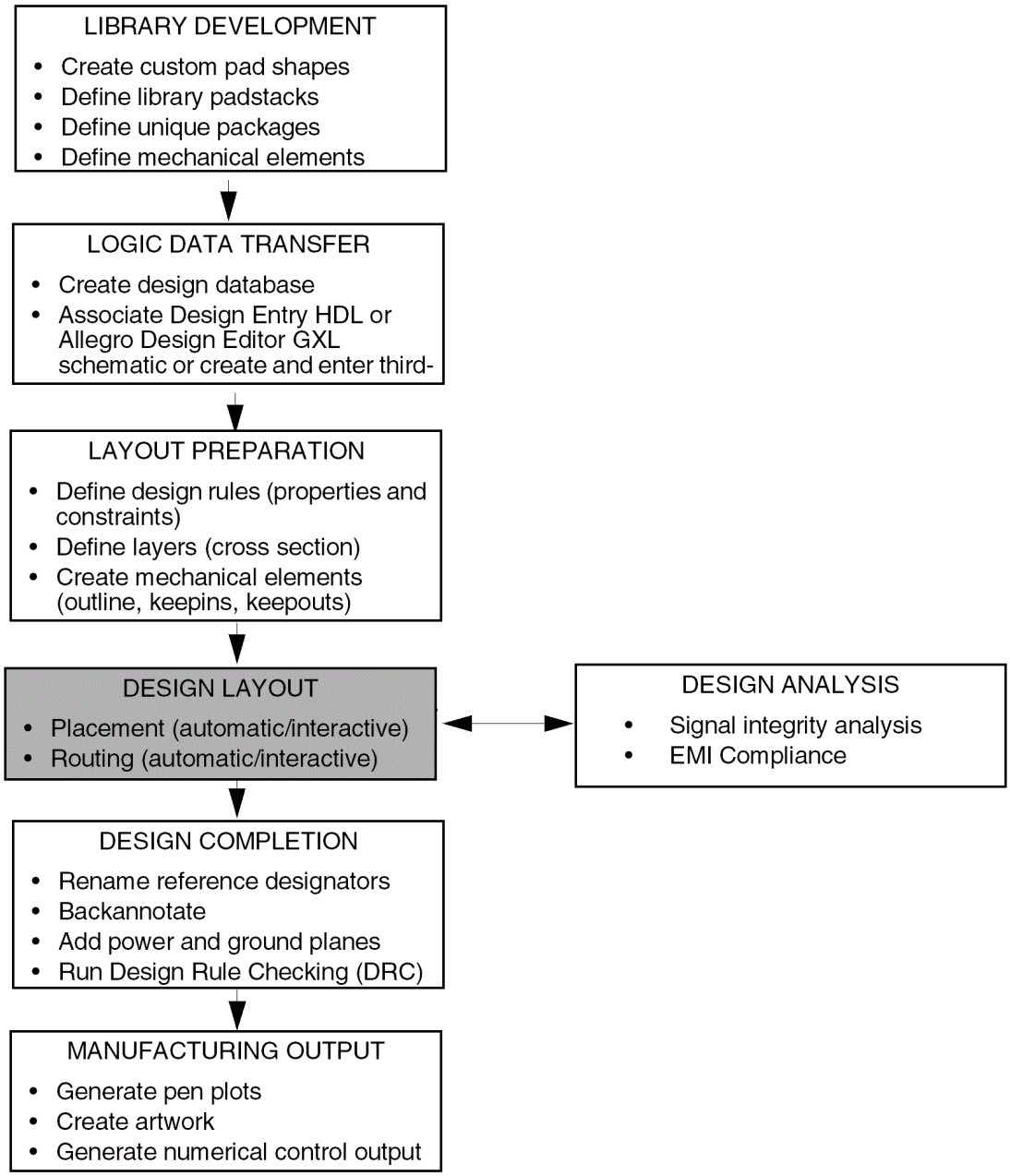
You can use the layout editor swapping capabilities to decrease the average wire length or uncross ratsnest lines. Automatic swapping exchanges pins or functions (gates) according to the controls and within the area that you define. Interactive swapping lets you swap selected elements as well as pins or functions.

The following figures show the placement process in the layout editor, and placement in a design flow.

Placement Process



Placement in a Design Flow



5

Preparation for Placing Elements

Before you can proceed with either manual or automatic placement, you must complete certain tasks, as well as determine design requirements. This section covers those tasks.

- Determining Design Requirements
- Automatic Placement Prerequisites
- Setting Placement Grids
- Assigning Placement Properties
- Creating a Floorplan Using Rooms
- Specifying Timing Data

Determining Design Requirements

Design considerations can influence how you prepare for placement. The following table lists questions to determine the design requirements for placing a design.

Determining Requirements for a Design

Checklist	Yes	No	Do this...
Does the design require that you restrict component placement to certain rooms?	X		Create a room into which to place certain components. Attach the ROOM property to each component to be placed in that room and attach the ROOM_TYPE property to the room boundary to control DRC error reporting for rooms. Run automatic placement for each room in the design.

Does the design contain components that must be placed in certain areas?	X		Place these components manually before performing automatic placement.
Do all components in each group have to be in the same room or placement area?		X	Add rooms and use the ROOM and ROOM_TYPE properties. See *Setup – Outlines – Room Outline* (`room outline` command).
Do you want to influence component positioning during automatic placement (for proximity to related components, rotation, mirroring, and so on) by applying specific weighting?	X		Set the appropriate automatic placement weight properties and parameters.
Does the design use alternate symbols?	X		Define alternate symbols with property values.
Do you want to swap components?		X	You can assign the FIXED property to each component that is not to be swapped.
Do you want to swap functions or pins?	X		Set the required controls for automatic swapping of functions and pins. You can also swap functions and pins interactively.
Do you want to use real-time Design for Assembly DRC for package-to-package clearance checks during interactive placement?	X		Use the DFA Constraints Dialog spreadsheet to define spacing rules between symbol definition pairs, available by choosing *Setup – DFA Spreadsheet* (`dfa_spreadsheet` command). You can also assign the DFA_DEV_CLASS property to symbols and create a class, to which the spacing values defined for the class in the DFA Constraints Dialog spreadsheet default.

Related Topics

- room outline
- Assigning Placement Properties
- Setting Automatic Placement Parameters Interactively
- Placing Alternate Symbols

Automatic Placement Prerequisites

The following table summarizes the items that must exist before running automatic placement in its two modes: interactive and automatic. Note that certain items are only recommendations.

Summary of Automatic Placement Prerequisites

Placement Prerequisite	Interactive Mode	Automatic Mode
Package and padstacks	Required	Required
Netlist	Required	Required
Package	Not required	Required
Placement grid	Not required You can define a non-etch/conductor grid for interactive placement.	Required
Physical rooms	Required only if floorplanning	Required only if floorplanning
Placement tag to designate components for placement	Required unless placing by room	Required unless placing by room
Design outline defined with class BOARD GEOMETRY and subclass OUTLINE	Recommended	Recommended
Package symbols and height restricted areas	Recommended	Recommended

Related Topics

- Keepin and Keepout Areas
- Creating a Floorplan Using Rooms
- Working with Symbols

Setting Placement Grids

You can use grids to help control component placement. Manual placement and the interactive mode of automatic placement use a non-etch/conductor grid, which is optional. All modes of automatic placement require an automatic placement grid.

Creating a Non-Etch/Conductor Grid for Manual and Interactive Placement

A non-etch/conductor grid is a snap grid for placing components manually and in the interactive mode of automatic placement. It is called a "non-etch/conductor" grid because the grid is *not* used for routing. You can set an absolute control grid size and location for a non-etch/conductor grid.

The origin of the non-etch/conductor grid is the origin of a design file (0,0). The layout editor uses the route grid to complete pin-to-pin connections. You might want to keep the non-etch/conductor grid compatible with the route grid to reduce the number of off-grid pins.

To create a non-etch/conductor grid and control its display, choose *Setup – Design Parameters* and the *Display* tab of the Design Parameter Editor (`prmed` command) or the `define grid` command.

Creating a Grid for Interactive and Automatic Placement

The layout editor requires that you create a placement grid for the area where you run any mode of automatic placement. Automatic placement places packages only at the intersections of the placement grid lines. The grid has a class of BOARD GEOMETRY and a subclass of PLACE_GRID_TOP.

Placement grids can have the following characteristics:

- Non-uniform spacing between lines
- "Punctures" at grid line intersections to prevent placement at those locations
- Different grid spacing for each placement area (rooms or the entire design)

Even if the current placement area is only a room, you must still establish a grid for the entire package keepin area.

Once you complete placement for an area, you can make any necessary adjustments or generate a new grid for the next placement area. Continue this process for each subsequent area.

To create a window placement grid, choose *Place – Autoplace – Top Grids* (place set topgrid command) or *Place – Autoplace – Bottom Grids (* place set bottomgrid command). Menu items and commands are described in the *Allegro PCB and Package Physical Layout Command Reference*.

If you have not created a package keepin area for the automatic placement area, first create the package keepin.

Editing Placement Grids

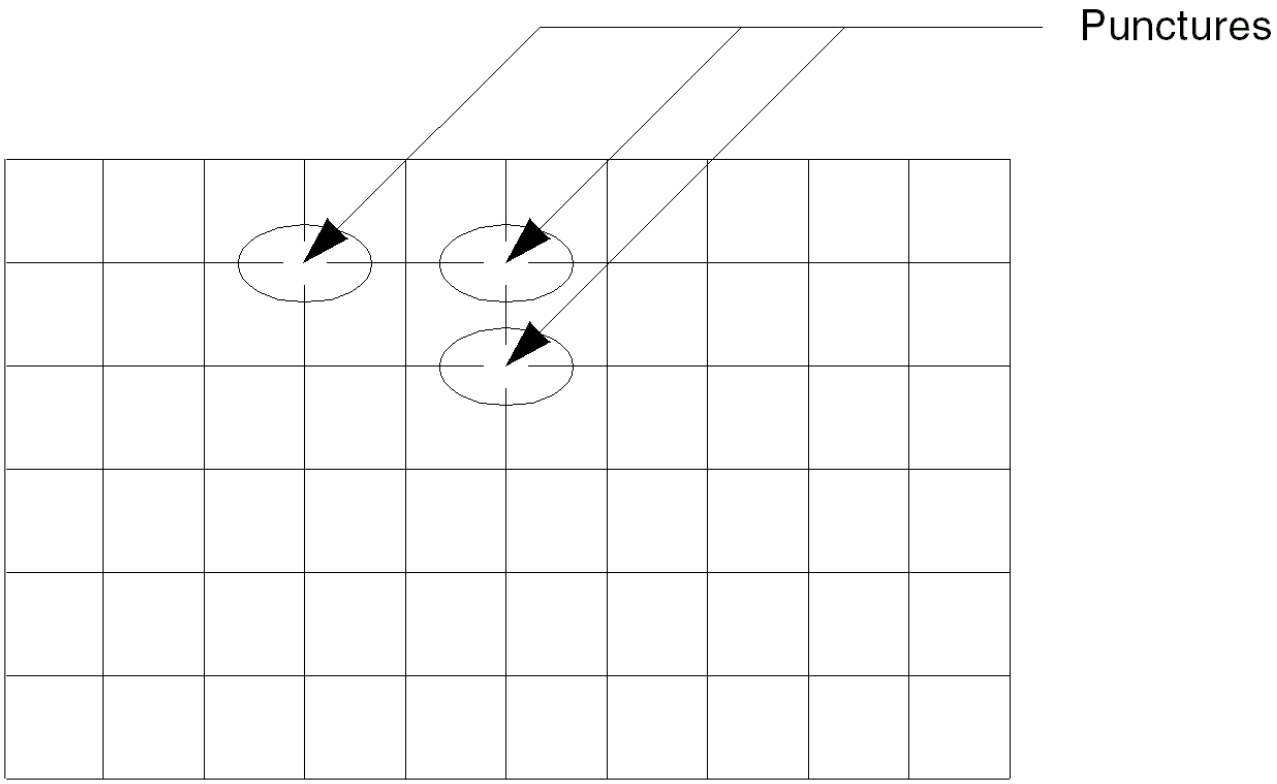
Grid-line intersections are package location for all modes of automatic placement. You can create a non-uniform grid to accommodate irregularly spaced components by choosing the following for grid lines:

- *Add – Line* (`add line` *command)*
- *Edit – Move* (`move` *command)*
- *Edit – Copy* (`copy` command)
- *Edit – Delete (delete* command)

Puncturing Intersections

You can create a point on the automatic placement grid where a component cannot be placed by removing a piece of grid line called a puncture, as shown in the the following figure. By deleting intersections, you can exclude package locations for automatic placement from some design areas. Puncturing lets you eliminate a single intersection point without removing an entire line.

Grid Line Punctures



To puncture a grid line, you use the `delete` command (menu-driven editing mode only).

- prmed
- define grid
- Adding Graphic Elements to a Design
- *add line*
- *move*
- *copy*
- *delete*

Assigning Placement Properties

You can assign placement properties to components, functions, and nets using *Edit – Properties* (`property edit` command), described in the *Allegro PCB and Package Physical Layout Command Reference*.

The following table identifies properties that affect placement and indicates where in this chapter they are described.

Properties That Affect Placement

To...	Use This Property...	See...
Identify components for both modes of automatic placement or placing specific components manually	PLACE_TAG	Identifying Components for Placement
Fix component placement	FIXED	Fixing Component Placement
Keep related components in the same room	ROOM	Keeping Related Components in the Same Room
Control DRC error reporting for rooms	ROOM_TYPE	Keeping Related Components in the Same Room
Keep related components close together	COMPONENT_ WEIGHT	Keeping Related Components Close Together

Keep related components on a net close together	WEIGHT	Keeping Related Components on a Net Close Together
Classify symbol definitions into user- defined classes, to which package-to-package DFA spacing values are defined on the DFA Constraints Dialog spreadsheet, available by choosing *Setup – DFA Constraint Spreadsheet* (`dfa_spreadsheet` command) for the class default.	DFA_DEV_CLASS	Assigning Symbols to Groups for Real-Time DFA

Identifying Components for Placement

Use the PLACE_TAG property to identify:

- Specific components for manual placement when you choose *Place – Manually* (`place manual` command).
- Components you are placing automatically

> ⚠ When you run automatic placement, be sure to remove any previously assigned PLACE_TAG properties by using the *Remove Tag* option in the Automatic Placement dialog box, accessed when you choose *Place – Autoplace – Parameters* (`place param` command).

PLACE_TAG Property and Cluster Feature

During automatic placement, the PLACE_TAG property works together with the *Cluster* option in the Automatic Placement dialog box accessed when you choose *Place – Autoplace – Parameters* (`place param` command). When *Cluster* is enabled, placed components (those with or without the *PLACE_TAG* property) affect the positioning of unplaced components:

- Automatic placement considers placed components outside the placement area when those components have the PLACE_TAG property.
- Automatic placement considers placed components *inside* the placement area when those components do NOT have the PLACE_TAG property.

If *Cluster* is turned off, all components outside the placement area influence both selection and positioning of unplaced components. Therefore, you can use the *Cluster* feature to select which components outside a placement area influence the placement in that area.

Fixing Component Placement

Use the FIXED property to prevent a component from being moved after placement. You can still change other parts of the symbol, such as text, lines, or rectangles.

If you try to move a component that has FIXED assigned, the layout editor displays a message explaining you cannot perform the operation.

Keeping Related Components in the Same Room

Use the ROOM property in conjunction with the ROOM_TYPE property to identify a component to place in the same room by automatic placement. The ROOM_TYPE property has several values from which you can choose to control DRC error reporting under various placement situations.

Both modes of automatic placement place components with an attached ROOM property having a value matching the name of the active room, even if they do not have a PLACE_TAG property attached.

Keeping Related Components Close Together

Use the COMPONENT_WEIGHT property to tell both modes of automatic placement how closely to position heavily connected components.

You might want to initially run automatic placement with a weight greater than 50 on the connector, using the whole design. If you add this property to more than one component on a net, the effective weight is multiplied on the net.

Keeping Related Components on a Net Close Together

Attach the WEIGHT property to a net to indicate how important it is for the components on the net to place close together during automatic placement.

While the primary purpose of the WEIGHT property is to keep a net (with a high weight) short, automatic placement uses the WEIGHT property to keep the components on the net close together.

Assigning Symbols to Groups for Real-Time DFA

You can classify symbol definitions into a user-defined class, to which package-to-package Design for Assembly (DFA) spacing values (defined in the DFA Constraints Dialog spreadsheet) for the class default by using the DFA_DEV_CLASS property. The DFA Constraints Dialog spreadsheet is available by choosing *Setup – DFA Constraint Spreadsheet* (`dfa_spreadsheet` command).

To add or remove symbol definitions from user-defined classes, or determine the symbol definitions with the DFA_DEV_CLASS property assigned to them, you use the DFA Classification Editor dialog box, available by clicking Show symbol classifications... on the DFA Constraints Dialog spreadsheet. The layout editor treats these classes as components comprised of symbols to which the DFA spacing values defined for the class default. For example, fifty versions of an 0805 package symbol may exist, all complying to the same DFA rule set. A single class line entry in the spreadsheet assumes the rules for each instance of the 0805 class of package symbols.

Clicking *Update* on the DFA Classification Editor dialog box assigns the DFA_DEV_CLASS property to the symbol definitions in the classes you specified.

Related Topics

- property edit
- dfa_spreadsheet
- place manual
- place param
- PLACE_TAG
- COMPONENT_WEIGHT
- Placing Elements Manually
- Creating a Floorplan Using Rooms

Creating a Floorplan Using Rooms

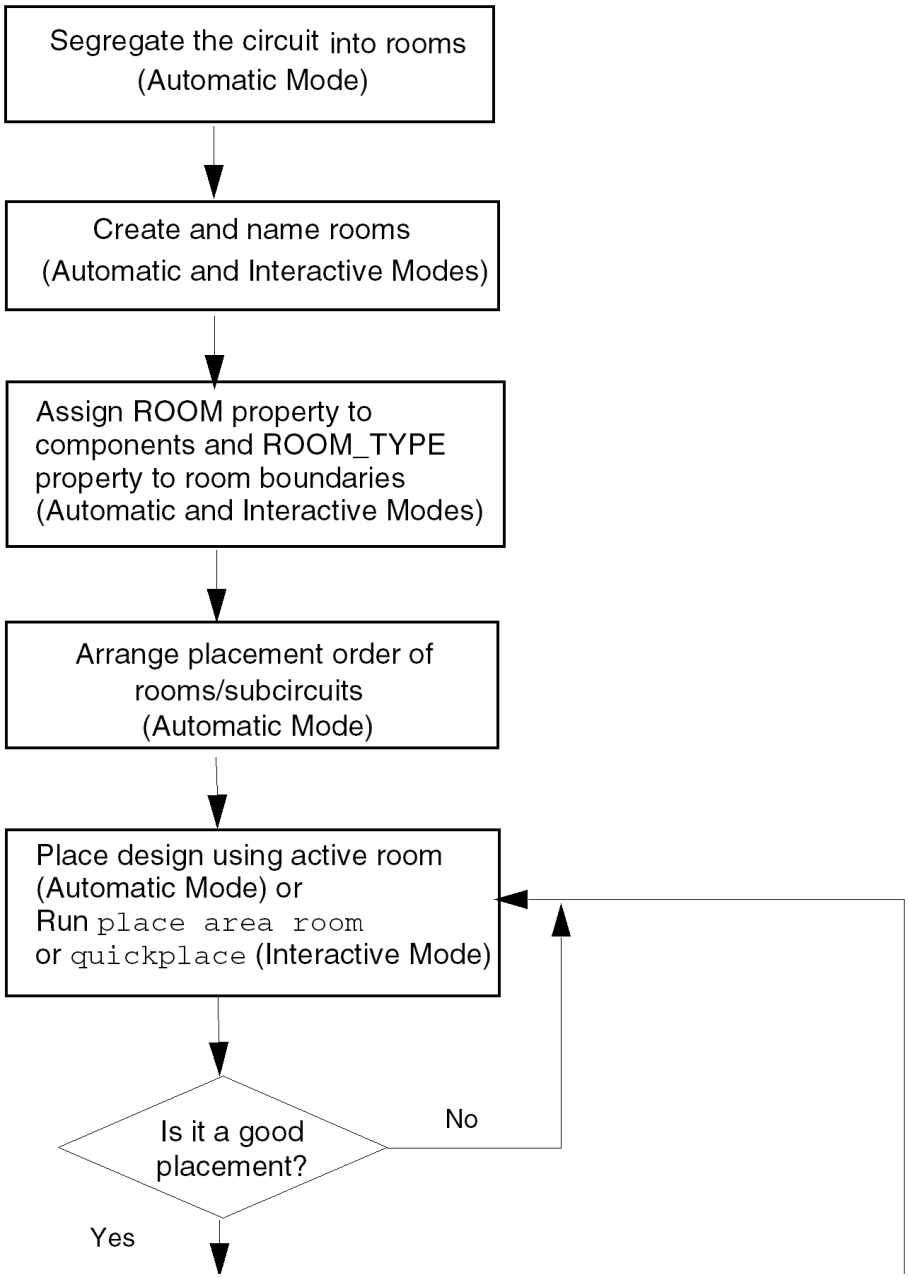
Floorplanning entails organizing and creating rooms in which to place specific components. Rooms

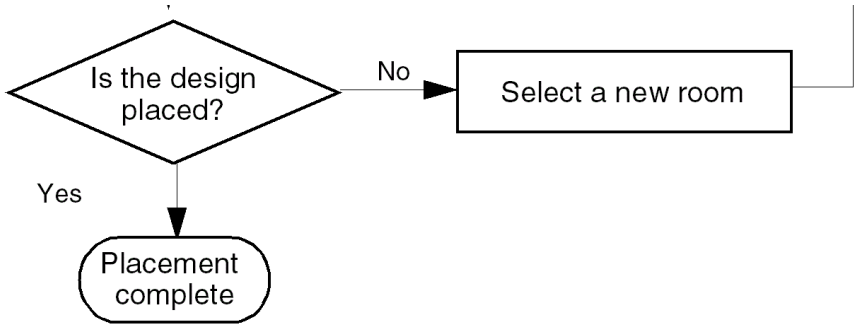
are rectangular areas that you create on the top and bottom sides of the board. When you choose a room as a placement area, that room becomes the active area for placement processing.

Using the ROOM and ROOM_TYPE properties in conjunction with each other, you can then restrict components to specific rooms, allowing you to functionally segregate a design and arrange components that must remain close together, as well as flag DRC errors when certain placement situations arise.

The following figure illustrates using rooms during floorplanning.

Using Rooms in Automatic Placement (Interactive and Automatic Modes)





Using the ROOM and ROOM_TYPE Properties

Use options on the Room Outline dialog box, available by invoking *Setup – Outlines – Room Outline* (`room outline` command), to create rooms and name them. Because the layout editor treats room boundaries as closed polygons on the TOP_ROOM, BOTTOM_ROOM, or BOTH_ROOMS subclasses of the BOARD GEOMETRY class, ensure that the *Options* tab has these settings.

You then assign that room name to the appropriate components with the ROOM property. (Another method of creating rooms is to choose *Add – Rectangle* (`add rect` command) and then *Add – Tex*t (`add text` command) to assign a room name.)

The layout editor uses the ROOM property name to map the components to the specified rooms. You can attach a room property to components during schematic creation (if using Allegro Design Entry HDL), netlist creation, or any time during physical design.

To further refine component placement in various rooms, you can attach the ROOM_TYPE property to a room boundary by invoking *Setup – Outlines – Room Outline* (`room outline` command). Another method of attaching the ROOM_TYPE property to a room boundary is to use *Edit – Properties* (`property edit` command).

By specifying HARD, SOFT, INCLUSIVE, HARD_STRADDLE, or INCLUSIVE_STRADDLE as the value of the ROOM_TYPE property, you dictate whether or not DRC errors occur, and under what placement conditions, as the table below illustrates:

Component	Soft	Hard	Hard_straddle	Inclusive	Inclusive_straddle
Member in room	No DRC	No DRC	No DRC	No DRC	No DRC

Member straddling room	No DRC	DRC	No DRC	DRC	No DRC
Member outside of room	No DRC	DRC	DRC	DRC	DRC
Non member in room	No DRC	DRC	DRC	No DRC	No DRC
Non member straddles room	No DRC	DRC	DRC	No DRC	No DRC

The ROOM_TYPE property may also be set on the root of the design, which then controls behavior for any rooms without an assigned ROOM_TYPE property. If no ROOM_TYPE property exists at either the room or the design level, no DRC error reporting occurs.

Examples of ROOM and ROOM_TYPE Properties' Effect on Placement

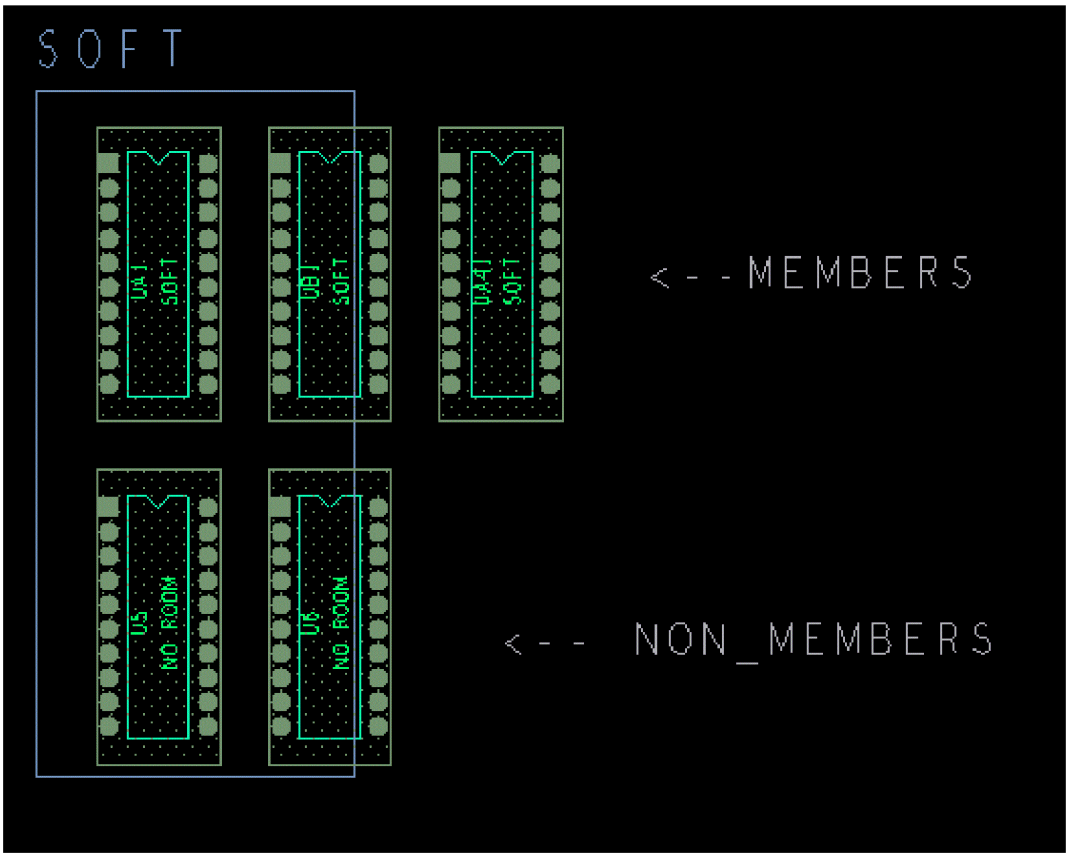
The following table illustrates how the ROOM and ROOM_TYPE properties interact when various components are placed.

Component	ROOM Property Value/ROOM_TYPE Property Value	DRC error
UA1	SOFT/SOFT	No
UB1	SOFT/SOFT	No
UA41	SOFT/SOFT	No
U5	None assigned/SOFT	No
U6	None assigned/SOFT	No
UA4	HARD/HARD	No
UB4	HARD/HARD	Yes
UB14	HARD/HARD	Yes
U1	None assigned/HARD	Yes

U2	None assigned/HARD	No
UA42	HARD_STRADDLE/HARD_STRADDLE	No
UB42	HARD_STRADDLE/HARD_STRADDLE	No
UA22	HARD_STRADDLE/HARD_STRADDLE	Yes
U7	None assigned/HARD_STRADDLE	Yes
U8	None assigned/HARD_STRADDLE	No
UA3	INCLUSIVE_STRADDLE/INCLUSIVE_STRADDLE	No
UB3	INCLUSIVE_STRADDLE/INCLUSIVE_STRADDLE	No
UA43	INCLUSIVE_STRADDLE/INCLUSIVE_STRADDLE	Yes
U9	None assigned/INCLUSIVE_STRADDLE	No
U10	None assigned/INCLUSIVE_STRADDLE	No
UA5	INCLUSIVE/INCLUSIVE	No
UB5	INCLUSIVE/INCLUSIVE	Yes
UA15	INCLUSIVE/INCLUSIVE	Yes
U3	None assigned/INCLUSIVE	No
U4	None assigned/INCLUSIVE	No

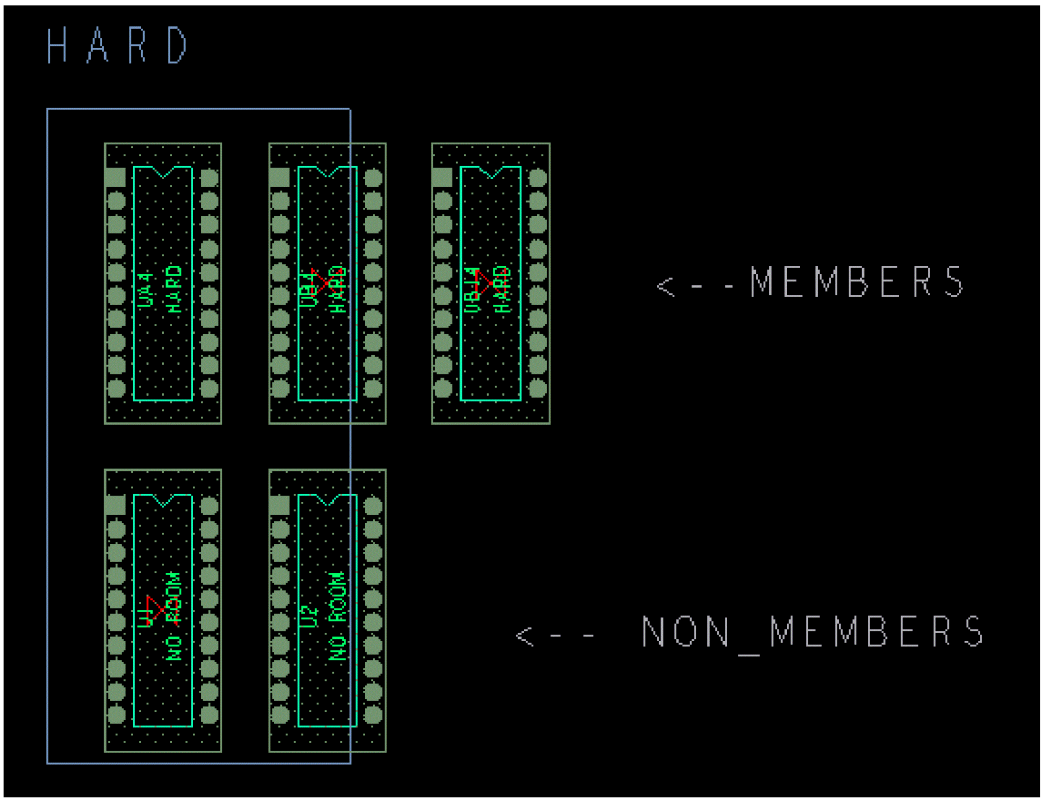
For the room named SOFT, no DRC errors occur for any components placed in it, as follows.

Component Placement with ROOM_TYPE Property of SOFT



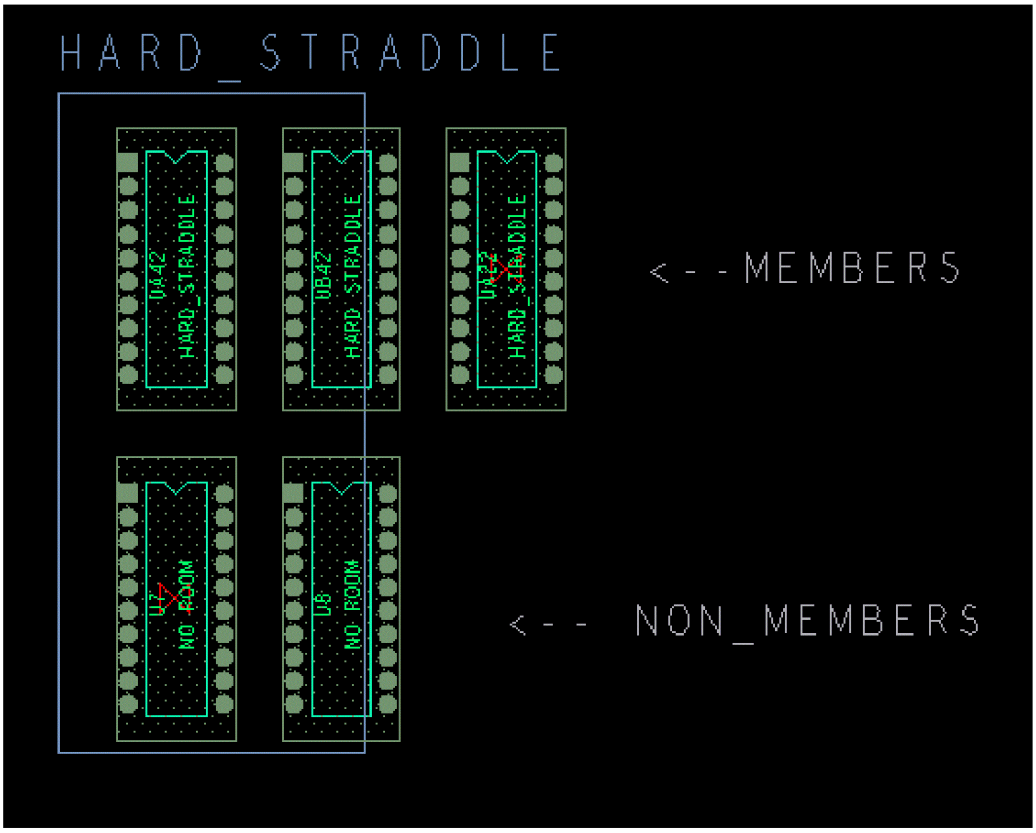
For the room named HARD, components belonging to it may reside entirely within its room boundary. DRC errors occur when you place a component outside this room. Any components that are not members of this room, yet are placed entirely within the room boundary, cause DRC errors, such as U1, as follows.

Component Placement with ROOM_TYPE Property of HARD



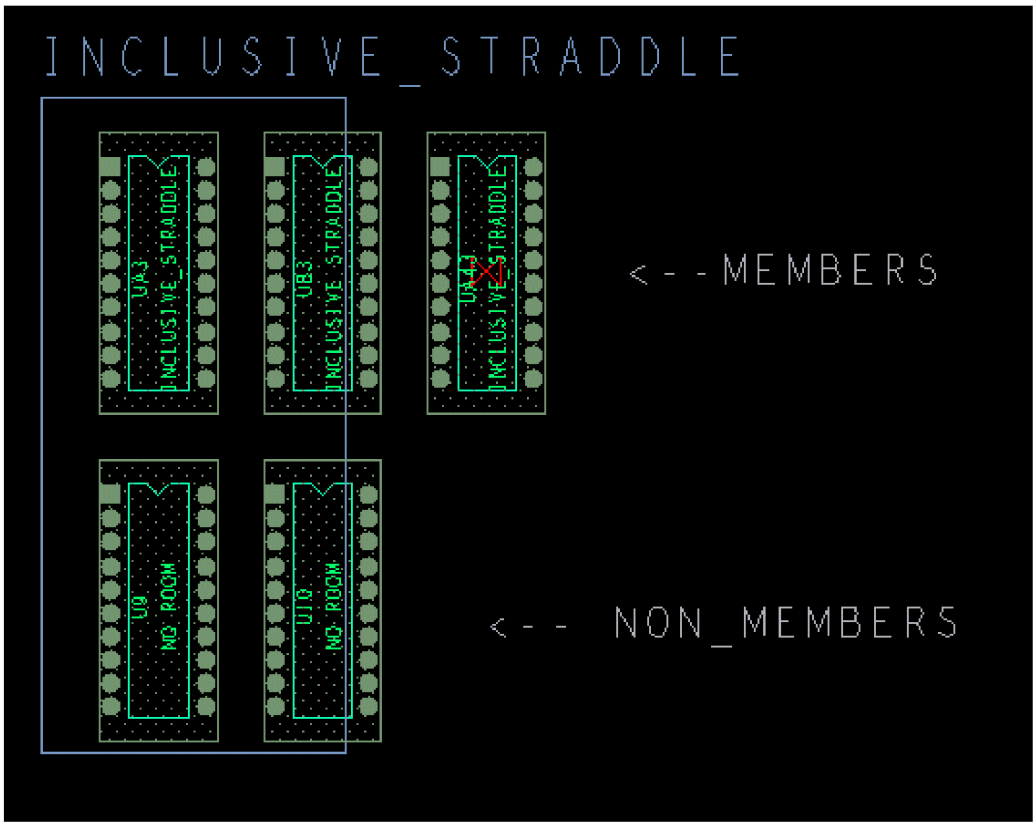
For the room named HARD_STRADDLE, components belonging to this room may straddle the room boundary without generating DRC errors. DRC errors occur when components are placed completely inside the room boundary, such as UA 22 as follows.

Component Placement with ROOM_TYPE Property of HARD_STRADDLE



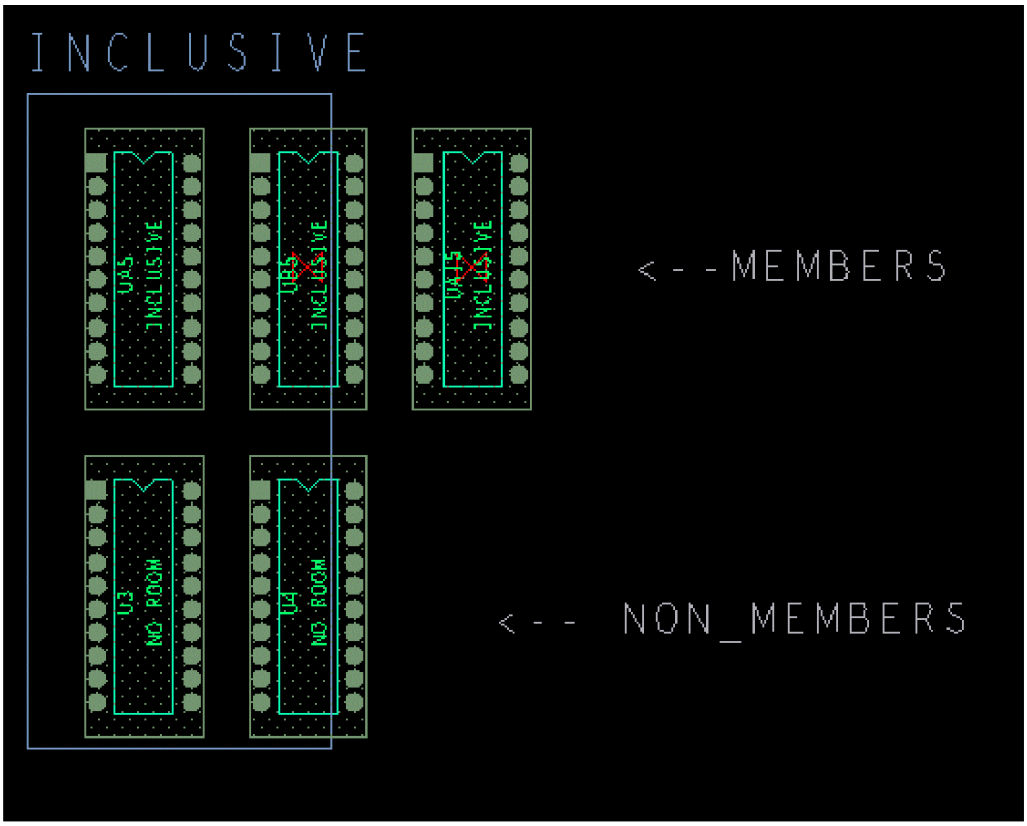
For the room named INCLUSIVE_STRADDLE, all components may reside entirely in the room or to straddle the boundary without generating DRC errors. DRC errors occur when components belonging to this room are placed entirely outside it, such as UA43 as follows.

Component Placement with ROOM_TYPE Property of INCLUSIVE _STRADDLE



For the room named INCLUSIVE, DRC errors occur when components belonging to it straddle the room boundary, such as UB5 and UA15 as follows.

Component Placement with ROOM_TYPE Property of INCLUSIVE



Related Topics

- Placing Elements Automatically
- room outline
- add rect
- room outline
- property edit

Specifying Timing Data

If you want automatic placement to be aware of timing data, you must supply that data in a `crit.dat` file, which must be located in the same directory as the design.

⚠ When you use a `crit.dat` file, choose *Place – Autoplace – Design* (place area design command) to specify the area for automatic placement. The menu item and command are described in the *Allegro PCB and Package Physical Layout Command Reference*.

The `crit.dat` file must contain the following information about each critical path:

- Critical path name
- Total allowable length
- One or both of the following:
  - Included pin-pair definitions
  - Alternate pins for the beginning or end of the path

Use a text editor to create the ASCII file that you name `crit.dat`. The format of the `crit.dat` file is explicit:

- Each critical path definition begins with the keyword `PATH`, on a line with fields that specify the name of the path and the path length definition.
- Each alternate net definition or pin-pair definition for the critical path appears on a separate line.
  - Alternate net definitions begin with the keyword `ALT`.
    Use `ALT` where the total path length is critical and the layout editor can choose the critical connection from the nets you provide as alternatives.
    `ALT` can exist only as the first or last definition in the critical path.
    The `ALT` designation gives automatic placement the flexibility to try different placement possibilities, because only the longest of the alternatives needs to be considered in the distance calculation.
    A single `ALT` line may occur in a file when the definition does not require a specific pin-to-pin designation.
  - The pin-pair definitions begin with the keyword `PP`, and alert the layout editor that the

corresponding pin-pair must be calculated in the order presented.

The following is an example `crit.dat` file:

Example crit.dat File

```
PATH  SEL0-RAS0 .45n
ALT NET U33.1
ALT NET U33.2
ALT NET U33.3
PP  U33.7 U13.12
PP  U13.11 RP6.5
ALT RP6.12 NET
PATH  SEL1-RAS1 5000
ALT NET U33.1
ALT NET U33.2
ALT NET U33.3
PP  U33.9 U13.9
```

Critical path SEL0-RAS0

Critical path SEL1-RAS1

Creating a crit.dat File

1. In the directory that contains the design, use a text editor to open a file called `crit.dat`.
2. Begin each critical path definition using the following format:

   `PATH <pathname> <maximum distance>`

`PATH`	Keyword that begins a critical path definition. All fields following it, until the next occurrence of PATH, define the critical path.
<pathname>	Identifies the critical path. Type a character string with a maximum length of 63 characters.
<maximum distance>	Specifies either a maximum length or a maximum time. In a length string, processing assumes the numbers are in user units. To specify a time string, provide the number and append either `n` (to indicate the value is in nanoseconds) or `p` (for picoseconds).

3. Specify the first definition in the critical path.
   If the total path length is critical, use the `ALT` format to specify the first definition in the critical path.

   `ALT NET <refdes>.<pin>`

<refdes>	Specifies the reference designator.

<*pin*>	Specifies the pin number.

You must use the same format for all subsequent `ALT` lines for the same connection. All `ALT` lines for a single net must be consecutive and for the same component.
For example

```
ALT NET U33.1

ALT NET U33.2

ALT NET U33.3
```

If your critical path must follow an exact order defined by pin-pairs, type the pin-pairs in the following format:

```
PP <refdes>.<pin> <refdes>.<pin>
```

4. Specify additional path definitions, as required.
   If `ALT` is the last definition in the critical path, the format must be:

   ```
   ALT <refdes>.<pin> NET
   ```

5. Save the file.

6

Placing Elements Manually

Before you can proceed with manual placement, you must complete certain tasks, as well as determine design requirements.

Using manual placement, you can place all components individually, or you can place components of the same type during one pass. For example, you can place all input/output components one at a time, then all ICs one at a time, then all discretes. Using the Quickplace feature lets you put all unplaced components outside the board outline of a design.

If your system includes automatic placement, you can alternate manual with automatic placement to:

- Preplace sensitive or fixed components
- Rearrange particular components placed automatically
- Optimize the overall placement of a design

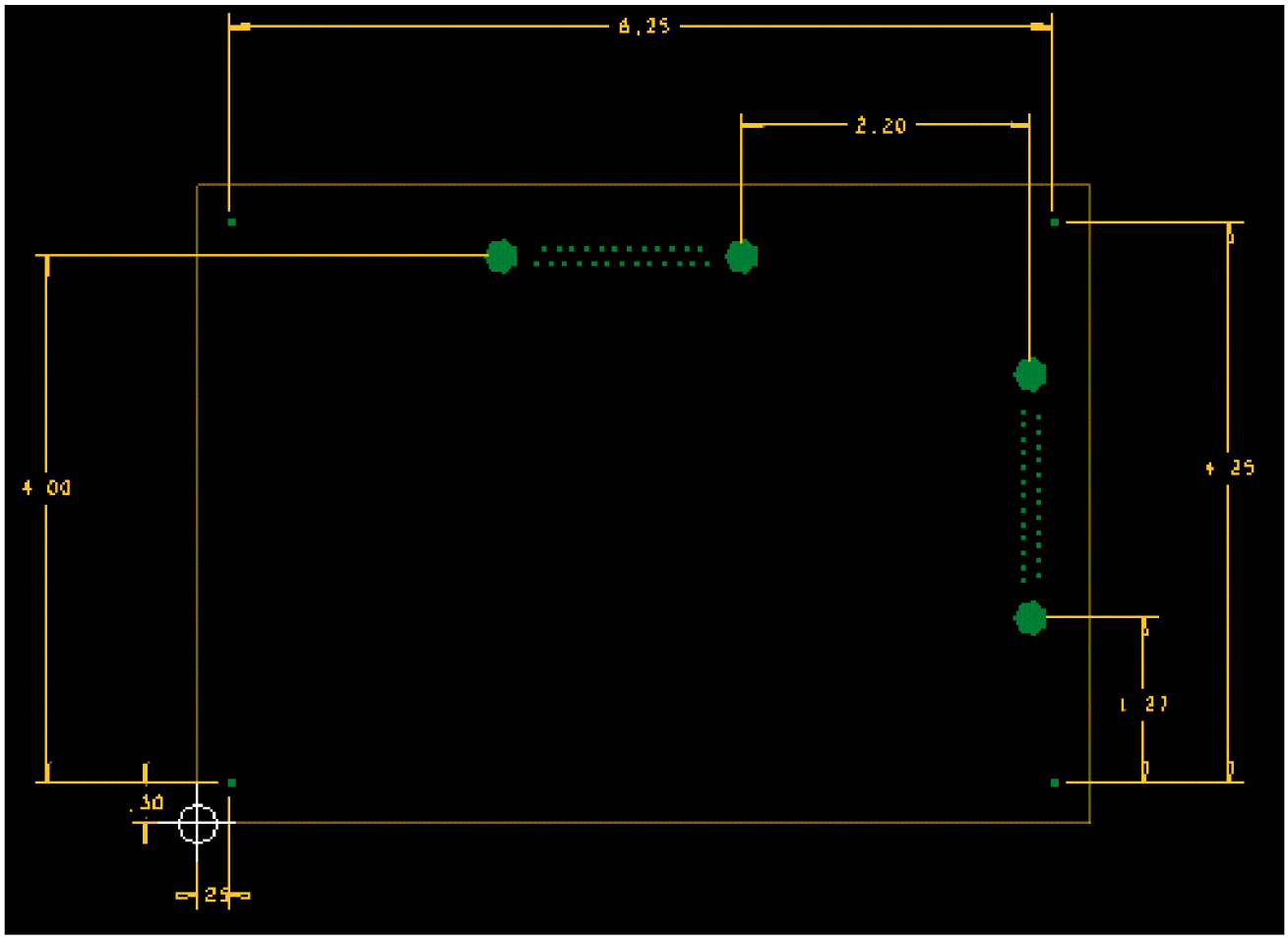
You can generate a report on the placed or unplaced status of the design at any time.

Changing the Drawing Origin during Placement

To change the drawing origin during placement, you can specify an exact point on the canvas as the location for the drawing origin using *Setup – Change Drawing Origin* (`chg origin` command). A circle with crosshairs designates the drawing origin. To display the origin regardless of grid visibility, enable the *Display Origin* parameter in the *Display* tab of the Design Parameter Editor, available by choosing *Setup – Design Parameters* (`prmed` command).

In the following figure, six locations require precise locations: four mounting holes and two connector mounting holes. Once you dimension the lower-left mounting hole from the board origin, you can easily gauge the placement of the upper-left mounting hole in relation to it. Then change the origin to the upper-left mounting hole location, place the upper-right mounting hole, change the origin to the upper-right hole location, and so on.

Using the Drawing Origin for Placement



Related Topics

- Preparation for Placing Elements
- Reviewing Placement Status and Results
- chg origin
- prmed

Accessing Manual Placement Features

Manual placement offers options for placing specific groups of unplaced elements in a design. You can place any of the following:

- ICs
- I/Os
- Discretes
- Symbols (package, mechanical, format)
- Module instances
- Module definitions

The layout editor places only the elements that you specify in the *Placement* dialog box. Each element in the specified group displays individually, with its symbol origin on the cursor.

Using the Placement Dialog Box

You place design elements by choosing *Place – Manually* (place manual command), which invokes the *Placement* dialog box, described in the *Allegro PCB and Package Physical Layout Command Reference*. The tabbed user interface allows you to choose for placement (either from the database or from the library files to which you have access) symbol types, components by reference designator, and modules.

Accessing Options in the Placement Pop-Up Menu

Various placement operations also feature a pop-up menu that lets you access options for controlling the operation.

To use the pop-up menu during a placement operation:

1. Choose the appropriate placement option.
2. Click right to display the pop-up menu.
3. Click right to make a selection from the menu.

The following selections are standard to many pop-up menus:

Done implements any changes made during the operation and quits the current mode (for example, swap function mode or move mode).

Oops cancels the last operation you performed in the current mode and keeps you in that mode.

Cancel cancels the current operation and quits the mode.

Next implements the last operation and keeps you in the current mode.

Move changes the position of the selected element and keeps you in that mode.

Mirror relocates the currently selected symbol you want to add to the drawing to the opposite side of the board.

Rotate rotates the currently selected symbol.

These selections are unique to *Place – Manually* (`place manual` command).

Alt Module chooses a different module definition for the current module instance.

Alt Symbol chooses a different symbol for the current symbol selection.

Show appears on the pop-up menu if you selected *Autohide* on the Placement dialog box. Selecting *Show* from the pop-up menu causes the Placement dialog box to re-appear once all elements have been placed. If *Autohide* is not selected, you must use *Show* to display the Placement dialog box.

Placing Symbols

You can place a package, mechanical, or format symbol in a design. Then you assign a reference designator to a package symbol, described below. This method of placement can be useful when you place components before the netlist is completed.

⚠ One instance of mechanical symbol is allowed in a board if it contains a route or place keepin.

Assigning Reference Designators to Package Symbols

Choose *Logic – Assign RefDes* (assign refdes command), described in the *Allegro PCB and Package Physical Layout Command Reference*, to assign a reference designator to a placed package symbol.

You can specify the increment to use when assigning a group of reference designators to symbols. For example, use an increment of 1 to assign reference designators U1, U2, U3, and so on. Use an increment of 10 to assign reference designators U10, U20, U30, and so on.

Reference designators must already be in the database as defined in the netlist. To do this, choose *File – Import – Logic* (netin command), described in the *Allegro PCB and Package Physical Layout Command Reference*. You add electrical data for the symbol using the netlist as well.

Placing Alternate Symbols

When you place a symbol, you can use one of the following:

- The symbol, which is initially attached to the cursor
- The alternate symbol, by choosing the *Alt Symbol* option from the pop-up menu to attach an alternate symbol to the cursor or the *Alternate Symbol* option from within Placement Application Mode

You can use alternate symbols for components as you place them in a design if you previously attached the ALT_SYMBOLS property type to the components using the Cadence schematic-capture tools Allegro Design Entry HDL or CIS. ALT_SYMBOLS defines an alternate package symbol that can be substituted for the primary package symbol. Or, if you are using a third-party schematic, in the device file, assign the ALT_SYMBOLS property to components by specifying a PACKAGEPROP property record. To do that, see Specifying Definition Properties in a Device File in the *Allegro X PCB Editor User Guide: Defining and Developing Libraries*.

When you place an alternate symbol, the layout editor shows a list of alternate symbols that are valid for the processed subclass (for example, TOP symbols if TOP is being placed). The valid side is the TOP, unless you chose *Mirror* on the *Options* tab of the control panel or in the *Drawing Options* dialog box, in which case, it is the BOTTOM.

You can limit the primary package symbol to the TOP or BOTTOM by using the ALT_SYMBOLS_HARD property. For more information on this property, see ALT_SYMBOLS_HARD in the *Allegro Platform Properties Reference*.

⚠ When building symbols, do not create them for the BOTTOM side of the design. Instead, build all symbols for the TOP and then mirror the symbols when placing to the BOTTOM.

Mirroring and Alternate Symbols

The *Mirror* option mirrors whatever symbol is appropriate for that layer. If the symbol changes, the layout editor displays the name of the new symbol in the message line. When you use alternate symbols, the *Mirror* option uses only the symbol and the alternate symbol you choose from the pop-up menu. If no alternate symbol exists, the symbol attached to the cursor mirrors. If the current symbol is valid for the other side of the design, it is used; otherwise, the first alternate symbol found in the property value for the side being mirrored appears with the cursor.

Placing Symbols Using Real-Time Design for Assembly DRC

During interactive placement, real-time Design For Assembly (DFA) design rules hone the precision and accuracy of package-to-package clearances. You create an external rules-driven table to define spacing rules between symbol definition pairs with the DFA Constraints Dialog spreadsheet, available by choosing *Setup – Constraints – DFA Constraint Spreadsheet* (dfa_spreadsheet command). These spacing values can then be applied to a design while the external file remains intact on disk. You can also assign the DFA_DEV_CLASS property to symbols and create a class in the DFA Classification Editor (available by clicking Show symbol classifications... on the DFA Constraints Dialog spreadsheet) to which the spacing values defined for the class default.

After you define DFA rules between specific components, as you place components, spacing circles appear on screen that highlight potential DFA rule violations when components are placed closer than specified DFA spacing constraints, or if no constraints are supplied, when DFA place-bounds touch or intersect. These spacing circles indicate the spacing value you set for the components when placement achieves tangency; that is, placed components adhere to DFA spacing rules without touching or intersecting. The placement grid you defined determines the degree of tangency. A 1-mil grid allows the greatest potential for tangency, for instance. To eliminate any cursor hesitation during placement of DFA-governed components, you can choose

Setup – User Preferences (enved command) to set the `dfa_pause` environment variable to zero.

If the spacing value is 50 mil, you can instantiate the component at the spacing circle location where tangency is achieved, or at a location that triggers a DRC if you wish to ignore the DRC feedback. For more information on meeting DFA requirements, refer to sing Dynamic Design for Assembly (DFA) Constraints.

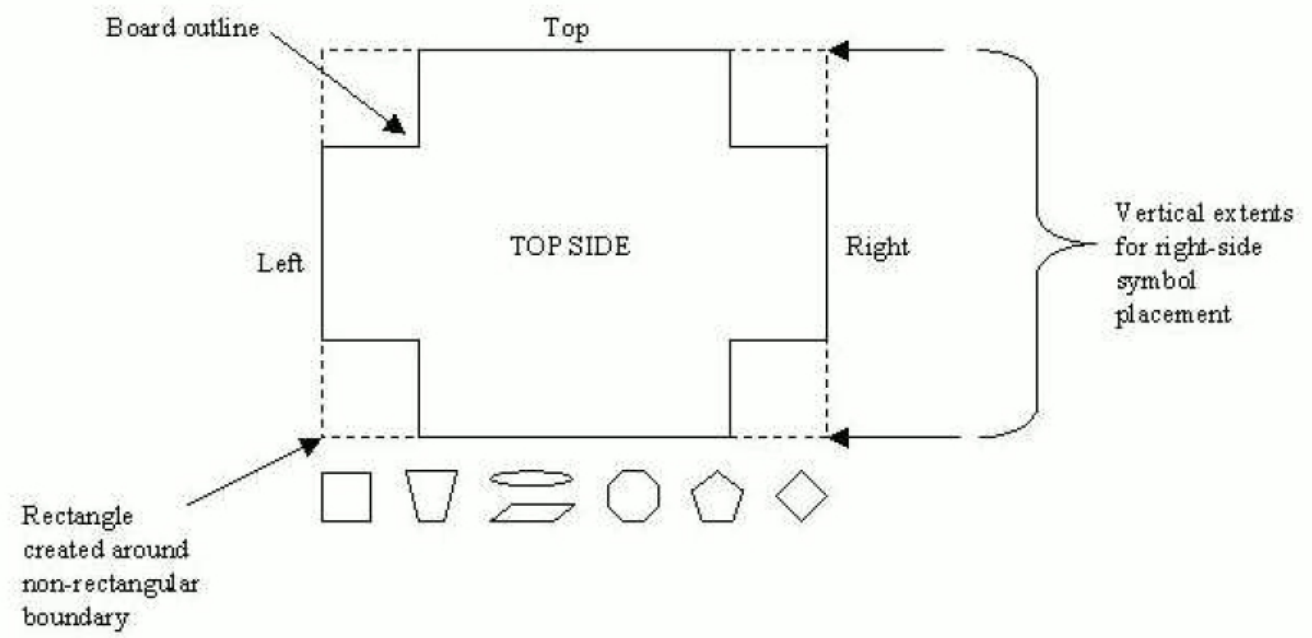
Adding Unplaced Components with Quickplace

The Quickplace feature lets you easily add unplaced components to the board design. You place logic bearing symbols outside the board outline, in either a default or user-defined configuration, creating a palette of symbols that you can view, filter, and move into the design. You also can place components by property/value, room, schematic page number (only with Allegro Design Entry HDL as the front end), or selection.

Choose *Place – Quickplace* (quickplace command) to filter the types of components you want to place on the outside edges of the board outline.

The layout editor places components in a non-rotated state along the edge of the board geometry boundary. If Quickplace does not detect a board outline, an error message appears. If the boundary is not rectangular, Quickplace creates a minimum outside rectangle (not a design element) whose minimum and maximum extents are the outer edges of the boundary geometry, as follows.

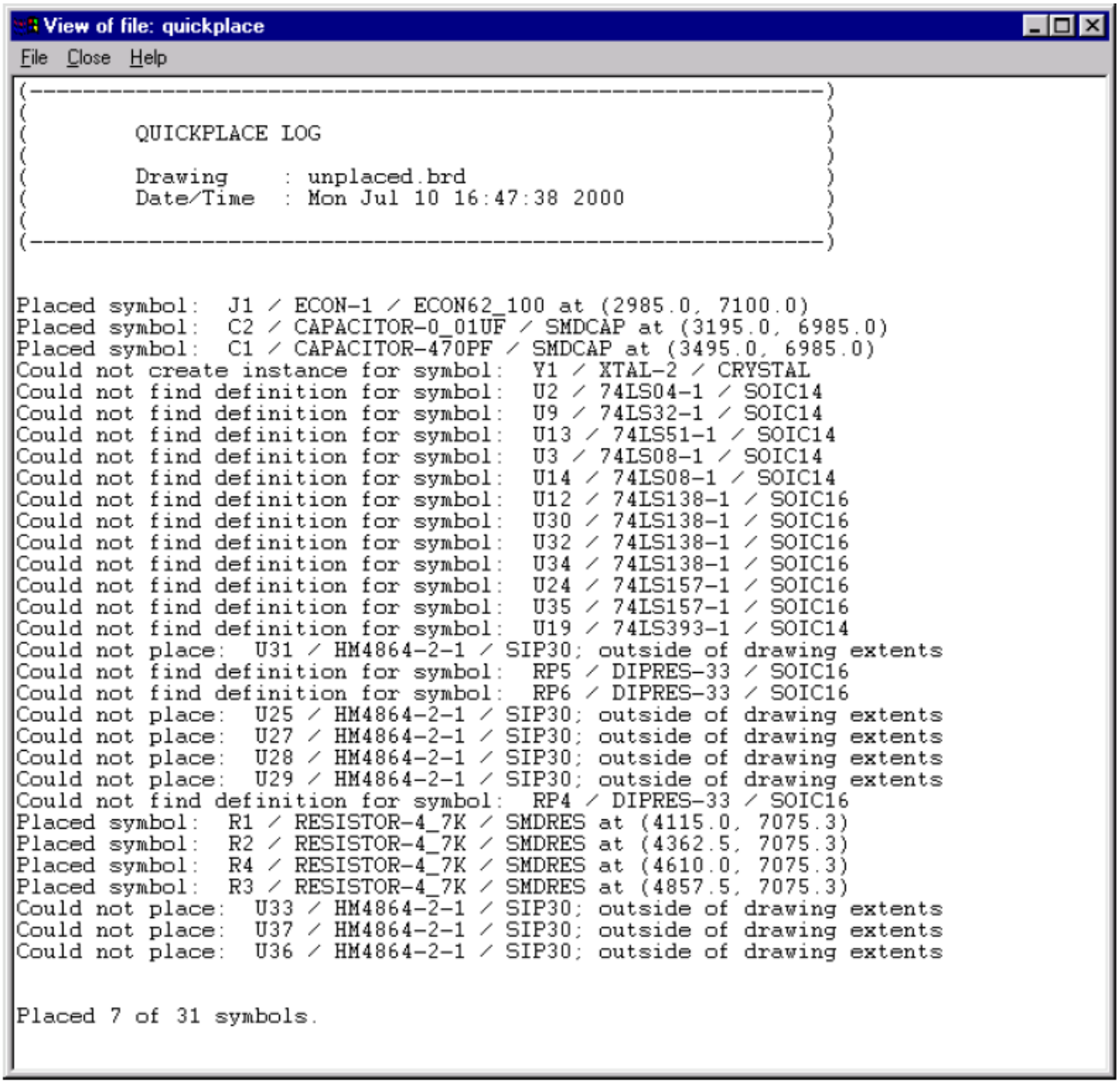
Board Geometry Rectangle



Independent of which edge or layer on which you elect to place them, Quickplace places the components adjacent to one another based upon their extents.

Once you place the components, a message at the bottom of the dialog box displays status. You can access the log file directly from the dialog box to get details of the placement operation, as follows.

Example of quickplace.log File



Using Quickplace in a Design Partition

Quickplace can also place components within a design partition.

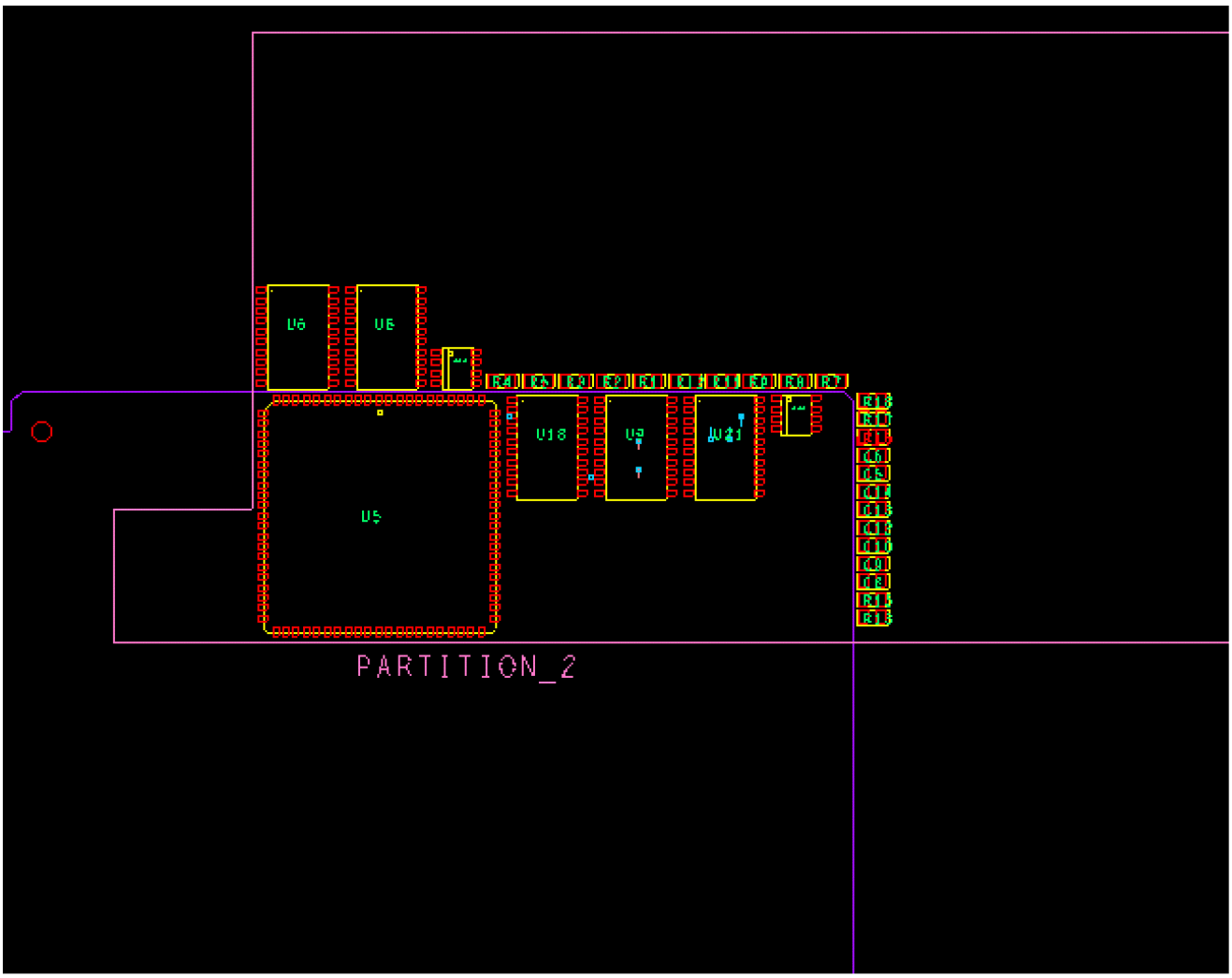
In the Quickplace dialog box, choose which components to place in the *Placement Filter* section.

Next, choose *Place by Partition* and select the partition name.

Once you click *Place* to add the components, Quickplace begins component placement on the top of the board until full, and then resumes placement on the bottom of the board. Quickplace toggles between the top and bottom of the board, continuing to place components in an overlapped, offset pattern, until all components are placed.

If the partition boundaries extend beyond the design boundaries, Quickplace continues to place the components along the partition edges, as follows.

Partition Boundaries that Extend Beyond Design Borders



Editing Nets

During the placement phase of a design, you may need to edit the logical connections of sets of vias and pins (otherwise known as nets). This section covers the net editing methodology. Choose *Logic – Net Logic* (net logic command), described in the *Allegro PCB and Package Physical Layout Command Reference*.

Net editing functionality includes:

- Creating new nets
- Editing pin assignments
- Renaming nets
- Removing nets
- Shorting nets

Nets that you have brought in to a design from a schematic capture may have been connected together with pseudo pins assigned with the NET_SHORT property. The NET_SHORT assignment eliminates DRCs that would otherwise occur from creation of the pseudo pins.

In the layout editor, you can short unconnected nets together without generating DRCs in a similar fashion by adding the NET_SHORT property to pins and vias that bridge planes. The syntax of the NET_SHORT property is:

```
<net1>:<net2>:...
```

For example:

```
NET_SHORT = GND1:GND2
```

Use *Edit – Properties* or the `property edit` command to choose vias to which to attach the NET_SHORT property. This procedure is described in "Assigning a Property to a Design Element".

Replacing Temporary Symbols

The layout editor lets you replace temporary symbols you may have created on the fly in SPECCTRAQUEST with information from your product library. To do this, choose *Place – Replace SQ Temporary – Devices* (replace temp_symbols command), described in the Allegro PCB and Package Physical Layout Command Reference.

Placing Components Using a Text File (PlaceText)

The layout editor lets you place components on a design using the ASCII text file `place_txt.txt`, which specifies component positions and orientations. This placement file is created primarily for use as input for a future design. These are the commands you can use:

- The plctxt out command generates the `place_txt.txt` file from an existing design.
- The plctxt in command places components in a new or an existing design using the `place_txt.txt` file.
- The plctxt command places or exports components in batch mode using the `place_txt.txt` file.

All these commands are described in the *Allegro PCB and Package Physical Layout Command Reference*.

When you use a placement file as input for a design, the `place_txt.txt` file must be in your current directory.

Make sure that the devpath and psmpath variables in your environment file point to the components you are loading. Type set at the command prompt to display the Defined Variables window and view the devpath and psmpath strings. To set or edit these variables, choose *Setup – User Preferences (enved* command), described in the Allegro *PCB and Package Physical Layout Command Reference*. The `devpath` environment variable is in the *Config_path*s category, and psmpath is in the *Design_paths* category.

This section contains reference information on the default placement file, `place_txt.txt`. This information may be useful if you decide to edit any of the data in the file.

A placement file contains up to six columns of numbers. Each line describes the placement of a single component, as follows:

```
REFDES x y <r> <m> <s>
```

`REFDES`	Reference designator
`x`	X-location coordinate
`y`	Y-location coordinate
r	Rotation angle The angle can be a whole number or a fraction. Default is 0 degrees.

m	Mirrored Default is no mirroring: place on TOP.
s	Symbol name This specification is used to indicate an alternate symbol. The default is to use the default symbol from the device file.

The reference designator and X,Y coordinates are the only elements that always appear for every component.

The following shows part of a sample `place_txt`.txt file. Notice that all four components use a DIP16 package (no alternate symbols). The first two are not rotated; the last two are rotated 90 degrees. The space between the rotation column and the symbol column indicates no components are mirrored.

Sample place_txt.txt File

UUNITS = MILS

R2 2275.000 4275.000 270 RC0805

R1 2350.000 5150.000 0 m RC0805

U2 2350.000 4575.000 90 DIP14_3

U1 2025.000 4975.000 0 m DIP14_3

You can use any text editor to edit the placement file. Observe the following conventions:

- Use white-space and comments, as required.
- Enclose comments in parentheses ( ).
- Use either upper- or lowercase letters. The layout editor always stores reference designators and symbol names in uppercase.
- Decimal points are acceptable when specifying locations and angles.

The layout editor outputs the units of a design when it creates a placement file. When you use the file as input to a new design, the layout editor looks for the design units and presents the output in the units appropriate to the new design.

7

Placing Elements Automatically

Before you can proceed with automatic placement, you must complete certain tasks, as well as determine design requirements.

The layout editor can place elements automatically based on controls that you specify before activating automatic placement. You choose the elements to be placed and define an area in which to place them. You can specify additional placement controls by defining placement parameters, and properties assigned to functions and elements.

You should manually place a seed component to control automatic placement within an area.

Automatic Placement Modes

The layout editor provides two modes for running automatic placement:

- Automatic mode

In this mode, the layout editor quickly places all tagged elements on the design. You can then examine the placement and reposition any elements that are not optimally placed, using automatic placement, manual placement, or automatic or interactive swapping.

- Interactive mode

In this mode, the layout editor places the tagged elements on the design one at a time. It optimally locates each component and then allows you to accept or edit the location.

About Setting Automatic Placement Parameters

Before running automatic placement, you must set specific parameters that dictate:

- Which placement algorithm to use
- Which point on the component to use as the origin
- What weights to apply to determine component positioning
- How to handle elements that it cannot place

- Whether to allow elements to overlap
- Whether to allow placement outside the placement area
- Whether to cluster heavily connected components
- How to display ratsnesting
- Whether to remove place tags upon completion
- How to specify timing data during placement
- How many iterations to perform, to a maximum of 10

T0 set automatic placement parameters. You cans Set the parameters interactively using the Automatic Placement dialog box, described in Setting Automatic Placement Parameters Interactively.

⚠ When you set the parameters interactively, you can also run automatic placement in automatic mode.

Before Setting Automatic Placement Parameters

Regardless of which method you use to set parameters, you need to do the following before setting them:

- Create a placement grid.
- Define package keepins and, if necessary, rooms into which you are placing design elements.
- Assign the PLACE_TAG property and any other properties to the components you are placing.

Design Level Messages

E-Missing placement keepin rectangle

You have not defined a package keepin for use by automatic placement.

W-Unplaced components are not PLACE TAGGED.

Automatic placement requires one of the following:

- PLACE_TAG properties on the components to place
- An active room (that is, the placement area is a room) and components with the corresponding ROOM property

W-No preplaced component is in area you are placing

You have not placed a seed component interactively. If you do not place a seed component, automatic placement places one for you. For optimal results, place the seed component for the active area manually.

Grid Level Messages

E-A placement grid was not found

Choose *Place – Autoplace – Top Grids* (`place set topgrid` command) or *Place – Autoplace – Bottom Grids (*`place set bottomgrid` command) to define the grid for automatic placement.

E-Grid subclass doesn't match the requested side.

You have not defined the automatic placement grid for the side you have chosen for placement. Choose *Place – Autoplace – Top Grids (*`place set topgrid` command) or *Place – Autoplace – Bottom Grids (*`place set bottomgrid` command) to define top or bottom or both grids for automatic placement.

Messages When Room is the Active Area

E-Room was not found

Be sure that you have added the room to the design and that you specified the correct room name in the *Area* definition.

E-Subclass of room does not match the placement grid

You have not defined the automatic placement grid for the side of the design where the active room is located. Choose *Place – Autoplace – Top Grids (*`place set topgrid` command) or *Place – Autoplace – Bottom Grids (*`place set bottomgrid` command) to define top or bottom or both grids for automatic placement.

E-Subclass of room doesn't match the requested side

The subclass of the active room identifies a different side of the design from the side you specified for placement.

W-No unplaced component matches active room

You have requested automatic placement for an active room whose components (with matching ROOM property) are all already placed in the design. Automatic placement places in the room any unplaced components with a PLACE_TAG property.

Setting Automatic Placement Parameters Interactively

⚠ Be sure to follow the tasks listed in Before Setting Automatic Placement Parameters.

You set automatic placement parameters interactively by using the Automatic Placement dialog box. This dialog box also allows you to run automatic placement in automatic mode.

If you choose to also run automatic placement from this dialog box, be aware that automatic placement run in automatic mode finishes quickly. One way to become more familiar with the placement parameters is to run automatic placement using the default settings on the Automatic Placement dialog box. After you study the results, adjust one placement parameter at a time interactively, and rerun automatic placement for the same placement area.

As the automatic placement process occurs within a placement area, the weighted parameters help automatic placement make decisions based on connection to the seed component.

To set automatic placement parameters and/or run automatic placement in automatic mode, choose *Place – Autoplace – Parameters* (`place param` command).

Tips on Applying Weights

When you apply weights in the Automatic Placement dialog box:

- Try different weight values and rerun the placement program to observe the results. Cadence recommends that you run placement on one area of the design so, if results are unsatisfactory, you can change the parameters and rerun automatic placement.
- To determine which parameters give you the best placement results, adjust one parameter at

a time.
For example, if rotation is important to you, concentrate on accomplishing the desired rotation and use the default values for direction, mirror, and straightness.

- Values you set for direction, mirror, and straightness affect automatic swapping.

The WEIGHT and COMPONENT_WEIGHT properties also affect swapping.

Running Automatic Placement

Prerequisites

Before running automatic placement in either mode, be sure to do the following:

- Perform all the necessary prerequisites, including defining a package keepin and an automatic placement grid.
- Adjust the parameters, described in Setting Automatic Placement Parameters Interactively.
- Update the symbol library.

⚠ Manually placing a seed component in the placement area helps assure a more appropriate placement. If you fail to place a seed component, one is chosen.

Automatic Placement and Your Placement Area

Automatic placement requires you to determine the placement area. The area can be:

- The entire design (the default)
  Automatic placement uses the package keepin boundary, not the board outline. If you are placing a design that uses timing data in a `crit.dat` file.
- A room inside the package keepin. The room that you specify as the placement area becomes the active room.
- A window you draw inside the package keepin.

In automatic mode, automatic placement looks at each component with a PLACE_TAG property attached and performs placement operations based on the controls you set, until it has placed each tagged component.

If there are no unplaced components with the PLACE_TAG property attached, and the active placement area is a room, the layout editor places any components with the ROOM property attached with a value matching the active room name.

Automatic Placement of Alternate Symbols

The following responses to specific conditions occur during automatic mode:

- If you defined alternate symbols as property values, the layout editor uses the package symbol valid for the side of the design being placed.
- If only the TOP side is set up for automatic placement, the layout editor places only the original package symbol and ignores any existing alternate symbols defined for the TOP side.

Ways of Running Automatic Placement

Your placement area determines how you run automatic placement. A description of the modes appears in Automatic Placement Modes.

Placement Area	Menu Items/Commands	Notes
Design	Choose or run in this order: • *Place – Autoplace – Parameters* (`place param` command — automatic mode) • place execute (automatic mode) • *Place – Autoplace – Design* (`place area design` command— interactive mode)	The layout editor chooses the area within a package keepin and places components automatically in that area.

Room	Choose or run in this order: 1. *Place – Autoplace – Room* (`place area room` command) 2. Then choose of run any of these: ◦ *Place – Autoplace – Parameters* (`place param` command — automatic mode) ◦ `place execute` (automatic mode)	The layout editor places components having a ROOM property set to the name of the specified room automatically in that room. The room must be located in a package keepin.

Window	Choose or run in this order: 1. *Place – Autoplace – Window* (`place area window` command) 2. Choose or run any of these: ◦ *Place – Autoplace – Parameters* (`place param` command — automatic mode) ◦ `place execute` (automatic mode) ◦ *Place – Interactive* (`place interactive` command — interactive mode)	The layout editor places components into the window you draw within a package keepin on the design.

⚠ To view the current, active area of the design for automatic placement, choose *Place – Autoplace – List* (`place area list` command).

When placement is complete, you can review the placement log, any best placement results design file, and placement reports.

Related Topics

- Preparation for Placing Elements
- Placing Elements Manually
- Swapping Components, Functions, and Pins
- Automatic Placement Prerequisites
- Setting Placement Grids
- Assigning Placement Properties
- reating a Floorplan Using Rooms
- place set topgrid
- place set bottomgrid
- place param
- Automatic Placement Prerequisites
- Specifying Timing Data
- place area design
- place area room
- place area window
- place execute
- place interactive
- place area list
- Reviewing Placement Status and Results

8

Placing Embedded Components

While designing the physical layout, layout designers have the option of placing components on the internal PCB layers. Components that are placed on internal PCB layers are called embedded components. This chapter covers the tasks to be performed while designing a layout with embedded components.

⚠ The features discussed in this chapter are available with the *Miniaturization* option only.

Specifying Component Placement

A component can be placed as an embedded component if one of the following situations is true.

- The EMBEDDED_PLACEMENT property is attached to the component and the property value is set to Required or Optional.
- The EMBEDDED_SOFT property is assigned to the drawing database, with its value set to TRUE.
- The DUAL_SIDED_COMPONENT property, is assigned to the symbol in the Symbol Editor, with its value set to TRUE.

Removing unassigned symbol vias is primarily done to free up space for routing. The EMB_INDIRECT_VIA_SUPPRESS property is assigned to the component definition, component instance or symbol pin. If a component is placed on an Indirect Attach embedded layer, this property suppresses all via pads associated with the component if the pin is not on a named net.

The indirect attach via pads will only be restored if:

1. The symbol pin changes to a named net.
2. The symbol is moved to a layer which is not indirect attach.
3. The property is removed.

9

Swapping Components, Functions, and Pins

You can individually swap components, pins, and functions—interactively or automatically—to refine the placement of a design; for instance, to decrease the average wire length or uncross ratsnest lines.

Interactive Swapping

The swap commands let you interactively swap individual elements to refine design placement. The following sections describe how to swap:

- Components
- Functions (gates)
- Pins

Swapping Components

Swapping components interactively does not affect etch/conductor, so any etch/conductor attached to a swapped component does not move with the component.

You cannot swap components under the following circumstances:

- When the FIXED and/or NO_SWAP_COMP property is assigned to either component
- When a component contains a pin on a net that is assigned the FIXED property
- When attempting a swap that would cross a module boundary

If you swap a component with any of the constraints listed above, a message informs you that the swap failed.

When the layout editor performs the component swap, it assigns to each swap candidate the X,Y coordinates, rotation, and mirroring that, before the swap, were associated with the swap partner.

To swap components interactively, choose *Place – Swap – Components* (swap componentscommand).

You can also execute the command from within the Placement Edit application mode, in which the command functions in a pre-selection use model: You choose an element first, then right click and execute the command from the pop-up menu. placement application mode, which provides an intuitive environment in which commands used frequently during placement are readily accessible from right mouse button pop-up menus, based on a selection set of design elements you have chosen.

The placement edit application mode can be activated in several ways. You can:

- Choose the menu option *Setup – Application Mode – Place Mode*
- Enter `placemode` in the Command Console window.
- Click the appropriate toolbar icon (if added to your toolbar).
- Right click and choose *Application Mode – Placement Edit*

In the pre-selection use model, the command is only available if the selection set comprises exactly two components that you have chosen. If you choose components and clines, for example, a warning displays for each invalid element, and the tool ignores it.

Swapping Functions

Functions to be swapped must:

- Have the same function type and device type
- Be part of components with the same value and tolerance
- Have common pins on the same nets

The following property assignments restrict swapping functions interactively:

- A GROUP property attached to either function requires both functions to belong to the same group.
- A NO_SWAP_GATE_EXT property on the function means that functions are swappable only if they are in the same component.
- A SWAP_GROUP property on the function means that functions are swappable only within their swap group, as specified by this property. The SWAP_GROUP property is assigned during the F2B processing to any schematic symbol with the HAS_FIXED_SIZE = n property. The value of this property is the *spath* to the schematic symbol. The *spath* is displayed in the *Swap Group* column of the *Swapping* worksheet in of the *Properties* domain in Constraint Manager.

A function cannot be swapped if any of the following properties are assigned:

- A FIXED property on either net
- A FIX_ALL property on the component containing the function
- A NO_SWAP_GATE property on either the function or component

When the layout editor swaps functions, it replaces any etch/conductor connected to the functions with ratsnest lines.

To run interactive function swapping, choose *Place – Swap – Functions* (swap functionscommand).

Swapping Pins

You can swap two pins if their names occur in the same PINSWAP statement of their device file. All connecting etch/conductor is removed. All changes can be backannotated. Pins in a function with a NO_SWAP_PIN or a FIXED property attached are ineligible for swapping.

The following figure shows an example entry for device type 74LS00 that illustrates a device file where the two NAND2 input pins are swappable. Note the line containing the PINSWAP keyword.

Device File with Swappable Functions

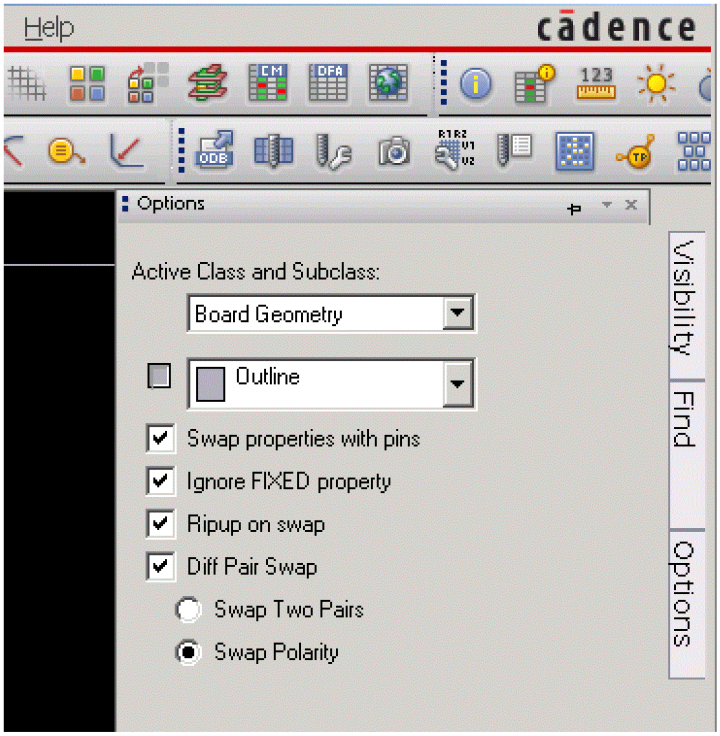
```
PACKAGE DIP14
CLASS IC
PINCOUNT 14
PINORDER              74LS00    A  B     Y
PINUSE                        74LS00   IN  IN    OUT
PINSWAP                       74LS00   A   B
FUNCTION                      G1  74LS00   1   2     3
FUNCTION                      G2  74LS00   4   5     6
FUNCTION                      G3  74LS00   9  10     8
FUNCTION                      G4  74LS00  12  13    11
POWER   +5V ; 14
```

To swap pins interactively, choose *Place – Swap – Pins (swap pins*command*)*. You can also execute the command from within Placement Edit application mode by selecting the first pin, then clicking the right mouse button and choosing *Swap Pins* from the popup menu. There are several command options available on the Options panel as shown in the following figure.

Swapping Differential Pair Pins

With the *Diff Pair Swap* option enabled in the Options panel, you can activate one of two differential pair pin swap modes. *Swap Two Pairs* mode lets you swap two pins at one end of a differential pair with two pins at one end of another differential pair. *Swap Polarity* mode lets you swap the negative and positive pins at one end of a single differential pair.

Pin Swap Options



Swap Two Pairs Precedence

When you choose one pin of the initial pin pair to swap, Allegro determines the eligibility of other differential pairs as well as appropriate pin polarity and highlights other component pins to swap with. The following precedence is used in determining eligibility in cases where there are conflicting differential pair definitions.

1. Model-defined differential pair
2. Library-defined differential pair
3. User-defined differential pair (using _P, _N, and +, - in the physical net names)

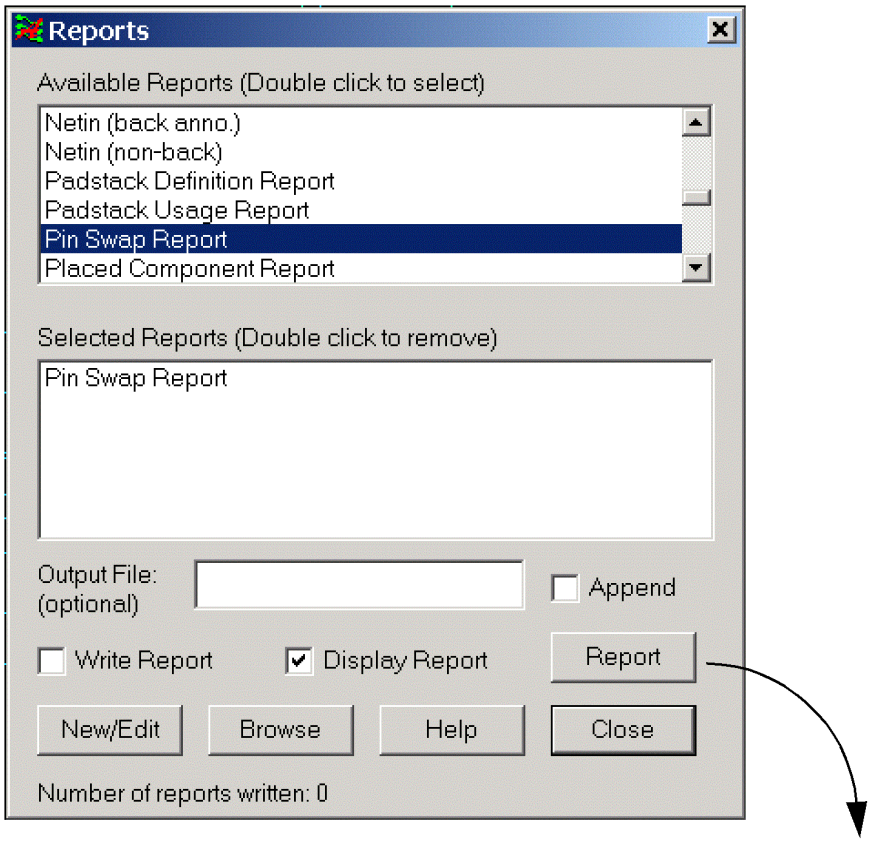
For details on defining differential pairs in library parts, see the Allegro Design Entry HDL - *Part Developer User Guide.*

For details on how differential pair pin swapping is handled in the Allegro flow, see the Allegro System Architect - *System Connectivity Manager User Guide.*

Pin Swap Report

You can create a standard Pin Swap report using the Reports dialog box as follows.

Reports Dialog Box and Sample Pin Swap Report



Pin Swap Report

REFDES	PIN_NUMBER	LAST_PIN_SWAP	NET_NAME	FUNC_DES	FUNC_SLOT_NAM
U1	1	U1.2	N1_N	TF-4	G1
U1	2	U1.1	N1_P	TF-4	G1
U3	1	U3.2	N1_N	TF-8	G1
U3	2	U3.1	N1_P	TF-8	G1
U3	8	U3.8	N4_P	TF-7	G2
U3	9	U3.9	N4_N	TF-7	G2

All pins are tracked using the property LAST_PIN_SWAP that is attached to the pin. The property value is the name of the pin with which it was last swapped.

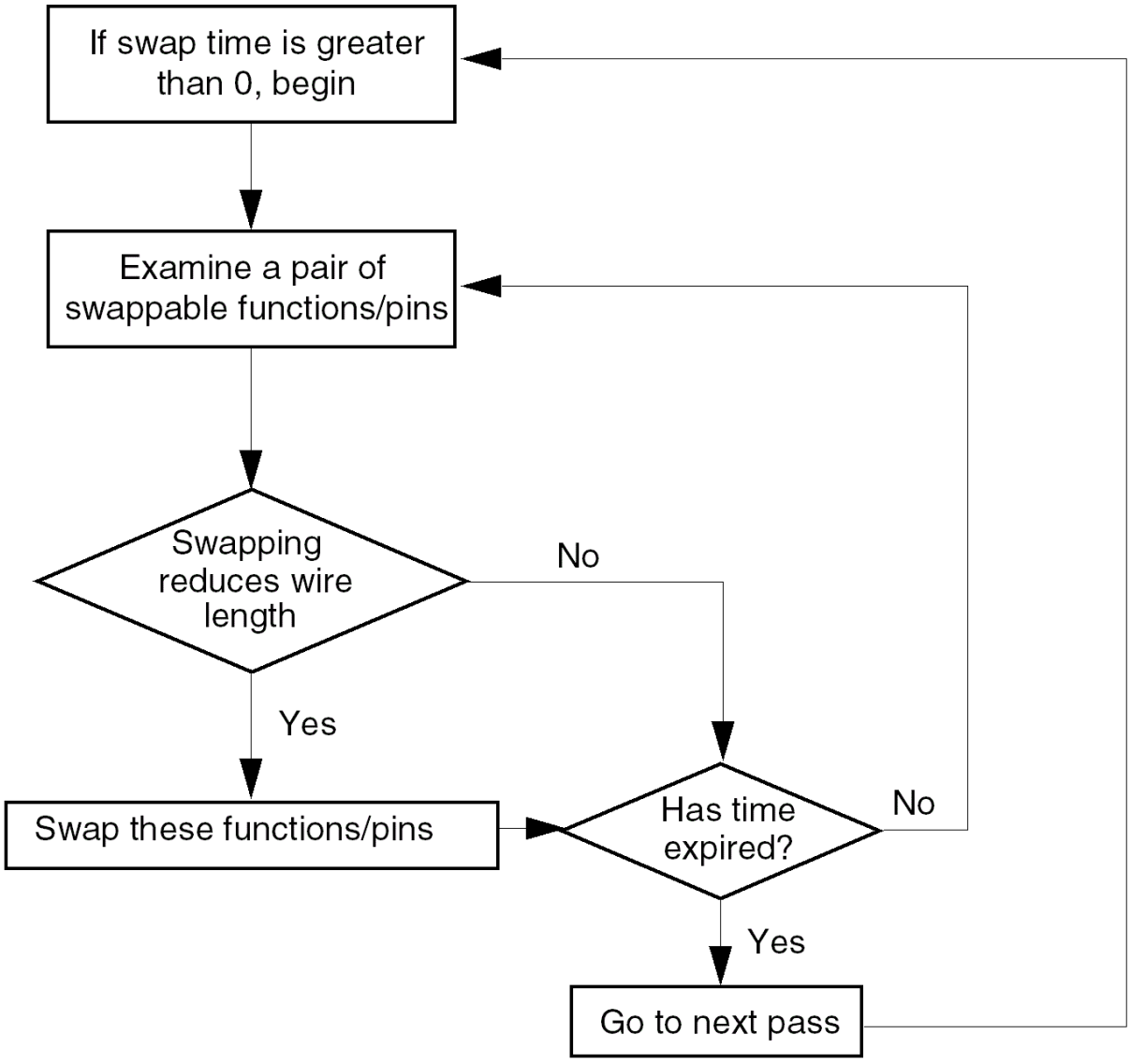
⚠ Updating an Allegro design from the schematic or third party netin causes all pin swap properties to be deleted.

Automatic Swapping

You can automatically swap pins and functions. You determine how automatic swapping occurs by setting swap parameters and assigning certain swap properties to reference designators, functions, and nets.

You can define up to 10 passes for automatic swapping. During each pass, the layout editor looks first at functions to swap, then pins. Each pass ends when the layout editor either runs out of time or cannot find any more logical swap candidates.The following figure describes a single pass.

Automatic Swap Flowchart



Prerequisites

Before running automatic swap, you must do the following:

- Create a package keepin.
- Place at least two components of the same device or function type on the design
- Set the swap controls, as explained in the following section.

Controlling and Running Automatic Swapping

You must set the following controls before running automatic swap:

- Assign appropriate swap properties to selected reference designators, functions, and nets
- Set appropriate placement parameter weights in the Automatic Placement dialog box (How Automatic Placement Parameters Affect Swapping)
- Define the swap area (Defining the Swap Area)
- Set the swap parameters (Setting the Swap Parameters and Running Automatic Swap)

The following sections describe how to set the swap controls.

Setting Swap Properties

Automatic swapping checks the swap parameters and certain properties when examining each function and pin for possible swap. Properties assigned to nets, components, and functions can affect the likelihood of a swap.

The layout editor also considers the following weights when deciding whether a function or pin is swappable:

- WEIGHT property on nets
- COMPONENT_WEIGHT property on reference designators
- Weights assigned as automatic placement parameters.

You can assign placement properties to components, functions, and nets by choosing *Edit – Properties* (property edit command).

Setting Component Properties

You can assign the following properties to a reference designator:

FIX_ALL	Allows no swapping on this component of its functions or of its pins.
NO_SWAP_GATE	Indicates that the functions (that is, gates) within the component cannot be swapped.
NO_SWAP_GATE_EXT	Indicates that the functions in the component can be swapped only within the component.
NO_SWAP_PIN	Prevents pin swapping on this component.
NO_SWAP_COMP	Is not implemented.

Setting Function Properties

You can assign the following properties to a function designator:

NO_SWAP_GATE	Indicates that the function cannot be swapped. It stays fixed in its current slot in its current component.
SWAP_GROUP	Indicates that the function can be swapped only with functions in the same swap group (that is, have the same value for the SWAP_GROUP property).
NO_SWAP_GATE_EXT	Indicates that this function cannot be swapped with one from another component. However, the function can be swapped between slots within the component.
NO_SWAP_PIN	Indicates that pins in this function cannot be swapped.
GROUP	Attached to either function targeted for swap, requires that both belong to the same group.

Setting Net Properties

You can assign the properties described below to a net to affect automatic swapping:

FIXED	Prevents any automatic program from changing the net. For swapping, this property prevents automatic swap from swapping pins on the net or swapping functions involving the net.
WEIGHT	Assigns a value from *0* to *100* to a net. The value that you assign indicates the importance of the length of a particular net in relation to the other nets in the design.
	The weight value tells automatic swap how important it is to shorten the net. If it is vital for a certain net to be as short as possible, then assign a high value to the property when assigning it to the net.

The following examples show how the layout editor views the WEIGHT property:

- A value of *0* means do not consider the net for swapping.
- Each value less than 50 progressively decreases the importance of finding swap possibilities to shorten the net.
- A value of *50* is the neutral weight, equivalent to the net having no WEIGHT property attached.
- Each value greater than 50 progressively increases the importance of seeking swap possibilities to shorten the net.

How Automatic Placement Parameters Affect Swapping

Weights applied to the automatic placement parameters affect decisions made during automatic swap. Information in this section helps you determine whether to change the weight values in the Automatic Placement dialog box (accessed by choosing *Place – Autoplace – Parameters* or by using the place param command).

The following summary describes the rules that apply when using weights:

- A weight of *0* disables the parameter.
- A weight between 1 and 50 means that the parameter is considered only after considering weightier parameters.
- A weight of *50* is neutral and has the same effect as not applying the parameter.
- A weight between 51 and 100 indicates that this parameter is a preferred action.

North, East, South, and West Weights

Weights applied to the four direction parameters indicate a preference for keeping connections either vertical or horizontal.

Assign a weight greater than 50 for either north or south directions to keep connections vertical. Assign a weight greater than 50 for east or west directions (or both) to keep connections horizontal.

Straight Weight

Assign a straightness weight below 50 to keep connections diagonal. Assign a weight above 50 to keep connections straight.

Mirror Weight

If pin and function swapping is allowed between components placed on opposite sides of the design, assign a nonzero mirror weight. A mirror weight of zero effectively disables swaps between the top and bottom of the design. The default mirror weight is *50*.

Defining the Swap Area

You can specify any of the following areas for automatic swapping with the appropriate command. The menu items and commands are described in the *Allegro PCB and Package Physical Layout Command Reference*.

Area	Menu Item/Command	Notes
Design	Choose or run: *Place – Autoswap – Design* (swap area design command)	The layout editor automatically selects the area within the package keepin boundary as the swapping area.
Room(s)	Choose or run: *Place – Autoswap – Room* (swap area room command)	You specify one or more rooms as swapping areas.

Window(s)	Choose or run: *Place – Autoswap – Window* (swap area window command)	You can select up to 16 rectangular window areas for swapping. Choosing windows as a swap area gives you a more tightly defined area. You can define an area very tightly, but the layout editor views all the windows as one when looking at swapping possibilities.

⚠ To view the current, active area of the design for automatic swapping, choose *Place – Autoswap – List* (*swap area list* command).

Setting the Swap Parameters and Running Automatic Swap

⚠ Be sure to follow the tasks listed in Prerequisites.

You set the swap parameters in the Automatic Swap dialog box. The dialog box lets you define parameters for 10 swapping passes. For each pass, you can set the time limit and indicate whether interroom swaps are permitted. The layout editor completes each swap pass by running the function swap first, then the pin swap.

To set the automatic swap parameters and run automatic swapping, choose *Place – Autoswap – Parameters* (swap param command), described in the *Allegro PCB and Package Physical Layout Command Reference*.

⚠ The swap execute command also runs automatic swapping, based on parameters in the Automatic Swap dialog box and the area defined in the swap area commands (*Place – Autoswap – Design* swap area design, *Place – Autoswap – List* swap area list, *Place – Autoswap – Room* swap area room, and *Place – Autoswap – Window* swap area window).

Reviewing Automatic Swapping Results

Every time you run automatic swap, the program writes a log file called `swap.log`. The file records what function pairs were swapped during each pass. The example in Figure 9-5 shows a portion of a `swap.log` file. The remainder of this section explains the contents of the file.

The example log file begins with a summary statement of the date, function, and execution:

```
Starting swapping.   Tue May 19 14:33:24 2004

Swapping components, functions, and pins of drawing simple.brd
```

```
Starting function swap.  Execution 1
```

Next, the log reports parameter settings and other messages for that pass:

```
Interroom swapping will not be allowed.

Time limit is 60 minutes.

1 component(s) are unplaced and won't be swapped.
```

Next, the file records all swaps performed during the pass:

```
Swapping functions F4 (U3) and F1 (U1)

Swapping functions F6 (U3) and F1 (U3)

Finally, the file records statistics for virtual wire length, weighted wire length,
and reduction:

Virtual wire length  = 200   Reduction = 100 (33.3 %)

Weighted wire length = 200   Reduction = 100 (33.3 %)
```

`Virtual wire length` is the total distance between all function pins without the influence of weights or all given neutral weight. The values report the length after completion of swapping for this pass.

`Weighted wire length` is the total distance between all function pins, with the lengths of each net increased or decreased by the following properties and parameters:

- WEIGHT property attached to the net
- COMPONENT_WEIGHT property attached to the component
- Mirror parameter
- Straight parameter
- Direction (North, East, South, and West) parameters

The values report the length after completion of swapping for this pass.

Reduction is the amount the reported wire length was reduced during the pass. The layout editor reports the actual amount, followed by the percentage change in parentheses.

Finally, the program logs the time at the completion of the pass:

```
Finished with function swap. Tue May 19 14:33:27
```

Example Swap Log File

```
Starting swapping. Tue May 19 14:33:24 20042004
Swapping components, functions, and pins of drawing simple.brd
Starting function swap. Execution 1 Tue May 19 14:33:24 2004
Interroom swapping will not be allowed.
Time limit is 60 minutes.

1 component(s) are unplaced and won't be swapped.

Swapping functions F4 (U3) and F1 (U1)
Swapping functions F6 (U3) and F1 (U3)

Starting virtual wire length  = 2000  Starting weighted wire length = 2000

Virtual wire length = 300    Reduction = 1700 (85.0 %)
Weighted wire length = 300   Reduction = 1700 (85.0 %)

Finished with function swap.   Tue Nov 21 14:33:26 1996

Starting pin swap. Execution 1 Tue Nov 21 14:33:26 1996
Swapping pins 1 (A) and 2 (B) of U3 (F3)

Reduction statistics for this pass.
Starting virtual wire length  = 300  Starting weighted wire length = 300

Virtual wire length = 200    Reduction = 100 (33.3 %)
Weighted wire length = 200   Reduction = 100 (33.3 %)

Reduction statistics for this pass.
Starting virtual wire length  = 200  Starting weighted wire length = 200

Virtual wire length = 200    Reduction = 0 (0.0 %)
Weighted wire length = 200   Reduction = 0 (0.0 %)

Reduction statistics for all passes.
Starting virtual wire length = 300  Starting weighted wire length = 300

Virtual wire length = 200    Reduction = 100 (33.3 %)
Weighted wire length = 200   Reduction = 100 (33.3 %)

Finished with pin swap.   Tue May 19 14:33:27 2004

Starting function swap. Execution 2 Tue May 19 14:33:27 20042004
Allowing interroom swapping.
Time limit is 60 minutes.
```

Reviewing Pin, Function, or Component Data

You can generate any of the following by choosing *Tools – Reports* (reportscommand):

- Component pin report
- Function report
- Component report

Related Topics

- report
- Keepin and Keepout Areas
- Placing Elements Manually
- Placing Elements Automatically

10

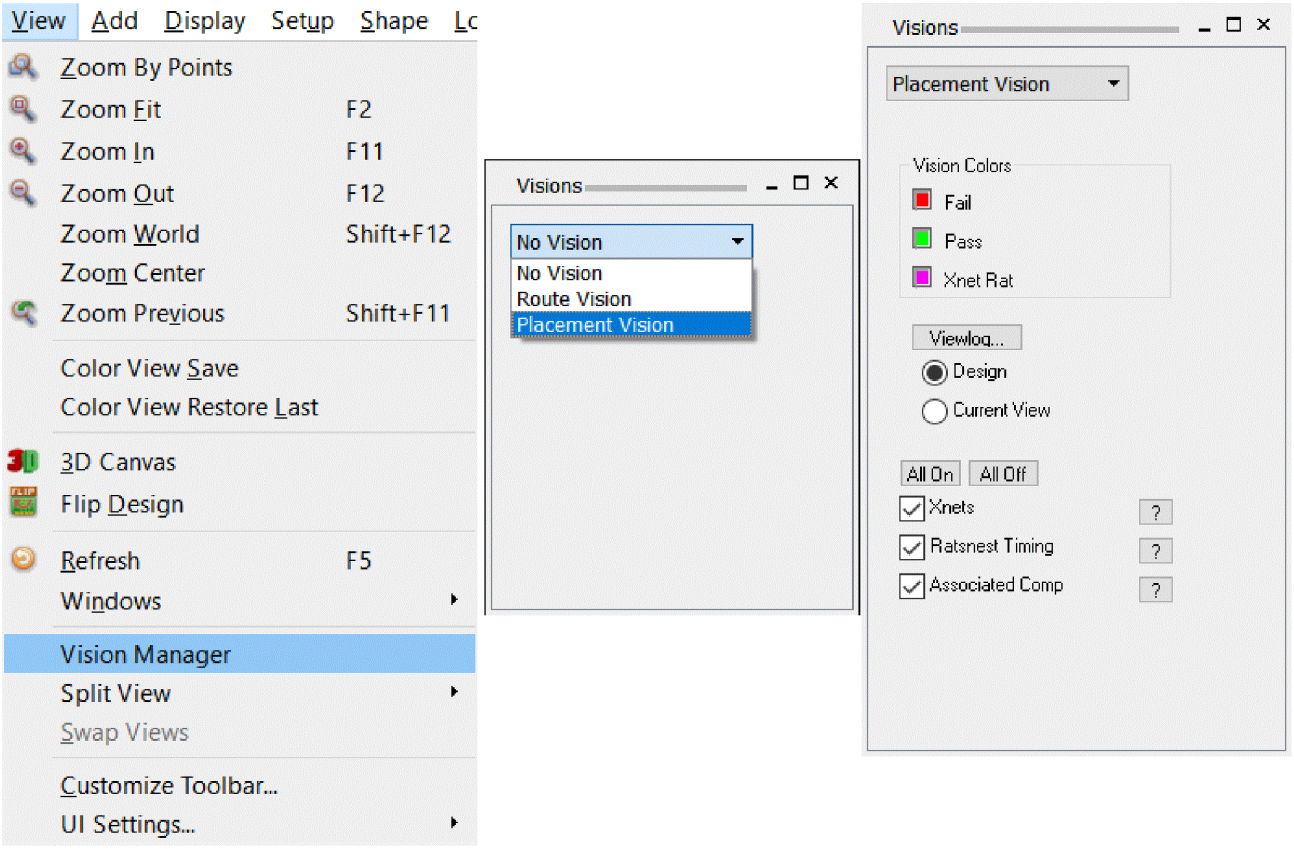
Reviewing Placement Status and Results

11

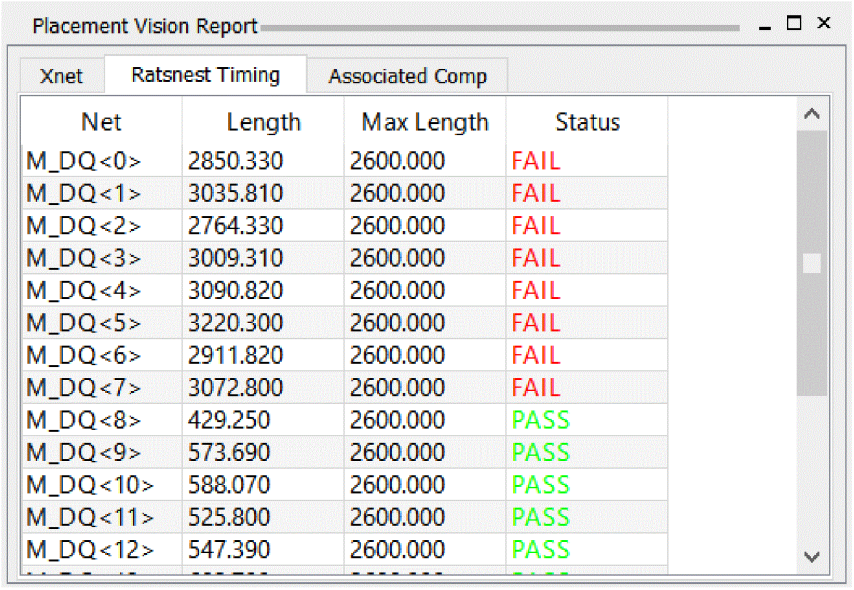
Optimizing Placement Using Placement Vision

Post-routing DRC checks often require changes in component placement and increases time to complete the design. A way to improve the placement is to provide various checks during the component placement design phase. These checks does not belong to the set of traditional DRCs, analyze the entire design and highlight the components and ratsnests violating the checks with user-defined colors on the design canvas. Adjusting component placement resolves failures and restores the default color of the objects.

Placement Vision can be used to enable these checks. To access placement vision, in PCB Editor choose, *View – Vision Manager* from the top menu. A new *Visions* tab opens. Select *Placement Vision* form the pull-down list. The user interface of *Placement Vision* includes three types of visions and settings to view the results.



A *Placement Vision Report* is generated showing the status of all the objects in three visions. An object failing the vision can be selected from the report window.

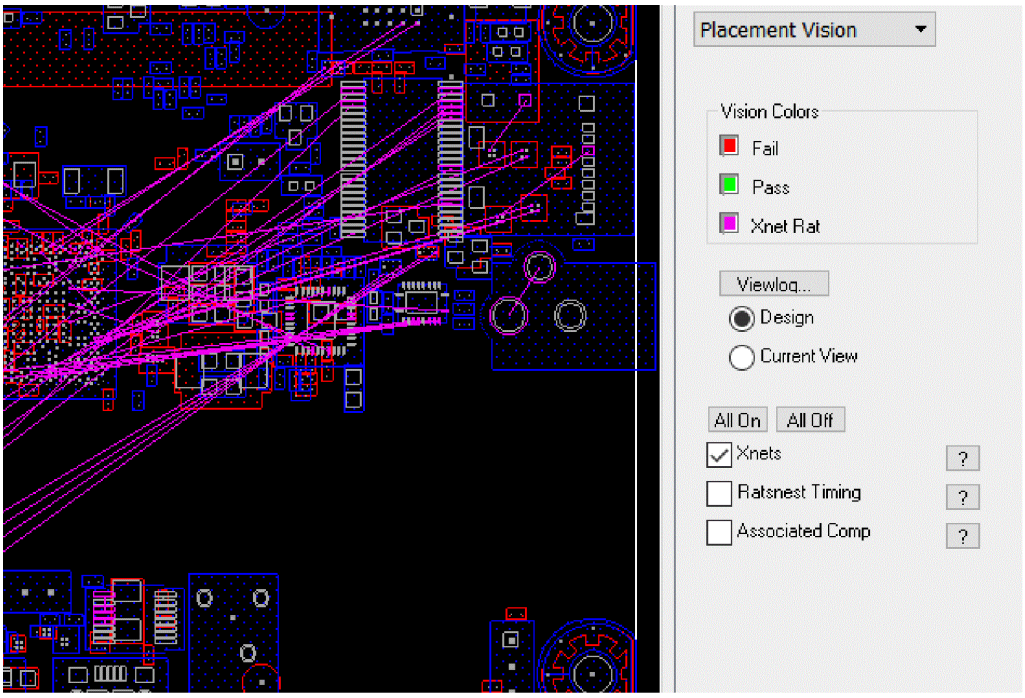


Net	Length	Max Length	Status
M_DQ<0>	2850.330	2600.000	FAIL
M_DQ<1>	3035.810	2600.000	FAIL
M_DQ<2>	2764.330	2600.000	FAIL
M_DQ<3>	3009.310	2600.000	FAIL
M_DQ<4>	3090.820	2600.000	FAIL
M_DQ<5>	3220.300	2600.000	FAIL
M_DQ<6>	2911.820	2600.000	FAIL
M_DQ<7>	3072.800	2600.000	FAIL
M_DQ<8>	429.250	2600.000	PASS
M_DQ<9>	573.690	2600.000	PASS
M_DQ<10>	588.070	2600.000	PASS
M_DQ<11>	525.800	2600.000	PASS
M_DQ<12>	547.390	2600.000	PASS

XNet Vision

Enabling this vision, starts displaying all the XNet rats of the design in the color specified in the *Vision Colors* section. The XNet rats are displayed even if the discrete components are not placed between the driver and receiver pins of the nets.

The purpose of this vision is to reduce the rats clutter so that the designer can focus on the placement of active components.

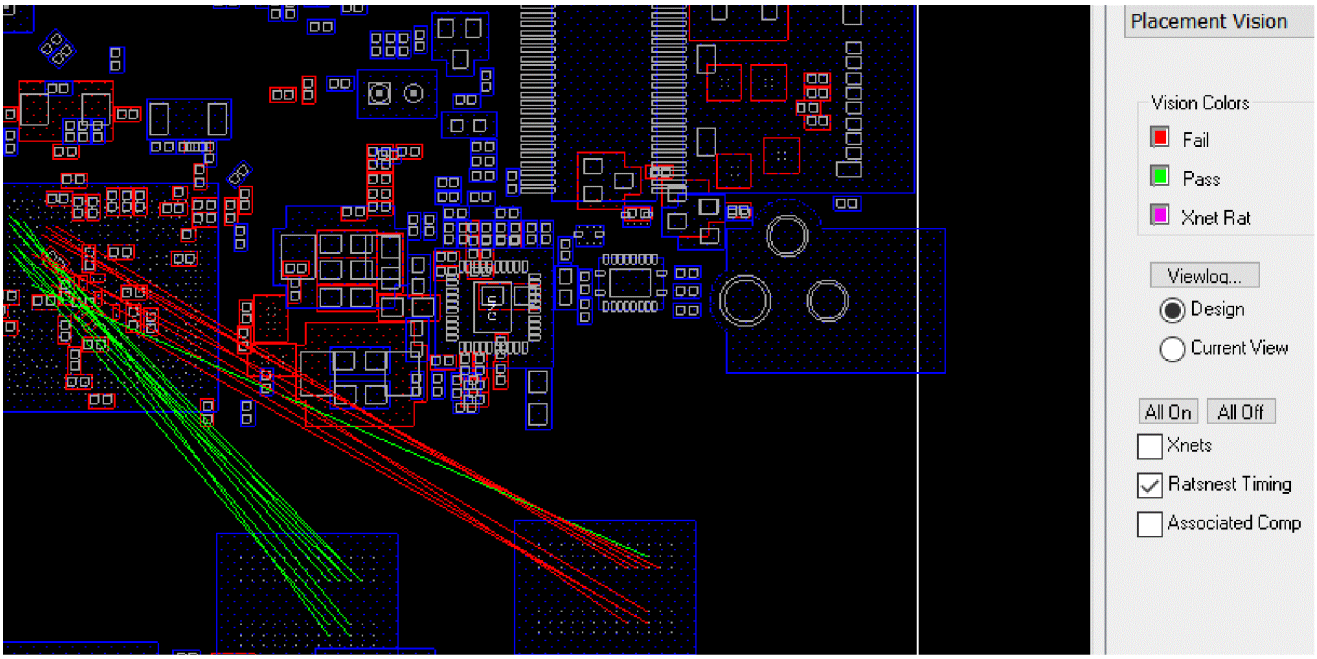


Ratsnest Timing

The *Ratsnest Timing* vision analyzes the rats for which timing constraints are defined and highlights the ratsnest in the colors specified in the *Vision Colors* section.

To view vision results, ensure that the *Propagation delay, Relative propagation delay,* and *Total etch length* checks are enabled in the Electrical section of the *Analysis Modes* dialog (*Setup – Constraints – Modes*).

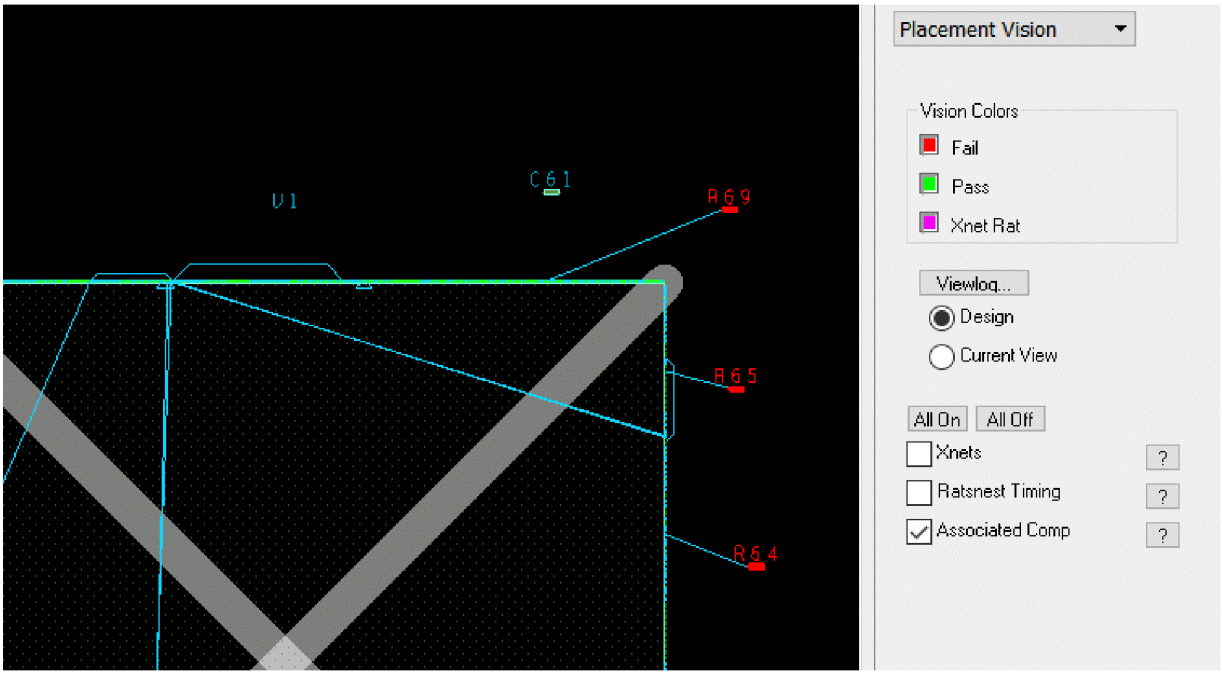
The purpose of this vision is to solve timing delay issues while doing component placement.



⚠ If *Xnets* vision is also enabled, only XNets rats are displayed and ratsnests timing vision is not shown.

Associated Comp Vision

The *Associated Comp* vision checks the spacing between associated discrete components and their parent components and highlights the associated component in the colors specified in the *Vision Colors* section. The purpose of this vision is to ensure correct the placement of associated components.



⚠ The *Associated Comp* vision is not available with OrCAD PCB Editor.

For more information on placement vision, see vision manager

12

Working with Groups and Modules

This chapter describes the concepts of groups and modules in and how to work with them. Groups and modules allow you to choose arbitrary database objects and create relationships among these discrete objects in a named design database, which you can then easily manipulate as a single unit (an association). The named database is referred to as a permanent group because it is saved with the database and can be referenced as a single object. Permanent groups let you reuse portions of designs by extracting them to new or existing designs—in effect, using them as building blocks upon which you can construct more complex objects. The creation of these units is sometimes known as "design reuse" because units can be saved and reused in other designs.

Although a natural correlation exists between the concepts of groups and modules, the layout editor treats these associations somewhat differently. In this chapter, therefore, separate sections cover groups and modules.

Related Topics

- Working with Groups
- PCB Editor: Working with Modules

Working with Groups

Groups can be referred to as permanent groups or persistent groups and should not be confused with temporary groups, an option available in the editor's pop-up menu (*Temp Group*) that allows you to temporarily group non-contiguous selections of design elements for application in an interactive command.

When an object is a member of a group, it should have a unique reference in the group. You cannot have a group of a symbol and a pin if the pin is part of the symbol. This would mean the pin has two references in the group.

Group Members

You create a permanent group by choosing objects to associate through a single parent object (the group). Objects you can include in a group are the following:

- Clines
- Components
- Cpoints
- Filled rectangles (frectangles)
- Groups (nested within other groups)
- Lines
- Nets
- Pins
- Rectangles
- Shapes
- Symbols
- Text
- Vias

Creating, Editing, and Disbanding Groups

You can work with groups in either the menu-driven editing mode, in which you choose a command, then the design element, or the pre-selection use model, in which you choose a design element (noun), and then a command (verb) from the right-mouse-button pop-up menu.

- In the menu-driven editing mode, choose *Edit – Groups* (`groupedit` command) to create, edit, and disband (remove objects from) a group in a design.
- In the pre-selection use model, access the `group add` and `disband group` commands from the right-mouse-button popup menu after you preselect group-supported physical elements. These commands are available from the right-mouse-button menu only in the placement and general edit application modes, and allow you to create new groups or dissolve existing ones.

Properties in Groups

The layout editor does not support properties which are applied directly to a group. Therefore, when you choose a group in the `property edit` command, you are actually choosing the members that make up the group. The layout editor applies the properties to the group members.

A group's behavior can be controlled with the use of properties. The FIXED property prevents the manipulation of any object in the group or the group itself.

Properties that you attach to objects at the group level remain attached to the individual objects after you disband the group. You may find it convenient to delete properties at the group level before disbanding the group, especially if it contains a large number of objects that otherwise have to be edited individually.

Importing and Exporting Groups

You can import and export groups by choosing *File – Export – Sub-Drawing (*`clpcopy` command) and *File – Import – Sub-Drawing* (`clppaste` command), respectively.

⚠ Groups that you import into a drawing do not retain their group status nor any properties applied at the group level. Additionally, reference designators no longer apply to the symbols, and all etch/conductor is attached to a dummy net.

Connecting Groups

Objects in a group may need to be routed to objects outside the group or to other groups in the design. In such instances when connect lines are owned by different groups or when they have properties intended to preserve the behavior of the cline (such as FIXED), the layout editor creates a connect point (cpoint) to maintain connectivity between clines while preventing them from merging.

Selecting and Finding Groups

The distinction between a group and the objects in it has implications when you attempt to choose or find groups. Some commands allow you to choose groups, such as *Edit – Groups* (`groupedit` command) or *Display – Element* (`show element` command). Other commands, such as *Edit – Move* (`move` command), do not allow you to choose the group but use the group item to choose objects that are group members.This behavior can be seen when you attach a property with groups enabled in the Find filter.

Group Member Selection Methodology

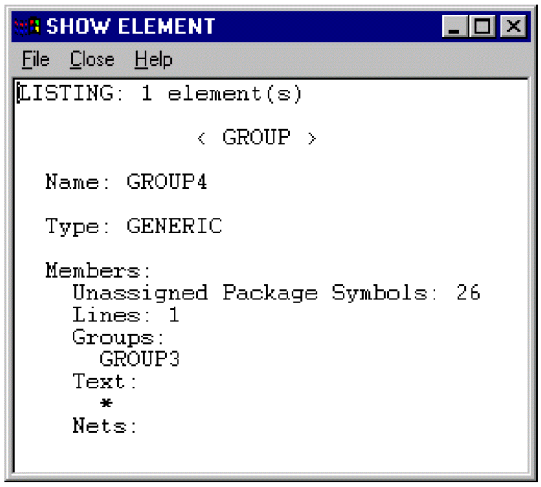
Keep the following three factors in mind for group member selection:

- You cannot choose group members with no visible class/subclass.
- If you cannot choose an object type, as seen in the Find filter, you cannot choose that object by the group member selection.
- Group members are chosen even when they are outside the viewing area.

Example of Group Selection

If you choose *Display – Element* (`show element` command) with *Groups* selected in the Find filter, the resulting Show Element window references the group as the element, not the objects within it, although they are listed as members.

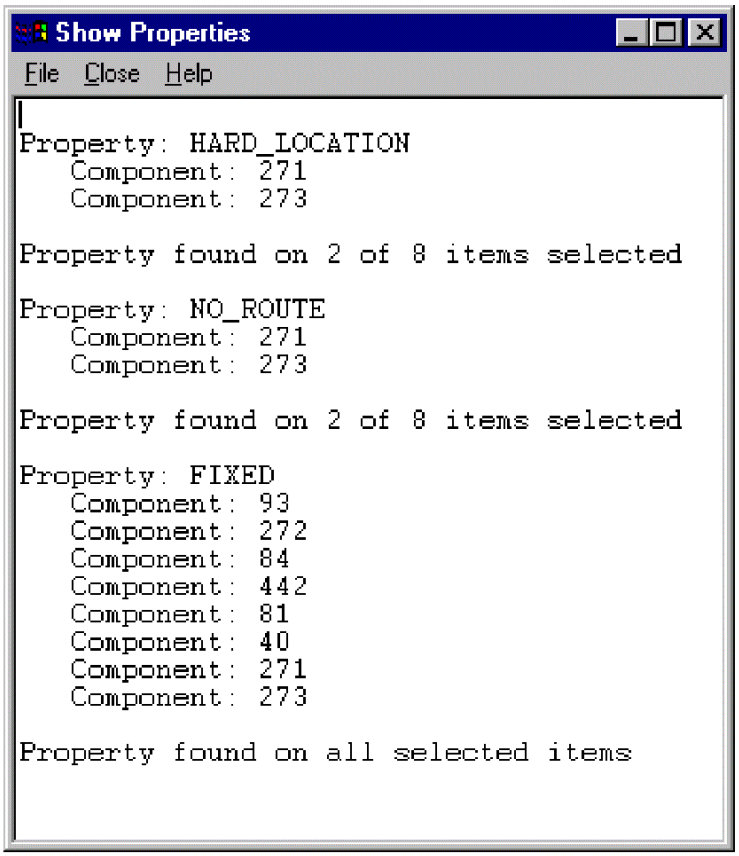
Show Element Window



Example of Group Member Selection

If you choose *Edit – Properties* (`property edit` command) with *Groups* selected in the Find filter, the Show Properties window (accessed by way of the *Show* button in the Edit Property dialog box) references the individual group members, as shown in the following figure.

Show Properties Window



You can choose groups by name, list, or window, or by picking them with the mouse. If you want to run a command on groups that are nested inside other groups or on components that are members of more than one group, you focus e selection using the *Reject* command in the pop-up menu. The reject function operates when you choose by mouse pick.

A chosen group (or object within a group) does not display a *Reject Item Selection* dialog box unless it is a member of more than two groups. The highlighted item is the one that is affected by the editing command that you run on the group (that is, the objects in the group).

Resolving Object Selections in Multiple Groups

Because objects can be members of multiple groups, you may need to resolve the group you want to choose. In the example below, you want to delete objects at the group level. However, the objects that you choose by mouse pick are members of multiple groups, so you may need to use *Reject* to choose which of the groups you want the command to act on.

Related Topics

- groupedit
- group add
- disband group
- Working with Groups
- clpcopy
- clppaste
- groupedit
- show element
- move
- property edit
- delete

PCB Editor: Working with Modules

In layout editors, modules are collections of physical entities that can include other modules. Modules may or may not have logic (represented by nets, components, and so on.) or a block (a collection of schematic information a schematic tool uses) associated with them. They can be permanently stored as module definition databases (*modulename*.`mdd`) in library files using the module path (modulepath) environment variable.

The design methodology you use to build a design largely determines how you create and use modules. Two basic methods exist:

- Schematic-driven
  You use your schematic capture system to determine the logic used across blocks and modules. Reference designators, net names, and so are resolved on the front end; the module definition in Allegro X PCB Editor supplies no logic information.

  ⚠ Module functionality is not supported with SCALD-based tools.

- Module-driven
  The module definition in Allegro X PCB Editor contains logic as well as physical data. Logic defined in the definition is transformed through command functionality (`dlib`, `refresh symbol`, and so on) so you can place and reuse it compatibly in a design with existing logic.

You can generate the Module report that lists all placed modules, sorted by module instance and module definition. To generate a Module report, choose *Tools – Reports* (reports command) or run the report batch command, both described in the *Allegro PCB and Package Physical Layout Command Reference*.

13

Partitioning a Design

To support multi-designer collaboration on a single design, a master designer, responsible for the master design (`.brd` or `.mcm`):

- Partitions the master design into sections or layers.
- Assigns these *partitions* to partition designers.
- Exports the partitions to independent databases usingWorkflow Manager.

⚠ Design partitioning functionality is available with Team Design option in Allegro PCB Designer and Allegro Packager Designer L.

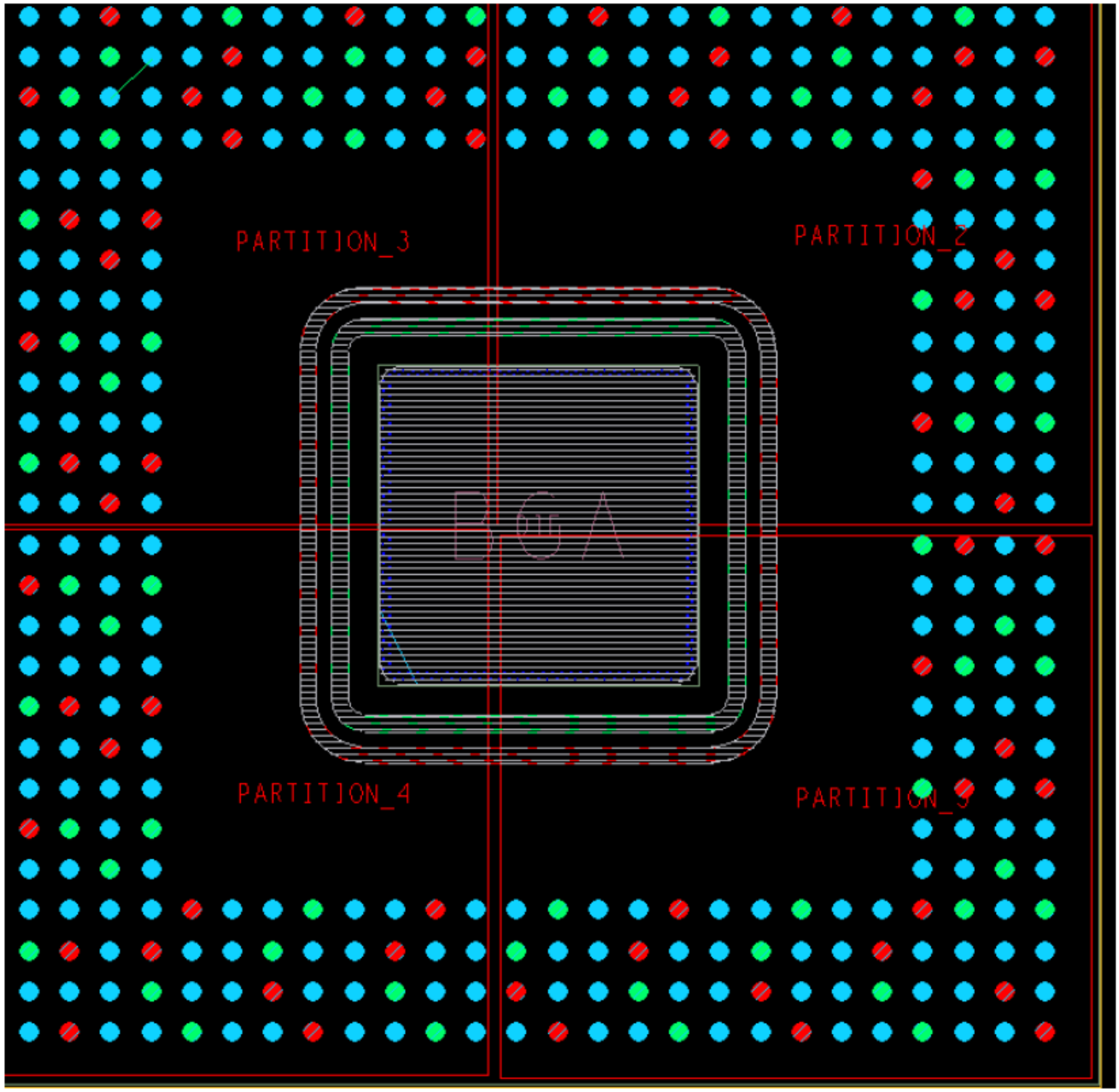
The *master designer* is the only designer allowed to create partitions, and import and export them to partition designers. A *partition designer*, subordinate to the master designer, is responsible for completing the assigned partition. The *Workflow Manager* allows the master designer to manage all partitions of the primary (master) design after they are created with the *Place – Design Partition – Create Partitions* (partition) menu command. For each partition created, an entry appears in the Workflow Manager. An extension of `.dpf`, `dps`, or `.dpm` is appended to partition files.

Partition designers performs placement, routing, constraint editing, create silkscreens on their respective partitions and refresh information from the master database or from other partitions through the Workflow Manager for the most current view of the design.

The master designer can import partitions into the master database, refresh the master database to include changes in the partitions, or retract (cancel) exported partitions back into the master database at any time. The master designer initiates all updates and controls all changes to the master database. No real-time updates occur to any database.

For more information see, Best Practices:Working with Design Partitions.

APD: Partitioned Design



APD: How Design Partitioning Works

When using APD, the master designer should wire bond dies prior to defining design partitions.

In the master design:

- If all the wire bonds belonging to a die lie entirely within an unexported partition in the master design, wire bond modify operations such as editing and deleting are allowed in the master design.
- If any part of the wire bond pattern, for example, a bond finger, lies within an exported partition, no wire bond modifications are allowed in the master design. A warning message indicates that one or more fingers are FIXED.
- Adding a wire bond is always allowed. Wire bonds can be added in the master design even if the partition boundary runs through the die and the partitions have been exported. This is consistent with Allegro X PCB Editor operation because adding of route connects and vias to the master design even within the boundary of an existing partition is allowed.

In the partition designs:

- Wire bonds are treated as fixed objects. Therefore the partition designer is unable to edit (add, delete or modify) these objects.
- To make changes to wire bonds, use one of these options:
  - Update all partitions to the master design and close all partitions. Make necessary edits to wire bonds in the master design, and then re-establish the partitions again
  - Close all partitions without synchronizing with the master design. Make necessary edits to wire bonds in the master design, and then re-establish the partitions again.

Partitioning a Design

Partitioning has no impact on a design until the master designer exports partitions. Cadence recommends that prior to exporting the partition:

- Load all logic
- Place or assign symbols to the partition

- Set up all constraints
- Adds guideports
- Assigns soft nets.
- Assess the required disk space by multiplying the master design's size by the number of planned partitions

Partition Databases

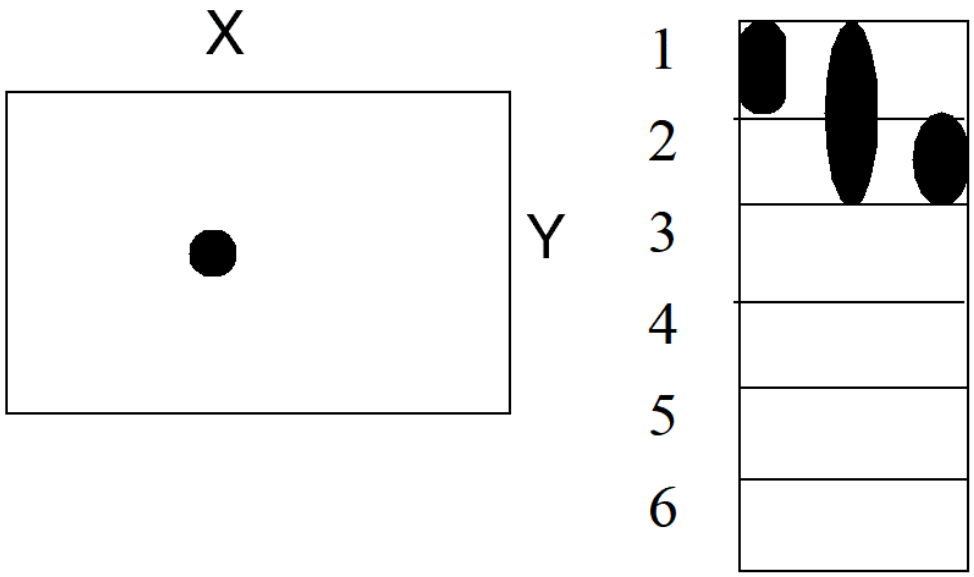
A *partition database* is a copy of the master database from which it was exported, and appended with the extension `.dpf`, `.dps`, or `.dpm`. Once you open the partition databases, a limited command set becomes available.

The `uprev` command does not support partition databases. The master designer must import all partitions prior to upreving the master design. You cannot archive partitions as part of the design history as they become obsolete once imported back into the master design; however, you can back up partition files. Most batch programs do not run on `.dpf` `.dps`, or `.dpm` files.

Partition Membership

Partition membership is determined at the time of export because the designer can add objects to the master design until it is exported.

This Object May Be a Partition Member...	Under These Conditions...
Module Instance	Is always a member.
Symbol Instance	When the symbol instance footprint lies within the partition boundary, including all pins, vias, and etch/conductor objects. If the partition includes the symbol, potentially some objects become members despite their being outside the partition.
Component Instance	Is always a member.
Diestacks	If the footprint of a die is within a partition, it is a member of the partition.
Net	Its pins, vias, clines, and shapes lie within the partition boundary.

Pin	Its connect point must be within the XY boundaries and its padstack definition must be with the layers of the partition, or its parent symbol instance must be a partition member. If a pin is a member, you cannot alter it physically, although you can swap it with other partition pins. For example, the following illustration shows a two-layer partition in a six-layer design. At the left, the connect point must be within the XY boundaries, and at the right, the padstack definition can be on layer 1, layer 2, or on layers 1 and 2. 
Via	Its connect point lies within the partition. A connect point on the boundary excludes the via from partition membership.
Bond Finger	If a bond finger's connect point is within a partition, it is a member of the partition. If the bond finger's connect point is on the boundary, the assumption is that it enables connections between partitions. The bond finger is not a member of the partition.
Shape	Its outline can lie within the partition and touch the partition boundary but does not intersect it.
Frectangle/Rectangle	It lies within the partition and touches the partition boundary but does not intersect it.
Cline	Its end points lie within the partition, but the center line cannot intersect the partition boundary.

Line	Its endpoints lie within the partition boundary, but its width may extend beyond it. The center line cannot intersect the partition boundary.
Text	When a rectangle bounds stand-alone text (not the extent box) but does not intersect the partition boundary, although it can be inside and touch it. If the text is a child of another object such as a symbol, the parent object's eligibility dictates partition membership. Consequently, text may appear as a partition member despite its lying outside the partition.
Rat_T	Its origin lies within a partition and can be relocated only if its parent net is a member.

Editing Within a Partition

Only objects meeting these class/subclass rules are eligible for editing within a partition. All master design data is in a partition database, but objects on the following list of layers are unlocked and therefore modifiable.

In addition, during partitioning, a master designer can attach the LOCKED property to symbols and modules to prevent partition designers from modifying the symbol or module contents (except for text editing). The partition designer is not allowed to remove the LOCKED property.

During import to the master design, only objects that are unlocked in the partition update to the master database.

⚠ The rats outside partition and unrats outside partition commands are available during partitioning. To dim the read-only objects in a by-layer partition (so that you can easily recognize the objects that you can edit), set the `display_readonly_intensity` environment variable in the User Preferences by choosing *Setup – User Preferences – Display – Visual* (enved).

Board Geometry	SILKSCREEN_TOP: No Auto-silkscreen objects SILKSCREEN_BOTTOM: No Auto-silkscreen objects. CONSTRAINT_REGION Any User-defined subclasses

Manufacturing	MFG_PROBE_TEXT: Text associated with test points. MFG_PROBE_TOP: Text associated with test points. MFG_PROBE_BOTTOM: Text associated with test points. MFG_NO_PROBE_TOP MFG_NO_PROBE_BOTTOM
Package Keepout	All subclasses examined
Etch/Conductor	All subclasses evaluated.
Route Keepout	All subclasses evaluated.
Via Keepout	All subclasses evaluated.
Via Class	All subclasses evaluated.
Anti Etch	All subclasses evaluated.
Boundary	All subclasses evaluated.
Package Geometry	Objects unrelated to symbols only

APD: Commands Allowed in the Design Partition Editor

The following table shows the ICP-specific commands that behave the same in the master and partition editors.

`add via structure`	`degas`	`offset via gen`	`view 3d`
`assemrules standard`	`derive assignment`	`pbar check`	`wire bond escape`
`bond finger dehilite`	`die escape gen`	`pop swap pin assignment`	`wire bond via estimatation`
`bond finger hilite`	`dpn`	`radial router`	
`bpa`	`etchback`	`rats layer`	

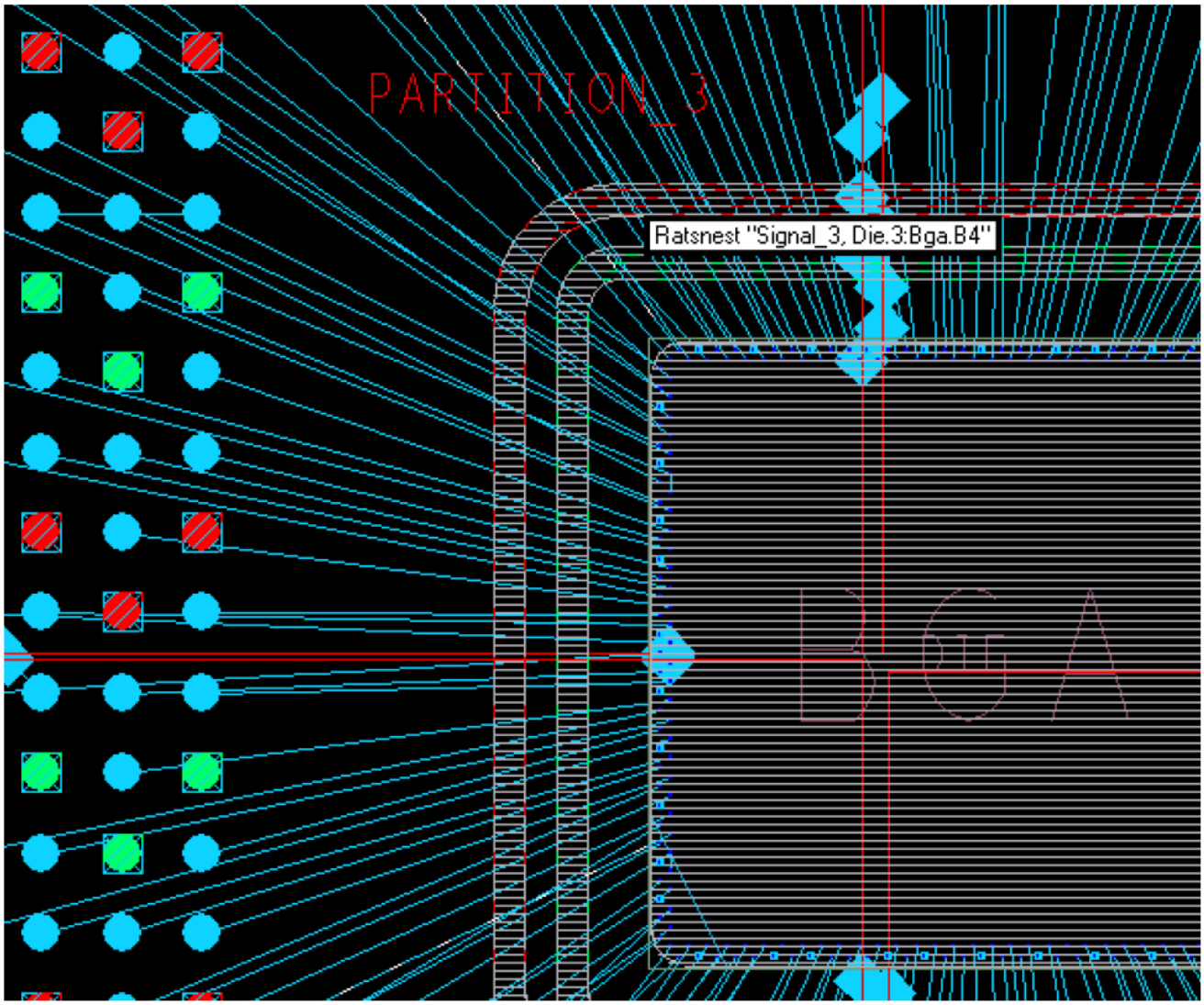
`compose line`	`keepout component`	`spread clines`	

The `die properties` command is read-only in the partition editor.

Using Guideports

During design partitioning, you can use *guideports* (guideport command), which are optional visual checkpoints that suggest potential connections for unrouted nets that cross partition boundaries, as you define an interconnect strategy. A *partition boundary* is a closed polygon that defines the design section assigned to the partition designer. The polygon cannot overlap or lie within another partition boundary, or contain voids or arc segments in the outline.

APD: Guideports



Guideports are unavailable during partition creation. Once a partition is created, you can fine-tune, move, and reconfigure guideport locations based on the *Spacing Criteria* parameters in the *Options* tab.

Only connections with one pin inside the partition and a target connection outside the partition receive a guideport, excluding pass-through connections; consequently, a guideport functions much like a Rat T in that it visually breaks a ratsnest line where it crosses the partition boundary, assisting the partition designer to run the trace.

Spacing and line width constraints locate guideports around the partition boundary. Guideports appear for every from-to based on the default grid in the same color as ratsnest lines. Multiple

guideports may exist on a single ratsnest line if it passes through a partition or enters multiple partitions where the edges are not coincident. Guideports contain no layer information.

When the master designer imports partitions back into the master design database, guideports remain in place unless you delete them, to restore scheduling.

Guideports – show element Command

When guideports exist on a net or on one associated with a chosen object, and you choose *Display – Element* (show element command) in a design partition (`.dpf` `.dps`, or `.dpm`) file, previously unscheduled nets appear in the text display dialog box as guideport-scheduled nets, as shown below. Nets you wholly or partly scheduled before creating guideports appear as user-scheduled, guideport scheduled nets, and net schedule appears as locked.

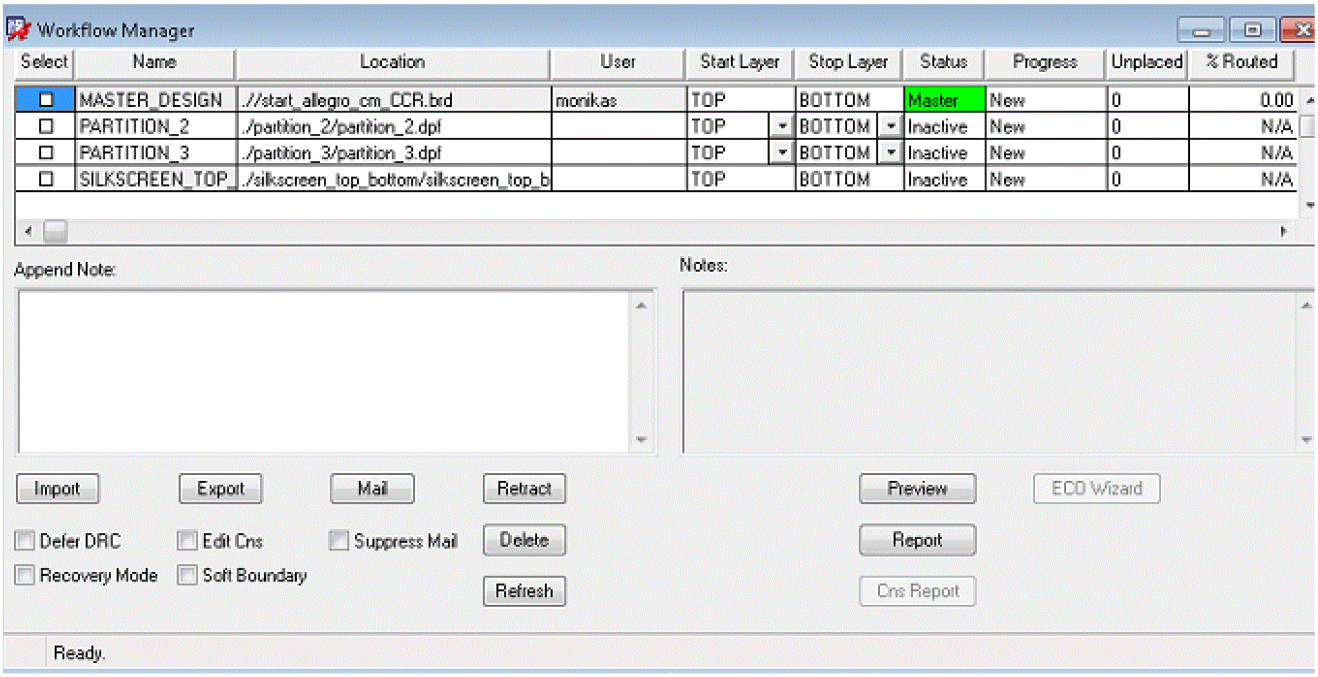
```
LISTING: 1 element(s)

<NET>

Net Name:     VCLKA
* user scheduled net*
* guideport scheduled net * schedule is locked*
U5: 34 U18.11 U8.11 T.1 T.2 U21.11

Via Count:              2
Total Etch Length:      3180.5 MIL
Total Path Length:      3781.5 MIL
Total Manhattan Length: 3793 MIL
Percent Manhattan:      99.70%

Pin     Type  SigNoise Model
---     ----  --------------
U5.34    OUT  CDSDefaultOutput
U18.11    IN  CDSDefaultOutput
```

```
U8.11     IN  CDSDefaultOutput

VCLKA.T.1

VCLKA.T.2

U21.11    IN  CDSDefaultOutput


3 unrouted connection(s) remaining

VCLKA.T.2 to U21.11

VCLKA.T.1 to VCLKA.T.2

U8.11 to VCLKA.T.1

Electrical Constraints assigned to net

pin order type: all rats are user defined

Object is read only
```

Handling Partitions with the Workflow Manager

For each partition created, an entry appears in the Workflow Manager. From the Workflow Manager, the master designer:

- Exports partitions to partition designers.
- Imports partitions to the master database.
- Refreshes or updates the work done by the partition designer within an assigned partition to the master database.
- Defers DRC update when importing partitions into the master database.
- Deletes specified partitions.
- Retracts or cancels all exported partitions when partition contents change.
- Generates a report for a specified partition.
- Previews information for a partition prior to export.
- Chooses to send or suppress email.
- Allows an obsolete partition to be imported.



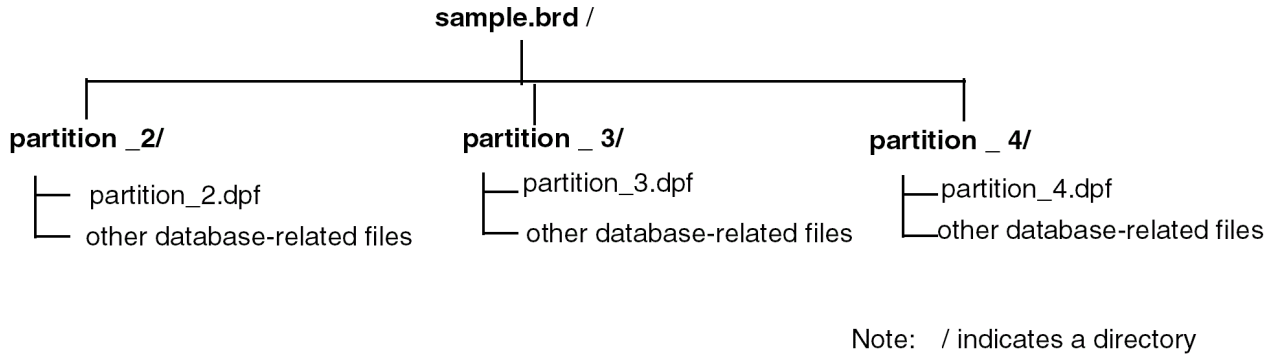
Exporting Partitions

The master designer assigns and exports each partition to each designer. Exporting partitions duplicates the database. The master designer may export as many partitions as required.

When a partition is exported, internal email automatically notifies partition designers to stop working and delete the partition database because it will no longer be allowed to import back to the master design.

Exporting a partition creates a directory with a name identical to that of the partition without the extension in the current design directory. If the partition designer uses that as the working directory, any secondary files created are private. Typically, the partition designer copies the partition database to a local machine to avoid working across the network.

PCB Editor: Partitioned Board



The above sample directory can be used as a central location to which to update designs. The master designer can then in turn download the latest versions. Browsing or obtaining the direct path to the partition database on the other designer's local machine are other options.

Refreshing Partitions

Prior to integrating the partition databases into the master database, the master designer can update the master database with partition designers' work to check, for example, whether the partition modifications function correctly in the master design and with those in other partitions. Partition designers can refresh information with the Workflow Manager from the master database or from other partitions for the most current view of the design.

Importing Partitions

Once partition work is complete, the master designer integrates the chosen partition databases into the master database using the *Import* function on the Workflow Manager. A partition database must be saved to disk and accessible either in the directory to which it was originally exported or via a path the partition designer specifies.

Ownership of the partition reverts to the master designer, rendering the exported partitions obsolete. The master designer notifies partition designers to stop working and delete their partition databases because they cannot be loaded into the master design once imported. If the partition designer needs to continue working, the master designer must export the partition again.

During partition imports, DRC runs on the imported objects to ensure correctness in the master design. The original objects that are partition members are either deleted or updated, depending the object type. The new partition objects load from the partition database.

Communicating with Designers

Internal email, available by running the mail command, facilitates communication among team members. Mail is automatically generated whenever partitions are exported.

Notes are another means of communication. The master designer can tailor instructions, guidelines, comments, and design information for each partition, which can be updated with the *Refresh* button. No interprocess communication occurs to pass notes real-time.

Assigning Soft Nets to Partitions

The master designer can designate certain nets as soft when partitioning the master design. Then after the partition is exported, the partition designer can pick and route these nets even if they extend beyond the boundary of the active assigned partition. Soft nets are highlighted in the owner's partition database but are dimmed and read-only in all other partitions.

For additional information, see the soft net command.

Working in a Partition Database

A partition designer may open a partition database and access a restricted set of commands and menus. All editable objects of the partition appear normally; everything outside the partition dims.

The partition designer adds, deletes, or modifies objects inside the partition boundary but is restricted from modifying units, accuracy, drawing size, origin, or the assignment table or any objects outside the partition. Information only commands can be executed outside the boundary. If the partition owns parent objects, related child objects outside the partition may be modified. The designer can also work with softnets outside the boundaries of the partition.

Modifications to grids, colors, and command options within the partition do not update into the master database design when the partition is imported. No constraints can be added or modified. Constraint settings can be viewed but not changed. Placement can be changed if the partition owns the entire symbol. Nets completely inside the partition can be re-scheduled. After partitioning, to edit the constraints in the master design, the partition database must be imported back into the master design, and after editing constraints, the partition exported again.

Actions Excluded after Partition Export

After exporting partitions, certain operations are unavailable to protect the work of the partition designers. These include:

- Modifying the following:
  - Accuracy or units
  - Cross-section
  - Drawing size and origin
  - Objects in the exported partition
  - Property definitions
- Editing or importing logic
- Refreshing symbols and padstacks
- Autoswapping
- Renaming reference designators
- Modifying design padstacks or replacing padstacks
- Deleting objects or scheduling nets on a net whose pins exist in the exported partition

  ⚠ You cannot do either of these tasks if you do not own the data for both the master and partition database. Once the master designer exports a partition, the master no longer owns partition data.

- Using guideports

Managing Files

An exclusively Windows or UNIX/Linux environment streamlines data flow for the design-partitioning team because similar environments support the same path and network identifiers. Windows translates UNIX/Linux paths using network-sharing software, but Windows paths may be problematic for UNIX/Linux UNIX/Linux. Consequently, it may be required to manually copy databases to certain locations to ensure accessibility.

When the master designer is on Windows, UNIX/Linux partition designers obtain databases from a repository on a UNIX/Linux machine from which the master designer can download and upload the databases. Network-sharing utilities map the UNIX/Linux system to permit access from Windows. If the situation is reversed, the master designer's UNIX/Linux system can be mapped using a network-sharing utility to let the Windows users navigate to the UNIX/Linux system as if it were Windows. The database is copied to its original location for the master designer to access it.

Supporting Constraint Manager

By default, Constraint Manager shows nets with at least one pin in the partition and includes nets related to those in match groups, buses, or differential pairs. All constraints are read-only. An analysis may be generated from the Constraint Manager. When in full view, nets excluded from the partition appear as stippled.

Supporting Testprep

Design partition supports manual test prep inside the partition only. Testprep parameters in the partition database cannot be changed. Conflicts can occur in the text attached to testpoints. Editable text strings store names, thereby hampering sequence verification. Newly created testpoints prefix the probe name with the partition name; for example, PARTITION_4_GND-1.

Supporting Allegro PCB Router

Design Package Router creates a temporary route keepin or fence that matches the partition boundary for exported partitions. The entire design is available to Allegro PCB Router.

14

Using Placement Edit Application Mode

The placement edit application mode provides an environment that lets you perform tasks relevant during placement, such as:

- Placing components
- Aligning components and modules
- Moving components incrementally with arrow keys
- Moving associated components
- Replicating circuits based on common connectivity and devices using a schematic-independent template model
- Propagating updates made to one place replicate module to all other instances with the same base name across the design
- Using a component's alternate symbols (right-mouse-button pop-up menu)

⚠ Not all tasks are available in APD.

Mode Acess and Verification

Activate placement edit application mode in the following ways:

- Choose the menu option *Setup – Application Mode – Placement Edit.*
- Enter the `placementedit` command in the Command Console window.
- Right click and choose *Application Mode – Placement Edit.*
- Click the appropriate toolbar icon (if added to your toolbar). 

You can quickly verify that placement edit application mode is active when "PLC" displays in the status bar.

Using the Placement List Foldable Window Tab to Place Components

Within the placement edit application mode, the Options foldable window tab displays a dockable version of the Placement List tab, typically accessed by choosing *Place – Manually* (place manual command), featuring the *Mirror* and *Place by Refdes* options. To access the Placement dialog box in its entirety, click *More Options*. The full Placement dialog box provides the capability to choose all remaining components in addition to a variety of selection filters to narrow display results.

Aligning Components and Modules

To maximize routing channels and printed-circuit board real estate, the placement edit application mode features the align components command, available from the right-mouse-button pop-up menu, that lets you fine-tune the alignment of already placed components along X, Y, or odd angle lines, using the following criteria:

- Components must exist on the same subclass.
- More than one component must be chosen.
- Components must not have the FIXED property assigned to them.

Offering similar functionality as the `align components` command, the align modules command is also available from a pop-up menu to align module instances (`.mdd`). You can also mirror and rotate these modules and then align them from the right-mouse-button pop-up menu.

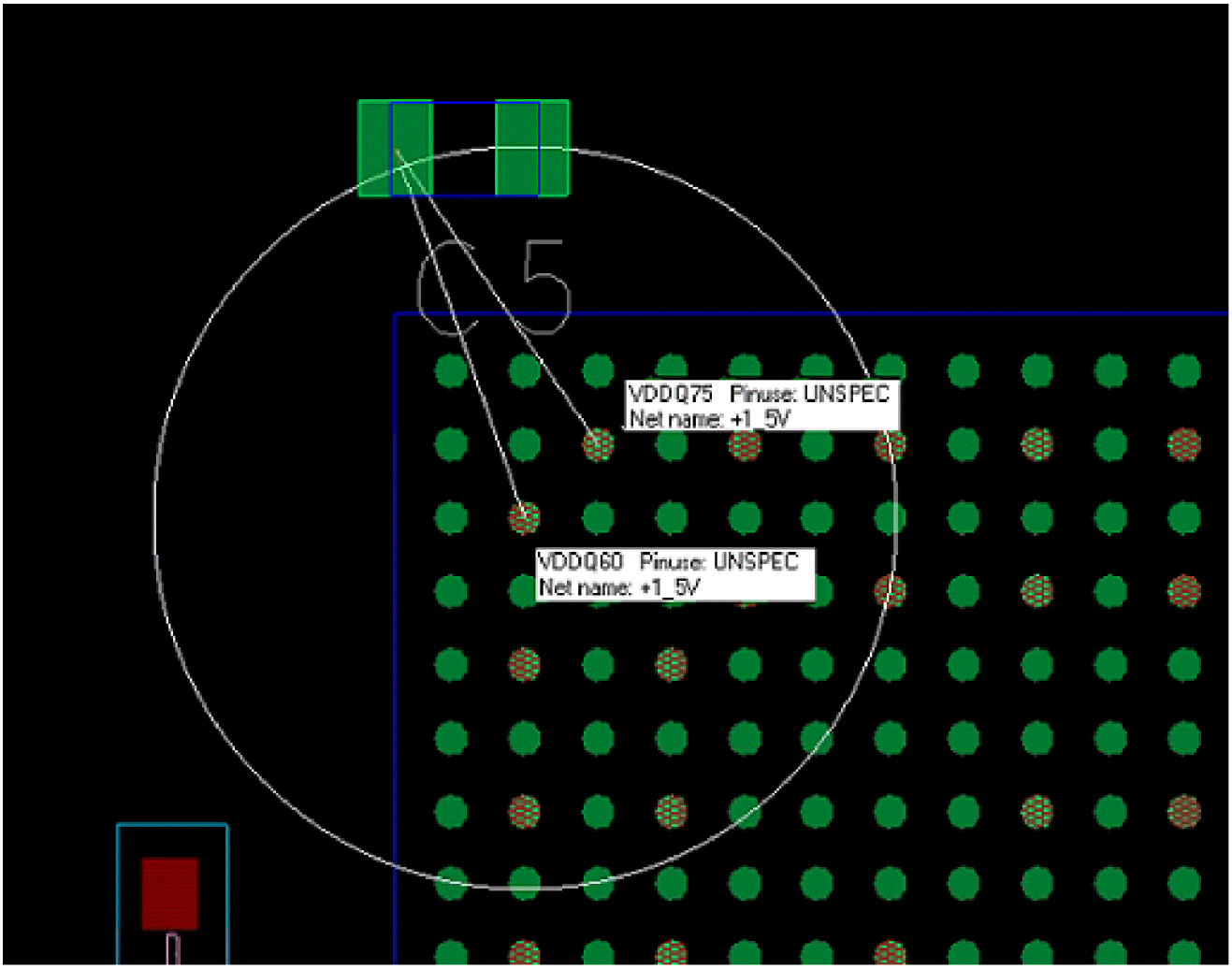
Moving Components Incrementally

It is often necessary to move or nudge a component a small distance, sometimes as little as one mil. It is quite difficult moving such small distances using a mouse. *Shift*-click to select and move a component incrementally if no interactive command is active. Use the *Shift* keys, plus the *Up, Down, Left*, and *Right* arrow keys—defined as default aliases in the system `env` file—to move selected elements in 1-grid increments in the desired direction. Ensure that a chosen element remains highlighted, at which point the *Shift* plus arrow keys can be used to move it incrementally.

Moving Associated Components

Associated components should be placed within a specified range from the parent components. During placement, this range is shown visually as a circle around the parent component pin. The radius of the circle is determined by the value of the directive `assoc_parent_pin_spacing`. This System Connectivity Manager directive is defined in the START_DESIGNSTUDIO section of the `cds.cpm` file. The directive associates certain properties to the parent and the associated components in the packaged files `pstxprt.dat` and `pstxnet.dat`. These properties are available on logical import.

On moving an associated component using the `move` command, the circle is visible to specify that it is an associated component.



PCB Editor: Circuit Replication Flow

For multi-channel or designs with repetitive circuitry, the placement edit application mode's template-copy model duplicates placement and interconnect data for common circuit blocks. This eliminates the need to create re-use modules in Allegro Design Entry HDL. For example, circuits include memory modules, IO channels, and capacitor schemes associated with BGAs or other active components. The place replicate commands are available in the pre-selection use model, in which you choose an element first, then right click and execute the commands.

⚠ The schematic-independent place replicate functionality assumes a netlist has already been read into the design.

The place replicate create command designates a circuit as a "seed," and creates a place replicate module definition database (`.mdd`) file from it, essentially a template containing pre-placed symbols you have chosen, their associated logic, routed etch, shapes, vias, text, and structures to be used to replicate additional circuits.

To support etch replication, the symbol extents of all selected symbols form a border that helps determine whether the etch becomes part of the template. A cline becomes part of the template if it:

- is marked as a pin escape and connected to one of the symbols in the selection set
- starts and ends on pins belonging to components of symbols in the selection set
- starts on a pin belonging to a component of a symbol in the selection set and ends on a via within the extents border
- starts on a pin belonging to a component of a symbol in the selection set and ends within the extents border

A shape becomes part of the template if all the pins within that shape belong to symbols that were selected to form the place replicate template.

The place replicate apply command is used to overlay the "seed" data onto additional pre-selected symbols. Circuit are replicated when their symbol and component definitions, and net connectivity match the criteria in the `.mdd`. The command matches symbols and alternate symbols if any symbol in the target replicated circuit:

- is the alternate of a symbol in the seed circuit, or
- has an alternate symbol in the seed circuit

One of the following dialog boxes may display:

- The *Place Replicate Component Swap Interface* dialog box displays a subset of circuits by reference designator in the current selection set that match the `.mdd` criteria, and which are

targeted for replication. This dialog box displays only when there are components in the selection set that can be swapped with components in the *Next Circuit* section, such as capacitors for instance. These components must share the same symbol and component definitions and net connectivity, and components are only considered for swapping if the integrity of the circuit is maintained. Once any substitutions are made, the component group shown in the dialog box appears on your cursor, one instance at a time. The tool then continues to locate circuits in the selection set that match the seed circuit, and displays the dialog box to allow you to substitute one component for another (unless you enable *Hide Form*).

- The *Place Replicate Unmatched Component Interface* dialog box only displays when a component in the seed circuit cannot be matched in the targeted replicated circuit, and assists in troubleshooting these situations.

After placement, these symbols comprise a group database object, named CR_<*user-assigned_name*>_<*1 -n*>, where CR refers to circuit replicate, and you specify the user-assigned_name if the`.mdd` is created from the current design.

The place replicate update command propagates modifications, such as to placement and routing, to all other instances of a replicated circuit with the same base name across the design.

Using Alternate Symbols

You can choose a particular alternate symbol to use in place of a component, globally or by selection if you previously attached the ALT_SYMBOLS property type to the components using the Cadence schematic-capture tools Allegro Design Entry HDL or CIS. ALT_SYMBOLS defines an alternate package symbol that can be substituted for the primary package symbol. Or, if you are using a third-party schematic, in the device file, assign the ALT_SYMBOLS property to components by specifying a PACKAGEPROP property record.

If any alternate symbols are defined for one or several selected symbol instances of the same type, when you right-click, the following pop-up menus display, each of which expands into the list of available symbols you can replace the original(s) with.

- **Selected Instances**, a list of alternate symbols displays. The chosen alternate symbol replaces the currently selected symbol instances. If the symbol definition for the alternate symbol cannot be found, the original symbol instance remain intact.
- **All instances,** the chosen alternate symbol replaces multiple symbols of the same type as the preselected symbol instances. Any symbol instances which cannot be replaced with the alternate remain intact.

If any alternate symbols are defined for one or several selected symbol instances of different types, when you right click, each symbol name displays, and each of those names then expands into the available alternate symbols from which you may choose.

After an alternate symbol is chosen, a confirmer dialog box appears that lets you preserve or rip up etch/conductor. For procedural information, refer to the use altsym command.

15

Generating APD Paste Resistor Symbols

The Thick/Thin-Film Resistor Synthesizer generates thick- and thin-film paste resistor symbols in APD.

This section:

- Provides introductory information on the Thick/Thin-Film Resistor Synthesizer and how it works
- Identifies prerequisites for using the Thick/Thin-Film Resistor Synthesizer, including:
    - How component properties tables work and how to create them
    - The available resistor generation properties
    - How a resistor specification file works and how to create it
    - How the film resistor control file works and how to create it
- Describes how to run the `film res`*s* command

At the end of this section, see a complete list of topics covered in the balance of the appendix.

Thick/Thin-Film Resistor Synthesizer Fundamentals

The Thick/Thin-Film Resistor Synthesizer automatically generates thick-or thin-film paste resistor symbols. The resistor synthesizer uses ink characterizations and user-defined parameters to create resistor geometries. Parameters that you can define include:

- Resistor values and tolerances
- Minimum and maximum specified length and width
- Power dissipation
- Power/temperature darting factors

You can control size and reduce manufacturing costs by having the Resistor Synthesizer optimize the design area and inks generated.

The Thick/Thin-Film Resistor Synthesizer can perform the following processes during resistor creation, to identify potential manufacturing issues:

Resistor-loop checking	Locating cyclic loops enables more precise measurement of resistors before laser trimming.
Laser-trim analysis	Laser-trim checking selects the appropriate cut for the laser at the laser width and step size required to minimize burnout. Additional analysis routines ensure that the resistor geometries provide ample physical space for laser trim, based on the Kirov and step size of the laser during manufacturing.

The Resistor Synthesizer can also create and run the scripts that actually generate resistor component symbols and a board layout (`.brd`) file that contains various CONDUCTOR subclasses for the different materials used in each layer of your design.

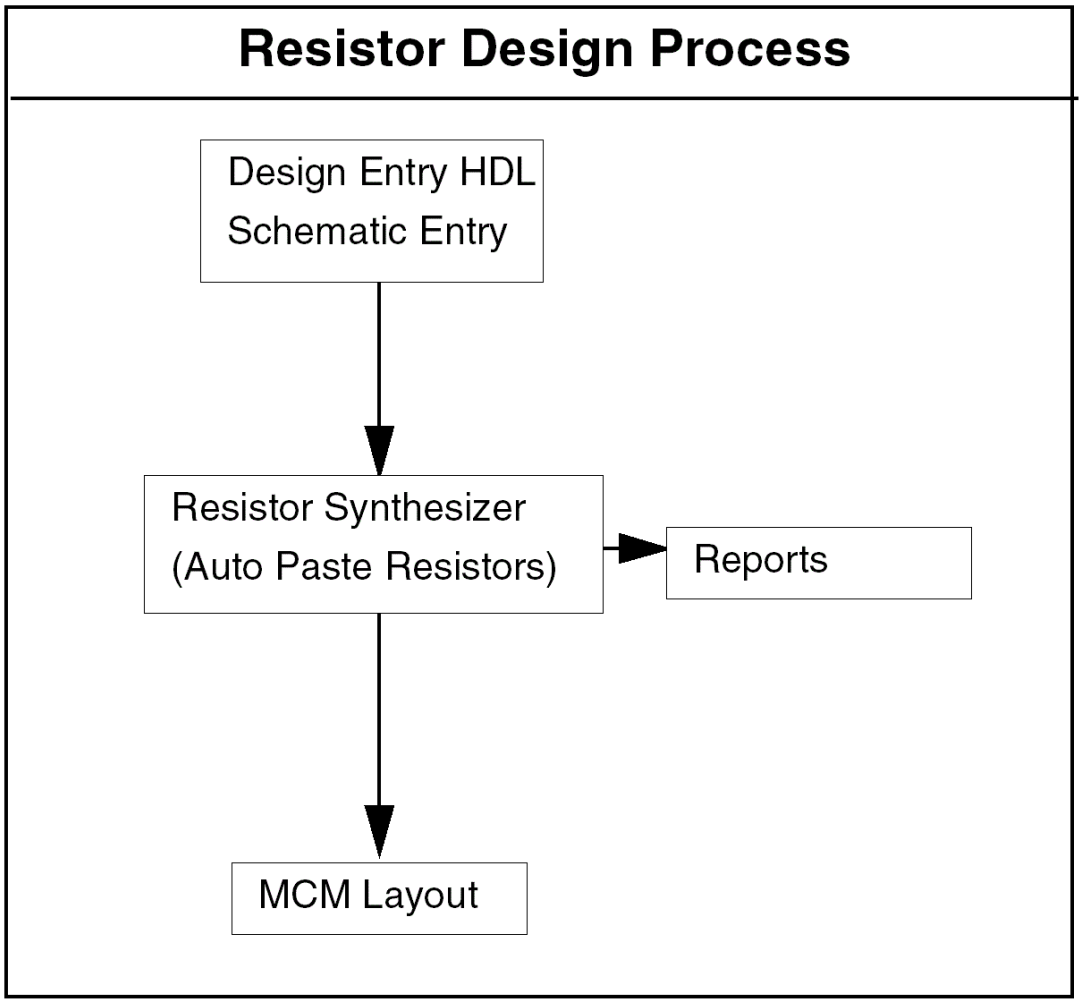
You can specify resistor property information, which further defines the type of resistors generated, through certain Cadence front-end tools, namely

Design Entry HDL	For schematic entry
The Compiler	For creating netlist files that describe the Design Entry HDL schematic
Packager-XL	For translating schematic information from the Compiler. Packager-XL packages parts into physical packages, translates Design Entry HDL signals to the tool's nets, and translates the Design Entry HDL nodes to the tool's pins.

The following figure shows the relationship between Design Entry HDL and the Resistor Synthesizer.

If you do not use Design Entry HD, the Resistor Synthesizer lets you supply resistor property information through a resistor specification file. This file is an ASCII text file that you create.

Software Tools for Resistor Synthesis



You control how resistors are generated by

- Specifying the *command directives* (parameters) for resistor generation in a film resistor control file
  Sample film resistor control files are supplied with the Thick/Thin-Film Resistor Synthesizer (in `<`*install_dir*`>/share/pcb/text`). You can use one of the sample control files, edit a sample control file, or create your own control file.
  You use the control file to specify:
  - The type of symbols (either thick- or thin-film) the resistor synthesizer will create
  - Parameters that indicate the type of processing that will occur during resistor generation,

such as resistor-loop checking and laser trimming

- Attaching properties to parts that further define the resistor generation process

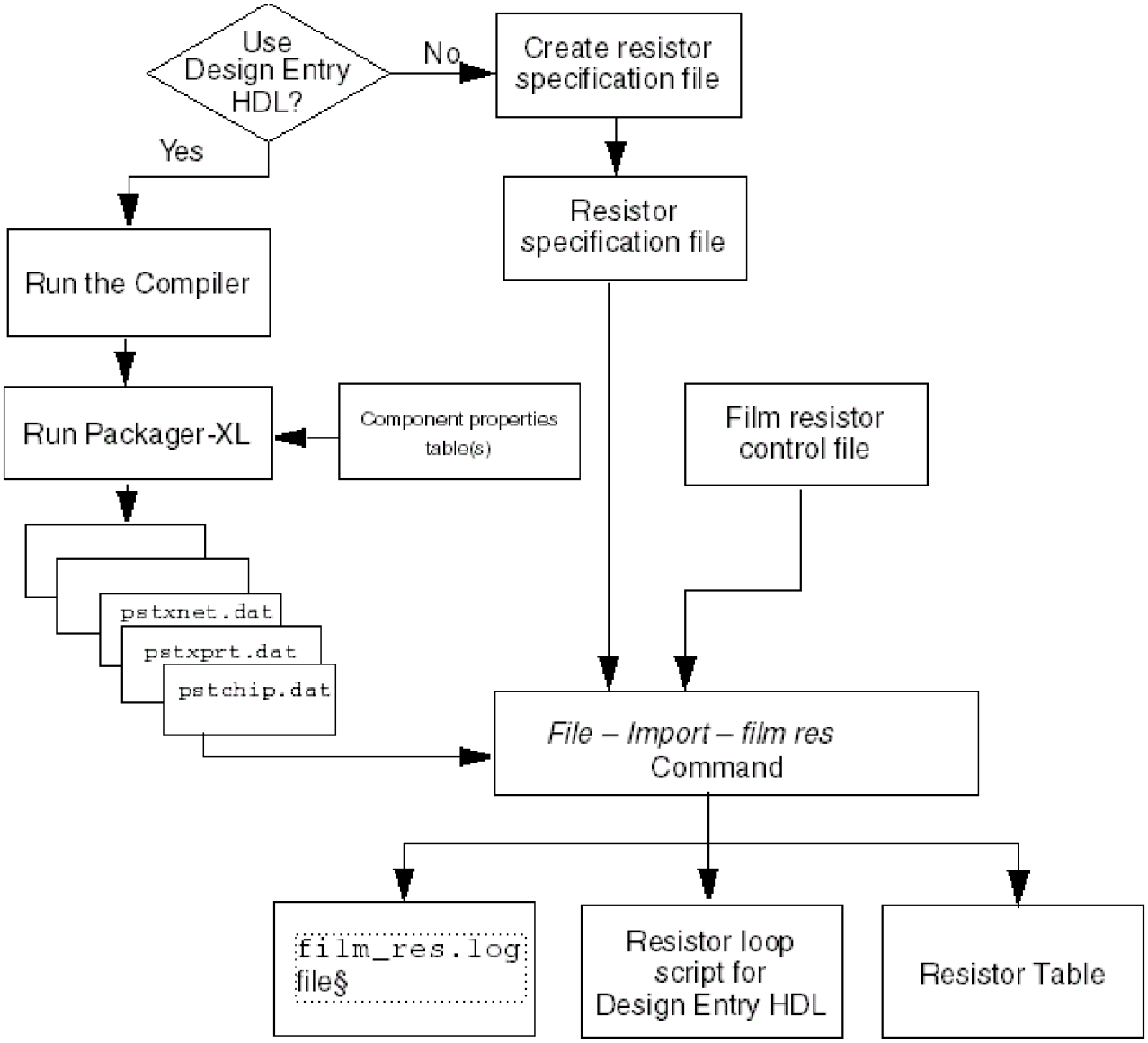
The way in which you specify resistor properties depends on whether you are using Design Entry HDL (which requires using the Compiler and Packager-XL) or not.

- If you use Design Entry HDL, you can specify resistor properties by:
  - Attaching the resistor generation properties to bodies in the Design Entry HDL schematic.
  - Applying properties to parts through *component properties tables* , which are *physical component tables* (that you create) for use by Packager-XL.
- If you are not using Design Entry HDL, you create a *resistor specification file* that identifies the resistor generation properties to be applied to resistor component types (this file contains information similar to the component properties tables that you can create if using Design Entry HDL).

You generate the actual resistor symbols by choosing *File – Import – Paste Resistor* (film res command).

The following figure provides an overview of the resistor generation process.

Resistor Generation Process



Resistor Generation Process

This section describes the steps typically used to design film resistors:

1. Set up the film resistor control file (`film_res.rcf`) and any other input files or data to be used.
2. Choose *File – Import – Paste Resistor* (film res command).

3. Choose *File – Import – Logic* (netin command) to update the design with the new netlist.
4. Place components, swap gates, and pins.
5. Backannotate changes to the schematic.
6. Route interactively or automatically.
7. Gloss to make lines as short as possible.
8. Add test points.
9. Automatically generate crossover dielectrics.
10. Manually add or edit local dielectric, then add shapes for global dielectrics.
11. Void and edit global dielectrics.
12. Add a documentation symbol and dimension.
13. Prepare Gerber files and any other manufacturing output.
14. Penplot.

Related Topics

- Before Using the Thick/Thin-Film Resistor Synthesizer
- Setting Resistor Generation Controls
- Running the Thick/Thin Film Resistor Synthesizer

Before Using the Thick/Thin-Film Resistor Synthesizer

When designing film resistors, note the following before you use the Thick/Thin-Film Resistor Synthesizer:

- If you use Design Entry HDL, be aware that by default, the Resistor Synthesizer assumes resistor instances are thick-film.
  To generate thin-film resistors, you must attach the RES_TYPE=THIN property to resistor instances in the schematic. If you do not attach the RES_TYPE=THIN property to resistor instances in the schematic, and though your film resistor control file indicates that thin-film resistors are to be generated, the Resistor Synthesizer stops processing.
- Before you use the Thick/Thin-Film Resistor Synthesizer, do the following:
  1. Determine the layer (class/subclass) structure required for your design.
     To determine which subclasses are required, determine which elements are required on artwork layers.
  2. Determine whether routing is allowed under (over) resistors on any layer and define the component pad layers accordingly.
     If routing is not allowed under resistors and you only have one routing layer, keep this in mind during placement because you cannot route through resistor pins.
  3. Create a dummy padstack in APD for all resistor symbols to be generated by the Resistor Synthesizer.
     You can edit the padstack after you add the resistor symbols to a design.
- Some companies require that inks and laser trims go in the same direction, which can affect placement.
- When lining up chip and wire parts during placement, remember that the ratsnest lines go to the pin, but you will be routing to the bond-pad pin escape.
- Special characteristics about the Thick/Thin-Film Resistor Synthesizer are:
  - Resistors are built and viewed from the top side and mirrored to the back, if necessary.
  - Rectangular resistors can have S- or L-cuts; top-hat resistors always have S-cut.
  - Double-plunge trims are not allowed.

- Thin-film resistors are generated as rectangular or serpentine (also referred to as *snake*) shape resistors.

Setting Resistor Generation Controls

You set the controls that determine thick- or thin-film resistor generation before you choose *File – Import – Paste Resistor* (film res command). You specify the controls in the following files:

- Physical component tables (only if you use Design Entry HDL)
  One of the ways you can supply property information for schematic bodies is through physical component tables referred to as *component properties tables*. Packager-XL uses the physical component tables to map properties to schematic bodies.
  Component properties tables let you define different component types from one basic component type. You can also use component properties tables to specify properties for resistors that are different from the properties contained in the component type definitions in the schematic library.
  Another use of component properties tables is to attach new body properties to a component type. You can do this without having to recreate or change the library files containing the component type definitions.
- A resistor specification file (only if you do not use Design Entry HDL)
  The resistor specification file supplies all the property information needed by the Thick/Thin-Film Resistor Synthesizer to generate resistor component symbols.
- Film resistor control file
  The control file contains command directives that further define how thick- or thin-film resistor symbols are to be generated.

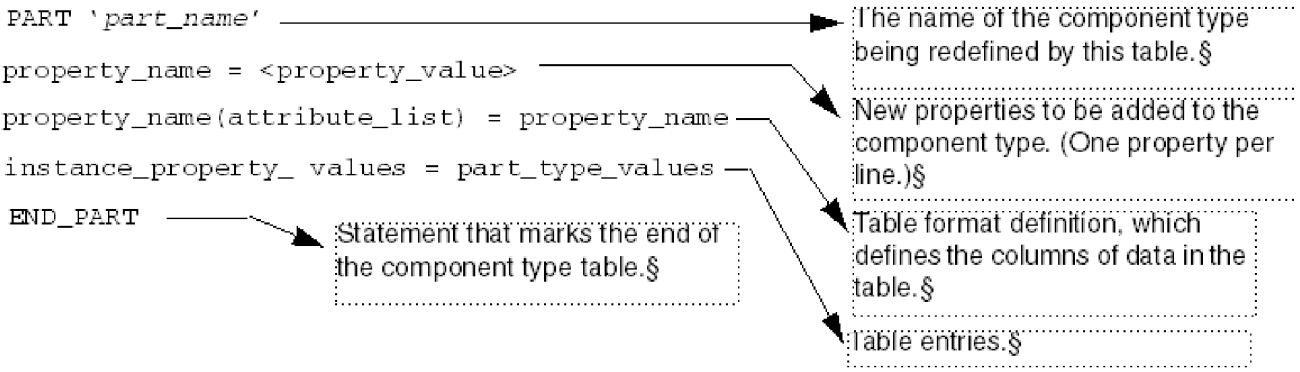
You create and locate these files in your working directory, so that the Thick/Thin-Film Resistor Synthesizer can use the resistor controls and properties when generating resistor symbols.

This section explains the following:

- How component properties tables work and how to create them
- The available resistor generation properties
- How a resistor specification file works and how to create it
- How the film resistor control file works and how to create it

A component type table defines the component properties for a specific component type in a component properties table. The following figure shows the format of a component type table.

Component Type Table Format



Following is an example of a component properties table with multiple component type tables. The following topics describe each section within a component type table.

Example of Multiple Component Type Tables in a Component Properties Table

```
FILE_TYPE = MULTI_PHYS_TABLE;
part 'c'
:location (opt) =jedec_type;
C1 (C1)          =capa1
C2 (C2)          =capa2
C3 (C3)          =capa3
C4 (C4)          =capa4
...
end_part

part mstf3
jedec_type = mstf3
end_part

part tl072
jedec_type = tl072
end_part
 
part 'r'
:location (opt)=value,tol,jedec_type, alt_symbols;
R1 (hybres1)    =28.7,1%,msr10, ( )
R2 (hybres2)    =5000K,1%,msr10, ( )
R3 (hybres3)    =20000K,1%,msr10, ( )
R4 (hybres4)    =20K,1%,mstf3, ( )
R5 (hybres5)    =100,1%,hybres5, (hybres5_1)
R6 (hybres6)    =5K,1%,hybres6,
(hybres6_1;B:hybres6_2)
R7 (hybres7)    =100,1%,hybres7, ()
R8 (hybres8)    =5K,1%,hybres8, (B:hybres8_1)
...
end_part
END.
```

New Properties for the Component Type

To add new properties to the component type, you specify each new property on a separate line in the file. For each property, you specify the corresponding property value.

Table Format Definition

The table format definition section of a component type table defines the column format of each line in the table for the resistor type. You can specify the following:

- Property and attributes for the property (if any)
  Attributes control how the Resistor Synthesizer processes properties. You can specify attributes that do the following:
  - Identify a property value as a string
    The property value on the instance must exactly match the value defined in the entry.
  - Identify a property value as a number
    This value should match the value of the table entry or should be inside a table entry range.
  - Identify a property value as a range
- The properties to be added to the new component types
- Separator character to be used between multiple properties
  If you have multiple properties in the table format definition section, you must separate each property listed with a separator character, such as a comma or blank space.

Attributes

You specify attributes for properties using the following indicators:

OPT	Indicates that the property is optional.
S	Indicates that the property value can be any string.
N	Indicates that the property value can be a number. The Resistor Synthesizer translates all these numeric values into real numbers, based on scale factor similar to the scale factors used in the Simulation Program with Integrated Circuit Emphasis (SPICE), or a scale file factor that you define.

	By default, the following set of scale factors (as defined by SPICE) is used to translate property values into real number values. T = 1E12 G = 1E9 MEG = 1E6 K = 1E3 M = 1E-3 U = 1E-6 N = 1E-9 P = 1E-12 F = 1E-15 This means that 1.234K, 1234, 1.234KOhm, and 1234 ohm are all translated to the same real value 1234. If the SPICE scale factors are not appropriate for your environment, define your own scale factor file that contains your set of scale factors.
R	Indicates the property values in the table are ranges. You specify the actual range in the table entries section of the component table file.

Table Entries

The table entries section of a component type table contains the actual property values for the instance properties specified in the table format definition.

The table entries can also include:

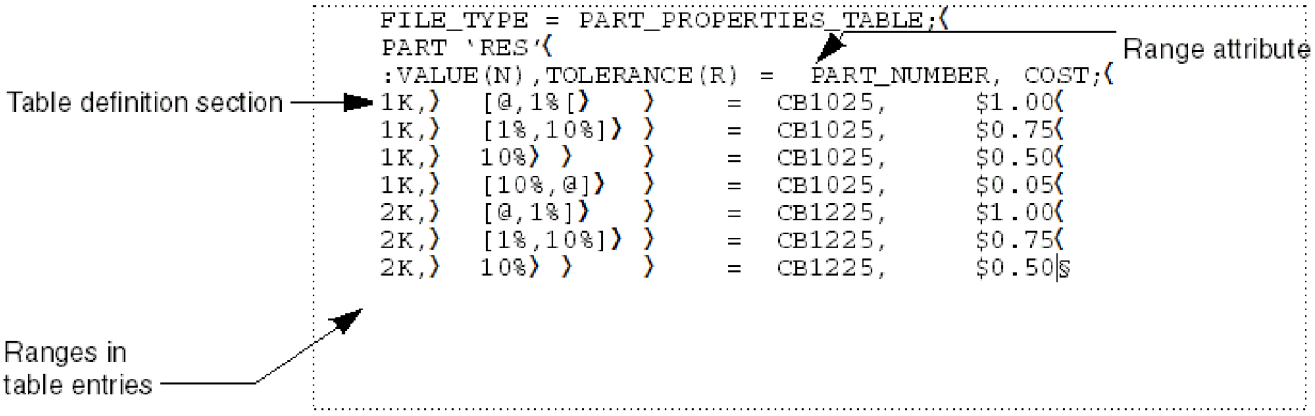
- Ranges if the range attribute is specified in the table format definition
- Component subtype names

Ranges

If you specified the range attribute for a property in the table format definition section of the component type table, you need to enter the appropriate range values in the table entries section.

For example, in the following, the table definition section has a range attribute for the TOLERANCE property.

Example of Range Attributes in a Component Properties Table



The cost and tolerance values for 1K resistors described above are the following:

- '1K' resistors with a TOLERANCE less than 1% have a COST of '$1.00'.
- '1K' resistors with a TOLERANCE greater than or equal to 1% and less than 10% have a COST of '$0.75'.
- '1K' resistors with a TOLERANCE equal to 10% have a COST of '$0.50'.
- '1K' resistors with a TOLERANCE greater than 10% have a COST of '$0.05'.

To specify a range

- Use one of the following formats:

```
border_char  value separator_char value     border_char value
```

–or–

```
value
```

`border_char`	A [(left bracket) or] (right bracket). To include a value in a range specification use the [at the left side, or a] the right side of the range. To exclude a value from a range specification you must have the] character at the left side, or the [at the right side of the range.
`value`	Any SPICE-like (with your own scale factors) number, or the @ (at sign), which indicates infinity.
`separator_char`	A colon (:) or a comma.

The second form indicates a range which contains only one discrete value. This form is the same as

```
[ value : value ]
```

Redundant Range Specification

The Resistor Synthesizer does not check the validity of the different ranges. It compares a property on an instance of the component with the range values in the same order as specified in the table.

For example, in the following table the second table entry is redundant.

```
FILE_TYPE = PART_PROPERTIES_TABLE;

PART 'RES'

:VALUE(N),     TOLERANCE(R) =  PART_NUMBER,      COST;

1K,     [@,10%[ =  CB1025,     $1.00

1K,     [2%,5%[ =  CB1025,     $0.50

1K,     [10%,@] =  CB1025,     $0.05

END_PART

END.
```

Component Subtype Names

The Resistor Synthesizer usually generates new component types with the same name as the original component type. Packager-XL generates new component type names for subtypes by appending a - (dash) and an integer to the component type name.

For example, subtype names for the component type 'RESISTOR' are 'RESISTOR-1', 'RESISTOR-2' and 'RESISTOR-3'.

You can assign a subtype name in the component properties table by specifying a suffix after the property name. By using this feature, the subtype name remains the same each run of the program. Subtype names are suffixes that you enclose with ( ) parentheses.

To specify subtypes, be sure to do the following:

- Set `part_type_length` directive in the command file to control the subtype length
- Specify the name of the component properties table files to be used in resistor generation in the `part_table_file` directive in the control file

⚠ Cadence recommends that you use this subtype specification whenever possible.

The following shows a component properties table with subtype names.
Component Properties Table with Subtype Names

```
1.)  FILE_TYPE = MULTI_PHYS_TABLE;(
2.)  PART 'RES'(
3.)  :VALUE(N),)     TOLERANCE )     =PART_NUMBER, )COST;(
4.)  1K,) )     )     2% (1K)  )     =  1285,) )     $.50(
5.)  2.3K,)     )     1% (2.3K))     =  1300,) )     $.50(
6.)  1K,) )     )     5% (1K,5%)     =  1024,) )     $.24(
7.)  5K,) )     )     1% (!)   )     =  1000,) )     $.43(
8. ) 1K,) )     )     3% (!)   )     =  1028,) )     $.24(
9. ) 1K,) )     )     4% ) )   )     =  1028,) )     $.24(
10.) END_PART(
11.) END.¶
§
```

The suffix types can be the following:

Explicit	Specify the actual suffix within ( ) parentheses after the component subtype name. Packager-XL appends the actual suffix you specify to the component type name.
	For example, line 5 in the previous figure results in the subtype name ‘RES-2.3K. The subtype name for line 6 would be ‘RES-1K,5%.
Implicit	Specify the suffix with an (!) [exclamation point in parentheses]. Packager-XL appends the property values to the component type name. It places commas (,) between property values.
	For example, line 7 in the figure results in the subtype name ‘RES-5K,1%’. The subtype name for line 8 would be ‘RES-1K,3%.

The following characters are allowed in a suffix:

- A through Z (upper- and lowercase letters)
- 0 through 9
- Special characters, including: , $ % # & * + - and .

The length of the subtype name cannot exceed the length of a legal component type name as defined by default or the part_type_length command directive. If the name is longer than this limit, the Resistor Synthesizer generates an error message and truncates the name.

If you do not specify a suffix, a numeric suffix is created and appended to the component type name. For example, line 9 in the previous example might be ‘RES-1’.

This section describes additional specifications that can be added to the parts properties table.

The JEDEC_TYPE property is used by the Resistor Synthesizer as the resistor script name and symbol name to ensure a match between the netlist content and the resistor symbols.

⚠ Each resistor should have a JEDEC_TYPE property assigned to it.

The component identified in the following example can be automatically generated. All the necessary information can be extracted from the schematic.

```
part 'r'

:location (opt) =     value,     tol, jedec_type;

R5 (hybres5) =     100,      1%, hybres5

R6 (hybres6) =     5K,      1%, hybres6

R7 (hybres7) =     100,      1%, hybres7

R8 (hybres8) =     5K,      1%, hybres8

R9 (hybres9) =     5K,      1%, hybres9

end_part
```

After having defined the resistor symbols in APD, this netlist should be read in and assigned.

Other components besides resistors should also be assigned through a JEDEC_TYPE property to a corresponding component from the tool library.

To format a component type table in a component properties table:

1. Enter the name of the component type using the following format:

   PART '*part_name*'

part_name	Name of the component for which the subsequent properties are being defined.

2. Enter the component type property list using the following format:

   property_name = <*property_value*>

property_name	Standard property name, consisting of up to 16 characters that include letters, digits, and an _ (underscore). The name must begin with a letter.

property_value	Any string terminated by the end of the line. If the value is too long and cannot fit on one line, enter a tilde (::) in the last position of the line as a continuation character. The tilde may appear between any two characters in the line. For example, the following lines are equivalent: LIB_SHAPE = EIA1212 LIB_SHAPE = EIA:: ⚠ If you want leading or trailing spaces around property values, enclose the property values with either single quotes or double quotes. If you need to use quotes as component of the property value, you can still use the quote notation to indicate leading or trailing spaces. You must use the quote notation that is not component of the value. For example, if you use single quotes as component of the property value, use double quotes to enclose the property value.

3. Enter the table format definition using the following format:

instance_property_list = <*part_property_list*>;

instance_property_list	List of property names that can be attached to an instance of a component. The format of this list is explained below.
part_property_list;	List of properties associated with the new component types. The format of this list is explained below, after the explanation of the *instance_property_list* format. The *instance_property_list* has one of the following formats: *property_name property_name* `(attribute_list) separator` *property_name* Standard resistor generation property name, consisting of up to 16 characters that can include letters, digits, and an _ (underscore). The name must begin with a letter.
	attribute_list List of attributes that apply to the property. If you have multiple attributes in the list, use commas to separate them. The attributes are the following: *OPT* Indicates the property is optional. This attribute can also have the format `OPT = '`*default value*`'` where *'default value'* indicates the property is optional and provides a value to be used if a value does not exist for the component instance. If the default value contains spaces, enclose the value in quotes.

	S Indicates the property value can be component of a string.The property value on an instance of a component should exactly match the value specified in the table. If you do not specify any attribute, the default is S. N Indicates the property value is a number. The Resistor Synthesizer translates all property values of the table having this attribute value into a real number. When the Resistor Synthesizer finds one of these properties on an instance, it first translates it into a real number. Then it compares the number with the table entries. If the Resistor Synthesizer finds an error during the translation into a real number, it generates an error message.
	R Indicates the property values specified are ranges. It also indicates the property value found on an instance of the component is a number. To match an entry this value should be in one of the specified ranges. *separator* A character that separates more than one property name and attributes in the list. The character cannot be a letter, digit, _ (underscore), = (equal sign), ( ) (left or right parentheses), { } (left or right brace), :: (tilde), : (colon), ; (semicolon), ’ (single quote), or " (double quote).

⚠ If you use the range property attribute, do not use [or] (brackets) as a separator.

The *part_property_list* has one of the following formats:

```
property_name;

property_name separator ...;
```

property_name	Property name, consisting of up to 16 characters that can include letters, digits, and an _ (underscore). The name must begin with a letter.
	You can specify as many properties as necessary. If you have more than one property name and attributes in the list, separate the properties with a separator character.
separator	A character that separates multiple properties in the list. The separator can be a space which allows any number of spaces or tab characters to appear.
;	A semicolon, which marks the end of the table definition.

4. Enter the table entries using one of the following formats:

```
instance_values = part_type_values

instance_values = part_type_values : new_properties
```

You use the second format only when you add new properties for the component type created for this table entry. A colon (:) separates the last `part_type_value` from any `new_properties`.

instance_values	A list of property values for the instance properties specified in the table format definition. They should meet the attributes as established in the table format definition section. If the attribute for the instance property is R (range), format the property type values using the range specification explained in "Ranges".
part_type_values	The actual values for each property instance
New_properties	The Resistor Synthesizer adds the list of *new_properties* to the new component type created for a particular table entry. The *new_properties* uses one of the following forms: *property property, property*. . . where each *property* has the form: *property_name = 'property_value'* ⚠ Enclose the *property_value* in quotes if it contains spaces. Each table entry must appear on one line. If an entry is too long, use the tilde (::) as a continuation character.

The following is an example:

```
FILE_TYPE = PART_PROPERTIES_TABLE;

PART '1/4W RES'

:VALUE(N) = PART_NUMBER,:: COST;

1K = CB1025,:: $0.05

1.2K = CB1225,:: $0.05

1.5K = CB1525,:: $0.05

.

.

END_PART

END.
```

Example Table Format Definitions

The following are examples of different table format definitions for different properties and attribute types:

Example of Optional Attribute

```
:TOL(OPT= ’5%’) = MIN_WIDTH;
```

In the above example, the TOL property is optional on the component. If the property is not present on the component, the default value 5% is taken and no warning messages are generated.

Example of Optional and Range Attribute

```
:TOL(OPT= ’5%’,R) = MIN_WIDTH;
```

In the above example, the TOL property is optional, the TOL property value is a number with the default value 5%, and the table entries specify ranges.

Example of a Separator in a Table Format Definition

The Thick/Thin-Film Resistor Synthesizer expects that each line in the following table example starts with a VALUE property followed by an equal sign (=). The next fields are PART_NUMBER and COST property values.

```
FILE_TYPE = PART_PROPERTIES_TABLE;

.

.

PART ‘RES’

.

.

:VALUE      =  PART_NUMBER      COST;

1K      =  CB1025     $0.05

1.2K      =  CB1225     $0.05

1.5K      =  CB1525     $0.05

2.2K     =  CB2225     $0.05

2.7K     =  CB2725     $0.05

.

.
```

```
END_PART

END.
```

In this example, the separator character is a space. When the separator character is a space, any number of spaces or tab characters can appear.

Control directives in a film resistor control file fall into six different categories:

Input controls	Control directives that identify the input files the Resistor Synthesizer must use
Dimension controls	Control directives that control the size or dimensions of the following: drawing size, trimming amount, grid, maximum length and width of generated resistors, minimum length and width of generated resistors, and units
Output controls	Control directives that indicate where the generated resistor symbols are to be located and how output processing is to occur (display of warning messages, number of errors allowed before processing stops, and so on)
APD controls	Control directives that determine how the resistor symbols are to be generated in APD
Resistor generation controls	Control directives that define how the Resistor Synthesizer is to generate the resistor symbols
Ink placement controls	Control directives that describe how the paste is to be applied

Sample Thick-Film Control File

Following is a sample command file for thick-film resistors. Note that the comments in braces explain how each directive works.

Sample Film Resistor Control File for Thick Film Resistors

```
technology thick
```

```
root_drawing test    { name of the root_drawing                }
use sample.wrk    { directory where the drawings are        }
units mil
{ Dimensions }
  grid 5     { put things on a 5 Mil grid             }
  min_length 30     { min for length = 40 mils               }
  min_width 30    { min for width = 40 mils                 }
  trimming laser 2 0.4 10 50     { layer=laser w=2 st=0.4 sp=10 min perc=30}

{ Resistor controls : }
temp 50degrees    { operate the device at 50 degree         }

{ RESISTOR ink description : FRONT of SUBSTRATE }
resistor_base substrate    { following are resistors on substrate    }
   resistor_pads top 15 5    { pad layer=top over=15 enc=5 }
   allegro_route_keepout_subclass bonding 5    { route keepout on layer bonding }
allegro_route_keepout_subclass top 5        { route keepout on layer top  }

   { resistor command format :
        resistor - keyword
        |    subclass name
        |    |    ohms/sq
        |    |    |     power
        |    |    |    |    current density (mA)
        |    |    |    |    |     Ptol (%)
        |    |    |    |    |    |    V Grad
        |    |    |    |    |    |    |    tophat_Ptol (%)
        |    |    |    |    |    |    |    |
        |    |    |    |    |    |    |    |     }
        resistor    r1    10     50mW    1    25    0     25
        resistor    r2    100    50mW    1    25    0    25
        resistor    r3    1000    50mW    1    25    0    25
        resistor    r4    10k    50mW    1    25    0    25

{ RESISTOR ink description : BACK of SUBSTRATE }
resistor_base back    { following are resistors on substrate    }
   resistor_pads bottom 15 5    { pad layer=bottom over=15 enc=5 }

{ resistor command format :
        resistor - keyword
        |    subclass name
        |    |    ohms/sq
        |    |    |    power
```

```
        |    |    |    |    current density (mA)
        |    |    |    |    |    Ptol (%)
        |    |    |    |    |    |    V Grad
        |    |    |    |    |    |    |     tophat_Ptol (%)
        |    |    |    |    |    |    |    |
        |    |    |    |    |    |    |    |                      }
        resistor    r3    1100    50mW    1    25    0      25
        resistor    r4    11k     50mW    1    25    0    25

{ ================================================================

  =============== Additional commands ============================== }


{ OUTPUT CONTROL :                              }


max_errors 100    { program stops if more than 100 errors   }

oversights on    { we want to display oversights           }

warnings on    { we want to display warnings             }


resistor_loops loops.script    { name of GED loops script               }
resistor_table resistor.tab    { name of resistor table                 }
part_table_file resistor.ptb    { use property part table                }



{ DIRECTIVES for controlling APD :           }
allegro_solutions valid    { all valid solutions must be generated   }
allegro_pad_shaped off    { padstacks are generated                 }
{
allegro_route_keepout_all 20    { if allowed, use ROUTE KEEPOUT ALL       }
}
allegro_pad_subclass pastemask 5 { additional etch data on top of shaped pads }

allegro_pad_subclass soldermask -5

allegro_dummy_padstack 'pinres.pad'  { dummy padstack name for etched pads }


{ Directives for controlling RESISTOR generation         }
default_base substrate
default_backside_base back
default_trim L_CUT    { enforce L_CUT on all rect resistors with
                                  no trim_property specified             }
```

```
{
default_shape RECT      { only want rectangular resistors         }
}
default_orientation HORIZ
area_optimization on
ink_optimization on    { don't want to use as few inks as possible}
resolution 100    { using standard led unit                 }
number_of_squares 0.2 5    { number of square must be > 0.2 and < 5  }
power_coef 1     { power multiplied by 1 before computation }
power_derating 0.625mW 125    { power derating starting at 125 degrees  }
power_units mm    { power is given in Watt per square mm    }
tophat_number_of_squares 4 15    { number of square must be > 4 and < 15   }
tophat_min_width 20     { min for width tophat = 20 mils          }
{
tophat_leg_length 50     { min tophat legs = 50% of width          }
}
trim_check on
```

Sample Thin-Film Control File

Following is a sample control file for thin-film resistors. Note that the comments in braces explain how each directive works.

Sample Control File for Thin-Film Resistors

```
{
  This is an example of the "film_res.ref" file for generating Thin Film
  Resistors. Users may have to change the parameter values to suit their
  application. Please refer to the Cadence "Thick/Thin Film Resistor
  Synthesizer User Guide" for detail description of each parameter.
}
technology thin

root_drawing test    { name of the root_drawing             }
use sample.wrk     { directory where the drawings are     }

units mil
{ Dimensions }
   grid 5
  min_length 30     { min for length = 30           }
  min_width  30     { min for width = 30           }
  trimming laser 2 0.01 10 30  { layer=laser w=2 st=0.01 sp=10 min width=30% }
```

```
  allegro_pad_subclass pastemask 5 { additional etch data on top of shaped pads }
  allegro_pad_subclass soldermask -5
{ -- end of dimensions }

{ Resistor controls :           }
temp 50degrees    { operate the device at 50 degree     }

{ RESISTOR ink description : FRONT of SUBSTRATE }
resistor_base substrate    { }
   resistor_pads top 15 5    { pads layer=top over=15 enc=5         }
   allegro_route_keepout_subclass top 5    { route keepout on layer top }
   allegro_route_keepout_subclass bonding 5
 { route keepout on layer bonding       }
   { resistor command format :
resistor - keyword
|    subclass name
|    |    ohms/sq
|    |    |    power
|    |    |    |    current (mA)
|    |    |    |    |     Ptol (%)
|    |    |    |    |    |
|    |    |    |    |    |    }
resistor    r2ko     2k    50mW    2.5    25
   allegro_subclass r2ko     allegro_subclass = layer             }
{ end of RESISTOR ink description }



{ ================================================================
  =============== Additional commands ==============================    }

{ ADditional commands for controlling RESISTOR generation     }
default_base substrate
default_orientation VERT
power_coef 1             { power multiplied by 1 before computation }
power_derating 0.625mW 125    { power derating starting at 125 degrees }
power_units mm     { power is in Watt per square mm     }
trim_check on


{ OUTPUT CONTROL :                          }
max_errors 100    { program stops if > 100 errors    }
oversights off    { display oversights             }
```

```
warnings on     { display warnings              }
suppress 31 32

resistor_loops loops.script    { name of Design Entry HDL loops script     }
resistor_table restab



{ Commands for controlling APD :        }
allegro_pad_shaped off    { only padstacks are used      }
{
allegro_route_keepout_all 5    { if allowed, use ROUTE KEEPOUT ALL     }
}
allegro_dummy_padstack 'pinres.pad'    { dummy padstack name      }
```

1. Using a text editor on your system, open a file that you name `film_res.rcf` or edit the `film_res.thick` or `film_res.thin` sample command file provided in <*install_dir*>/share/pcb/text.
2. In the first line of the file, enter the technology directive that identifies whether thick or thin film resistors are to be generated by the Resistor Synthesizer.
   For example

   technology thin

   The sample files specify the technology in the first line of the file.
3. Enter the command directives that control the resistor generation process.
   You do not have to enter the command directives in any particular order, except for the directives that control ink placement. You may find it easier to organize your file according to the type of directive (input controls, APD controls, resistor generation controls, and so on) for readability and tracking purposes.

   ⓘ When you specify the directives for ink placement, you must specify them in the following order:

   - *resistor_base*
   - *resistor_pads*
   - *allegro_route_keepout_subclass*
   - *resistor*
4. Save the file.

Related Topics

- Properties for Resistor Generation
- Control File Directives for Film Resistors

Running the Thick/Thin Film Resistor Synthesizer

This section explains how to run the Thick/Thin-Film Resistor Synthesizer using the `film res` command.

This section describes:

- Prerequisites for running *File – Import – Paste Resistor* (film res command)
- Running *File – Import – Paste Resistor* (film res command)
- The output generated by *File – Import – Paste Resistor* (film res command)

The Resistor Synthesizer checks whether all the resistors can be trimmed (trimmable from a measuring point of view). All resistors for which the resistance value cannot be measured independently from the rest of the circuitry are labeled as part of a loop.

When trimming the resistors, cyclic loops between several resistors make it hard to measure each resistor separately while trimming it.

You enable this checking by setting the resistor_loops <*script_file*> directive in the film resistor control file.

⚠ Cadence recommends that you always include the resistor_loops <*script_file*> directive in your control file to identify errors early in the resistor generation process.

The <*script_file*> is the generated script for Design Entry HDL that highlights the erroneous loops in the design when executing it within Design Entry HDL. Also highlighted are the nets connecting these resistors.

Cyclic loops detected during processing are listed in the log file as follows:

```
Starting to check cyclic loops for 'CHIP'

WARNING : Following resistors are part of a loop :

    R(R2)

    R(R3)

    R(R4)

You might have difficulty trimming them.

Finished checking for cyclic loops (00:00:00.27)
```

When checking for resistor loops, all other components beside film resistors are considered not yet

placed in the design. The existing connections to all those components are treated as open circuitry.

A resistor loop script might look as follows:

```
exclude        A  BO  WI  PR  NE  CO
find location= R4
include        A  BO
exclude        A  PR
exclude           B  BO  WI  PR  NE  CO
find location= R3
include        B  BO
exclude        B  PR
exclude        C  BO  WI  PR  NE  CO
find location= R2
include        C  BO
exclude        C  PR
show      group   A
show      group   B
show      group   C
show      net  "UN$1$3403$11P$INP"
show      net  "UN$1$R$6P$B"
show      net  "UN$1$R$7P$B"
```

The script, when run in Design Entry HDL, identifies the resistors for which measurements could not be taken, which might indicate potential trimming problems.

The following example shows what the Resistor Synthesizer generates if you include the `part_property_table` directive in the film resistor control file.

```
FILE_TYPE = MULTI_PHYS_TABLE;
 part ‘r’
 class = discrete
 :location (opt) = value,tol, jedec_type,
alt_symbols;
 R1 (R1) = 28.7OHM,     1%, hmcres1, ’()’
 R2 (R2) = 5MOHM,     ---, hmcres2, ’()’
 R3 (R3) = 20MOHM,     ---, hmcres3, ’()’
 R5 (R5) = 100OHM,     1%, hmcres5,
 (hmcres5_1,hmcres5_2)’
 R6 (R6) = 5KOHM, 1%, hmcres6,

’(hmcres6_1,hmcres6_2;B:hmcres6_3;B:hmcres6_4;B:
```

```
 hmcres6_5)'
...
 R17 (R17) = 27.5KOHM, 1%, hmcres17,
 '(hmcres17_1,hmcres17_2)'
 R18 (R18) = 71.5KOHM, 1%, hmcrs18,
 '(hmcres18_1,hmcres18_2;B:hmcres18_3;B:hmcres18)'
end_part
END.
```

The content of this component table file is assembled by interpreting the Design Entry HDL schematics or the content of the resistor specification file.

⚠ You do not have to specify the entire list of resistors in the design.

The directives in the control file and properties on the resistor instances in the design may result in more than one resistor geometry per instance. The alternate solutions generated are treated as alternate symbols and listed accordingly in the previous component table file example.

For alternate solutions on the backside of the substrate (listed as *B:* in the example), the `mirror` command issues an alternate symbol instead of mirroring when interactively placing the symbols.

Vertical solutions are treated as alternate symbols. When placing "rotated" resistor geometry, use one of the alternate symbols—do not use the `rotate` command.

⚠ Differences only occur when `correction_curves` are being defined.

For more information, refer to the descriptions of the following control file directives, namely `default_orientation`, `default_backside_base`, `default_trim`, and `allegro_solutions`, to identify which alternate solutions, if any, are being generated, and how they can be influenced.

⚠ For APD to interpret the content of a component table file, the Packager-XL command file must reference it. The Resistor Synthesizer reruns Packager-XL when necessary.

Related Topics

- Control File Directives for Film Resistors.

Control File Directives for Film Resistors

In related topics, you will find the following:

- A quick reference table listing the control directives applicable to thick- and thin-film resistors. The table is organized by category:
  - Input controls
  - Dimension controls
  - Output controls
  - APD controls
  - Resistor generation controls
  - Ink (paste) placement controls
- A detailed description of each directive and their corresponding arguments, arranged in alphabetical order.

Related Topics

- Control Directive Table
- Control Directive Descriptions

Control Directive Table

The following table lists the control directives available and indicates which directives can be specified in thin- or thick-film control files.

Directive	Brief Description	Used by Thin-Film	Used by Thick-Film
Input Controls			

`dir` or `directory` <*scald_dir*> [<*scald_dir* > . . .]	The name of the SCALD mapping file that the Resistor Synthesizer searches for generated cells	3	3
`lib` or `library` <*lib_name*> [<*lib_name*> ...]	The name of the libraries that the Resistor Synthesizer searches for cells not found in SCALD mapping files listed in the `dir` or `directory` directive	3	3
`library_file` <*chips_file_name*> [<*chips_file_name*>...}	The full path name of the library file containing physical information for components in the schematic design	3	3
`master_library` <*masterlib_name*> [<*masterlib_name*> . . .]	The names of several master libraries that the Resistor Synthesizer automatically searches	3	3
`part_table_file` <*part_table_name*>	The names of the resistor component properties table(s) containing property information	3	3
`resistor_specs` <*resistor_specs_file_ name*>	The name of an input file that contains resistor information to be used instead of a Design Entry HDL schematic		3
`root_drawing` <*root_drawing_name*>	The name of the Design Entry HDL root drawing	3	3
`scale_factor_file` <*scale_factor_filename*>	The name of a file that contains alternate scale factors used in component properties tables.	3	3
`technology <thin>` or `<thick>`	Identifies whether thin- or thick-film resistors are to be generated (must be specified on first line of control file)	3	3
`use` <*scald_dir*> [<*scald_dir* > . . .]	An alternate control directive for `dir` or `directory`. See `dir`.	3	3
Dimension Controls			
grid <*grid_value*>	The internal grid spacing (in microns unless you use the scale directive) on which all resistor geometry is based.	3	3

`max_length` <*value*>	The maximum length allowed for rectangular resistors generated (in microns, unless you use the `scale` directive).		3
`max_width` <*value*>	The maximum width allowed for rectangular resistors generated (in microns, unless you use the `scale` directive)		3
`min_length` <*value*>	The minimum length allowed for rectangular resistors generated (in microns, unless you use the `scale` directive)	3	3
`min_width`	The minimum width allowed for rectangular resistors generated (in microns, unless you use the `scale` directive)	3	3
`trimming` <*t_layer*> <*t_width*> <*t_step*><*t_space*> <*t_min*>	Required. Parameters that control the amount of resistor trimming	3	3
Output Controls			
`max_errors` <*max_errors_value*>	The maximum number of errors allowed before the Resistor Synthesizer stops	3	3
`oversights <on | off>`	If on, disables the display of oversights	3	3
`resistor_table` <*resistor_table_file*>	The name of the file where the resistor information will be output	3	3
`suppress <suppress_number>` [<*suppress_number*> ....]	Prevents the display of the specified number of warning and oversight messages	3	3
`warnings <on | off>`	If on, displays warning messages	3	3
APD Controls			

`allegro_dummy_padstack` *<padstack_name>*	Required. The name of a dummy padstack to be used when shaped resistor pads are generated	3	3
`allegro_pad_shaped <on | off>`	If on, APD generates resistor pads as filled rectangles inside the resistor component symbol	3	3
`allegro_pad_subclass` *<subclass_name> <value>*	The name of an CONDUCTOR subclass and the size by which shaped resistor pads are to be expanded or decreased on the specified subclass	3	3
`allegro_route_all_keepout_all` *<oversize_value>*	The oversize route keepout amount in microns (unless you use the `scale` directive) that determines the route keepout size over the resistor ink for subclass ALL	3	3
`allegro_route_keepout_subclass` *<pad_subclass> <oversize_value>*	The subclass of the route keepout and the oversize route keepout amount in microns (unless you use the `scale` directive)	3	3
`allegro_solutions < optimal | valid | all >`	Depending on the value specified, generates alternate symbols containing different solutions for a particular resistor		3
`allegro_subclass` *<force_subclass>*	Required for Thin Resistors only The layer or subclass (ink) for all resistors generated	3	3
Resistor Generation Controls			
`area_optimization <on | off>`	If on, the Resistor Synthesizer produces the layout of all resistors using the smallest possible area		3
`correction_curve`	The correction curves that indicate the resistivity of the resistor subclasses		3
current_coef *<factor_value>*	The safety factor used by the Resistor Synthesizer to multiply the nominal current (indicated for the resistor)	3	3

`current_units <micron |cm | mm | mil | inch>`	The current density unit used in the `resistor` directive	3	3
`default_base` *<default_base_name>*	The name of the resistor base that defines the set of inks from which to choose when generating a resistor		3
`default_backside_base_name` *<default_backside_ base_name>*	The name of the resistor_base that defines the set of inks from which to choose when resistors are generated on the backside of the substrate (mirrored resistors) This does not apply to Thin Resistors.		3
`default_orientation <horiz | vert>`	Generates the resistors either horizontally or vertically, depending on the argument specified		3
`default_shape <rect | tophat>`	The default shape, either rectangular or tophat, for resistors that do not have a SHAPE property attached		3
`default_trim` *<trim_type>* [*<trim_type>* ...]	The type of trim (single plunge cut, L-cut, or dual-plunge cut)		3
`delta_correction`	The parameters used to compute the `delta_tolrc` value	3	3
`extend_serpentine <on | off>`	If on, the Resistor Synthesizer can extend the first and last leg of a serpentine resistor so that its dimensions exactly meet the number of squares requirement if the resistor is not be trimmed		3
`ink_optimization <on | off>`	If on, the Resistor Synthesizer uses as few inks (pastes) as possible in the layout of all resistors This does not apply to Thin Resistors.		3
`number_of_squares` *<min_square>* *<max_square>*	The minimum and maximum number of squares for the rectangular resistors generated		3

`power_coef` <*value*>	A safety factor. The Resistor Synthesizer multiplies this factor by the nominal power in resistor calculations	3	3
`power_derating` <*derating_factor*> <*derating_temp*>	The maximum power density that decreases linearly from a certain temperature		3
`power_units <micron | cm | mm | mil | inch>`	The power density unit to be used in the resistor directive	3	3
`resistor_loops` <*script_filename*>	Enables the automatic checking of cyclic loops during resistor trimming, and writes a Design Entry HDL script file containing the controls for highlighting the existing loops (resistor bodies and their corresponding nets) Cadence recommends that you always include the `resistor_loops` <*script_ file*> directive in your control file to identify errors early in the resistor generation process.	3	3
`resolution` <*resolution_value*>	The internal units that serve as a scale factor between the user units and the internal units for calculation purposes	3	3
`square_correction` <*coef_a*> <*coef_b*>	The coefficient values used to correct the number of squares	3	3
`temp` <*temperature*>	The operating temperature of the thick- or thin-film resistor		3
`tophat_correction_ curves <on | off>`	If on, enables the use of correction curves in the calculation of tophat resistor geometry		3
`tophat_number_of_ squares` <*min_square*> <*max_square*>	The minimum and maximum number of squares of generated tophat resistors		3
`tophat_max_width` <*max_width*>	The maximum width of generated tophat resistors		3
`tophat_min_width` <*min_width*>	The minimum width of generated tophat resistors		3

`tophat_leg_length` *<length>*	The actual length of the tophat resistor legs, relative to their width		3
`trim_check <on | off>`	If on, disables the checking of resistors for trimming		3
`voltage_units <micron | cm | mm | mil | inch>`	The voltage gradient unit to be used in the resistor directive	3	3
`wider_serpentine <on | off>`	If on, increases the serpentine width to enhance trimming	3	
Ink (Paste) Controls			
`resistor_base`	The base (bottom layer) name for the resistor and resistor_pad directives	3	3
resistor	Required. Parameters that describe all resistive layers/subclasses	3	3
`resistor_pads` *<pad_layer> <pad_ovl> <pad_encl_w>*	Parameters that determine the two pads (top and bottom pads) created for the generated resistors	3	3
`units <mm | mil | micron | inch | cm>`	The units specification used in the `film res` control file. The units specification in the current design must be the same as the units specification in the control file otherwise the resistor symbols will not be generated.	3	3

Related Topics

- Control Directive Descriptions

Control Directive Descriptions

Wherever possible, control directive descriptions are placed on a single page. This may create areas of white space beneath certain descriptions.

allegro_dummy_padstack

Identifies which padstack to use as a default when generating shaped resistor pads.

```
dummy_padstack <padstack_name>
```

padstack_name is the name of a user-defined dummy padstack. The default is `pinres.pad`.

This dummy padstack must exist in the symbols directory identified in the `allegro_symbols_dir` directive.

Be sure that the dummy padstack corresponds to the technology (same CONDUCTOR subclass as the resistor pads). Otherwise false DRC errors may occur. The dummy padstack should be smaller than the smallest expected pad shape size.

⚠ If this dummy padstack does not exist, the Resistor Synthesizer creates a default padstack called `pinres.pad`.

allegro_pad_shaped

Indicates whether resistor pads are to be generated as filled rectangles inside the component symbol. These resistor pads refer to the dummy padstack.

```
allegro_pad_shaped < ON | OFF >
```

The default value is `OFF`, which means that the tool does not generate resistor pads as filled rectangles inside the component symbol. The Resistor Synthesizer looks for a padstack with a size that corresponds to the size of the pad of the particular resistor to be generated

```
(pin < size_x > _< size_y >

< mil | mic > .pad).
```

If this padstack does not exist in the symbols directory, it is generated from a script. A `.pad` file that reflects the necessary dimensions is created and referenced by the appropriate symbols.

The pin-area inside the padstack is generated on subclass TOP (for resistor_pads on top),

BOTTOM (for resistor_pads on bottom), or TOP and BOTTOM and every internal subclass (for other resistor_pads).

If you need to modify the completed resistor symbol after it is placed on the substrate, set this directive to `ON`. For pads as shapes, the symbol refers to the user-defined dummy padstack, regardless of whether padstacks with the appropriate size exist.

⚠ Be sure the dummy padstack contains a smaller pad size than the smallest expected pad size.

See the descriptions of the following directives that concern shaped pads:

```
allegro_dummy_padstack

allegro_symbols_dir

allegro_pad_subclass
```

allegro_pad_subclass

Indicates extra CONDUCTOR subclass data, such as soldermask or pastemask information, is required for post-processing the resistors.

```
allegro_pad_subclass <subclass_name> <value>
```

subclass_name	The newly generated CONDUCTOR subclass on which to put extra information
value	The size by which shaped resistor pads are expanded or decreased (positive or negative), and then laid out on the corresponding subclass
	The value is in microns unless a scale factor is used (specified in the `scale` directive).

⚠ This directive primarily concerns shaped resistor pads, since resistor symbols referencing `pin < size_x > _< size_y > < mil | mic > .pad` will never get this extra subclass data. This information is usually already part of the referenced padstack.

When automatically generating padstacks, information on SOLDERMASK or PASTEMASK or FILMMASK is created within the padstack (these predefined subclasses already exist). APD ignores other subclasses when padstacks are automatically generated.

The following examples are typical control file entries:

```
allegro_pad_subclass pastemask 5

allegro_pad_subclass soldermask -5
```

allegro_route_keepout_all

Generates a route keepout over the resistor ink for subclass ALL. By not enabling `allegro_route_keepout_all`, a route keepout is generated only for the subclasses specified through an `allegro_route_keepout_subclass` directive.

```
allegro_route_keepout_all < oversize_value >
```

`oversize_value` is the route keepout oversize amount in microns unless you use the `scale` directive.

⚠ Oversizing is done only in those directions where there is no overlap with conductor data from the padstack.

The default is to not generate any ROUTE KEEPOUT on subclass ALL.

⚠ Route keepout generation for subclass ALL is accepted only when no CONDUCTOR is generated under this route keepout. Otherwise DRC violations occur.

allegro_route_keepout_subclass

Specifies a subclass against which to generate ROUTE KEEPOUT. This information is necessary when generating resistor symbol script files to define a ROUTE KEEPOUT region on the resistor symbol for the appropriate subclass.

`allegro_route_keepout_subclass` *<pad_subclass>*

<oversize_value>

`pad_subclass`	The name of the subclass used for generating route keepout
`oversize_value`	The route keepout oversize amount in microns unless you use a scale control

⚠ Oversizing is only done in those directions where there is no overlap with conductor data from the padstack.

The `allegro_route_keepout_subclass` directive applies to the latest `resistor_base` directive.

The number of `allegro_route_keepout_subclass` directives is unlimited. Be sure to also include the top and bottom subclasses when you want route keepout for those. Refer to other tool directives for more information on controlling the generation of resistor scripts.

allegro_solutions

Generates alternate symbols containing different solutions for a particular resistor. The solutions may include geometries on other subclasses, other shapes or trim types, vertical and horizontal orientations, and even mirrored resistor symbols.

```
allegro_solutions < OPTIMAL | VALID | ALL >
```

The default value is `OPTIMAL`. By default, the Resistor Synthesizer generates one optimal solution. Selecting the VALID value causes the generation of all the possible solutions as alternate symbols. `VALID` means a layout on an ink that is part of the reduced set of inks after optimization. Selecting the `ALL` value creates all the solutions as alternate symbols, regardless of the ink on which they were found.

allegro_subclass

Specifies one subclass (ink) for all generated resistors.

```
allegro_subclass < force_subclass >
```

`force_layer` or `force_subclass` is the name of a layer/subclass described in a `resistor` directive. If not specified in the control file, all layers/ subclasses are considered to be valid and a solution with the smallest area is chosen.

⚠ This directive is required only in the case of thin resistors. The Resistor Synthesizer generates a layout only on this particular layer/subclass.

area_optimization

Indicates whether the layout of all resistors is to be produced with the smallest possible area.

```
area_optimization  <on|off>
```

The Resistor Synthesizer calculates all solutions that fit various criteria in the film resistor control file and chooses the solution that has the smallest area consumption (also taking into account the resistor pads). The default value is `on`.

correction_curve

Specifies correction curves for the resistivity of resistor subclasses.

```
correction_curve "[<R_width>]"
<R_length> <horizontal_correction> [<vertical_correction>]
<R_length> <horizontal_correction> [<vertical_correction>]]
end_curve
```

R_length and *R_width* is the actual length and width of the resistor and the correction_factor is used to multiply the Rnom specified in the resistor directive.

For example, if you have the following curve description:

- width less or = 60 mils
- length less or = 40 mils correction = 0.8
- length = 60 mils correction = 0.9
- length = 80 mils correction = 1.0
- length more than or = 100 mils correction = 1.1
- width = or more than 100 mils
- length less or = 40 mils correction = 0.85
- length = 60 mils correction = 0.95
- length = 80 mils correction = 1.05
- length more than or = 100 mils correction = 1.15

Then the following correction_curve directive lines would be used:

```
correction_curve 60
040 0.8
060 0.95
008 1.05
00 1.15
end_curve
```

The Resistor Synthesizer uses a double linear interpolation method to get the value of the correction factor.

The `correction_curve` directive applies to the latest `resistor` and `resistor_base` directives. If you do not specify a correction curve, the Resistor Synthesizer sets the correction factor to 1 (no correction).

⚠ The R_width parameter is optional. This is useful for a correction factor which is independent of the width of the resistor. The correction applies to rectangular resistors as well as to tophat resistors. Using the correction curves for top-hat resistors can be disabled through the `tophat_correction_curves` directive.

current_coef

Specifies the safety factor to be multiplied by the nominal current.

```
current_coef <current_coef_value>
```

The Resistor Synthesizer multiplies the nominal current requested for the resistor by `current_coef_value`.

current_units

Specifies the current density unit used in the `resistor` directive.

```
current_units < micron|cm|mm|mil|inch >
```

The default current density is in Amperes per micron.

For example

```
current_units mm
```

means that the current density given in the resistor control is in Ampere per millimeter.

default_base

Defines the `resistor_base` and the corresponding set of inks for the generation of the resistors.

```
default_base < default_base_name >
```

The `default_base` is a means of enforcing a certain resistor to a specific set of inks. The default value is `SUBSTRATE`.

A resistor_base defines a set of inks from which to choose when generating a hybrid resistor.

Normally the front of the substrate is the default resistor base with a set of inks and corresponding technology values attached. You can also define the back of the substrate as a different resistor base, containing the same or different inks with slightly modified technology values (modified because of the different firing times of the same pastes on the front or the back of the substrate).

default_backside_base_name

Defines the resistor_base and the corresponding set of inks for the generation of the resistors on the backside of the substrate (mirrored resistors).

```
default_backside_base < default_backside_ base_name >
```

The backside of the substrate is an additional resistor base, containing the same or different inks with slightly modified technology values (modified because of the different firing times of the same pastes on the front or the back of the substrate). Defining this base generates mirrored symbols and defines them as alternate for the bottom side.

The default is blank, indicating that no mirrored resistor symbols are to be generated.

⚠ This does not apply to Thin Resistors.

default_orientation

Indicates whether the Resistor Synthesizer is to generate resistors horizontally or vertically.

```
default_orientation < HORIZ | VERT >
```

The default value for orientation is blank, indicating that the horizontal as well as the vertical solution is being calculated. This directive is used for enforcing only horizontal or only vertical resistors.

⚠ When correction curves are defined, the vertical orientation does not correspond to only a 90-degree rotation. Because of different length/width ratios, different correction factors are applied, resulting in different resistor geometries.

default_trim

Sets the default trim type for the generated resistors that do not have the TRIM property attached.

```
default_trim < trim_type> [< trim_type>
```

Trim_type forces either single plunge cut (S-cut), L-cut, or dual plunge cut (D-cut).

Without a default_trim specification, the Resistor Synthesizer selects the trim type based on which trim type uses the smallest area, the smallest number of inks (see ink_optimization), and/or the best resolution.

⚠ You can specify one or two different trim types in this directive.

default_shape

Sets the default shape for the generated resistors that do not have the SHAPE property attached.

```
default_shape < RECT | TOPHAT >
```

Without a shape specification, the Resistor Synthesizer selects either `RECT` or `TOPHAT` based on which shape gives the solution with the smallest area, the smallest number of inks (see ink_optimization), and the best resolution.

delta_correction

Specifies the parameters used to compute the delta_tolrc value.

```
delta_correction < coef_t0> < coef_t1> < coef_t2> < coef_x>
```

where

```
delta_tolrc = coef_t0 + coef_t1 * R_val + coef_t2 * (R_val^coef_x)
```

dir or directory

Specifies the SCALD mapping file from which the used bodies are to be read.

```
dir < scald_file > [scald_file > ... ]
```

```
directory < scald_file > [scald_file > ... ]
```

The relative order in which you specify the `lib` or `library` and `dir` or `directory` directives in the film resistor control file is important. When the Resistor Synthesizer finds a `dir` or `directory` directive, it places all the cells in that library in front of the list of cells. When the Resistor Synthesizer finds a `lib` or `library` directive, it places the cells in that library at the end of the list of cells.

⚠ You must always specify an active directory.

extend_serpentine

Indicates whether to make a small extension of the first and last leg of a serpentine resistor if the resistor is not to be trimmed, so that the resistor's dimensions are adjusted to exactly meet the number of squares requirement.

```
extend_serpentine < ON | OFF >
```

The default is `ON`.

grid

Sets the grid value.

```
grid < grid_value >
```

`grid_value` is the internal grid spacing in microns unless you use the `scale` control directive. Use real numbers. Be sure the internal database unit is an integer.

The Resistor Synthesizer puts all resistor geometry on this grid. Be sure your grid is not too large as the grid value may influence the resistor dimensions. A grid that is too small requires time-consuming resistor computations.

For example, to put geometry on a grid of 25.4 microns, use following control:

```
scale 25.4  grid 1
```

The default values are: `scale = 1.0, grid_value = 1`.

```
060 0.9

080 1.0

100 1.1

end_curve

correction_curve 100

040 0.85
```

ink_optimization

Indicates whether as few inks as possible is to be used in the layout of all resistors.

```
ink_optimization < ON | OFF >
```

The Resistor Synthesizer starts calculating all solutions that fit the criteria specified in the film resistor control file. Then it chooses the inks with the highest use and selects the solution that has the smallest area (also taking into account the resistor pads).

The default value is `OFF`.

If you enable this control, the Resistor Synthesizer evaluates the list of possible solutions (taking into account user defined values, technology data and constraints) for each resistor. The Resistor Synthesizer retains only the most highly populated inks, which means that fewest inks as possible are used.

⚠ The resistor documentation (listing file, documentation cell, and resistor table file) contain each resistor's solution, which features the selected ink and smallest possible area. This control is only applicable to Thick Resistors.

lib or library

Specifies the additional libraries to be referenced by the Resistor Synthesizer during the resistor generation process. The Resistor Synthesizer searches the libraries specified in this directive for cells not found in the SCALD mapping files listed in the `dir` or `directory` directives.

```
lib < lib_name > [< lib_name > ... ]

library < lib_name > [< lib_name > ... ]
```

`lib_name` is the name of the SCALD directory that contains the library. The Resistor Synthesizer first searches the master directory to locate the library.

⚠ See the description of master_library directive for related information.

library_file

Specifies the libraries containing physical information for parts in the design. You can use the short version of a library file name with the `library` control.

```
library_file < chips_file_name > [< chips_file_name > ... ]
```

chips_file_name is the full path name of the library file.

master_library

Specifies the names of several master libraries. These files should contain the rooted path names of the libraries named in the library directive. The Resistor Synthesizer reads the system-wide master library file automatically.

```
master_library < masterlib_name > [< masterlib_name > ... ]
```

masterlib_name is the file or path name.

max_errors

This directive sets the maximum number of errors allowed before the Resistor Synthesizer stops execution. The default value is 500.

```
max_errors < max_errors_value >
```

max_length

Specifies the maximum length of generated rectangular resistors.

```
max_length < max_length_value >
```

max_length_value is the maximum length of the generated resistor in microns, unless you use the `scale` control.

The Resistor Synthesizer uses this value in the following equation:

```
R_length < = max_length_value
```

Solutions that do not satisfy this constraint are rejected.

⚠ If you do not specify this directive, the Resistor Synthesizer does not check maximum dimensions.

max_width

Specifies the maximum width of generated rectangular resistors.

```
max_width < max_width_value >
```

max_width_value is the maximum width of the generated resistor in microns, unless you use the scale directive.

The Resistor Synthesizer uses this value in the following equation:

```
R_width < = max_width_value >
```

Solutions that do not satisfy this constraint are rejected.

⚠ If you do not specify this directive, the Resistor Synthesizer does not check maximum dimensions.

min_length

Specifies the minimum length of generated rectangular resistors.

```
min_length < min_length_value >
```

min_length_value is the minimum length of the generated resistor in microns, unless you use the `scale` directive.

The Resistor Synthesizer uses this value in the following equation:

```
IR_length  > = min_length_value
```

⚠ The *min_length_value* must be on the grid, otherwise the Resistor Synthesizer issues an error message and stops processing.

min_width

Specifies the minimum width of generated rectangular resistors.

```
min_width < min_width_value >
```

min_width_value is the minimum width of the generated resistor in microns, unless you use a `scale` directive.

The Resistor Synthesizer uses this value in the following equation:

```
IR_length  > = min_width_value
```

⚠ The *min_width_value* must be on the grid, otherwise the Resistor Synthesizer issues an error message and stops processing.

number_of_squares

Specifies the minimum and maximum number of squares of the generated rectangular resistors.

```
number_of_squares <min_square> <max_square>
```

min_square and *max_square* are two real numbers that respectively fix the minimum and maximum values for the ratio.

For example:

```
0.33 < = R_length over R_width < = 5.0
```

The required number_of_squares directive:

```
number_of_squares 0.33  5.0
```

oversights

Indicates whether oversights are to be displayed.

```
oversights < ON | OFF >
```

The default value is *ON*.

part_table_file

Specifies the file containing the resistor component properties table.

```
part_table_file < part_table_name >
```

part_table_name is the path name of the component properties table.

This component properties table is automatically updated from the schematic and defines a list of possible alternate symbols for each symbol.

⚠ You can specify several names as values in one `PART_ TABLE_FILE` control, or you can use separate `PART_TABLE_FILE` controls to identify multiple component properties tables. For example, the following control specifies two component properties table files, `res.ptb` and `cap.ptb`:

```
PART_TABLE_FILE 'res.ptb', 'cap.ptb';
```

The control shown above is equivalent to the control:

```
PART_TABLE_FILE 'res.ptb';

PART_TABLE_FILE 'cap.ptb';
```

Packager-XL reads the table format definitions and finds the properties used to alter the component. If it finds any of these properties on an instance (body), it checks their values against the entries in the table. If an entry in the table for the given values on a component cannot be found, an error message is generated. You must either change the property values in the drawings or update the component properties tables.

power_coef

Specifies the safety factor to be multiplied by nominal power.

```
power_coef < power_coef_value >
```

The Resistor Synthesizer multiplies the nominal power requested for the resistor by *power_coef_value*. The default *power_coef_value* is `1.0`.

power_derating

Specifies the maximum power density that linearly decreases from a certain starting temperature.

```
power_derating < derating_factor > < derating_temp >
```

The Resistor Synthesizer uses *derating_factor* and *derating_temp* in the followings equations:

```
a. local_temp < = derating_temp

b. PDmax = Pden

c. local_temp  > derating_temp

d. PDmax = Pden - (local_temp -
```

where:

- `PD` max is the maximum power density that can be dissipated by the resistor.
- `P` den is the nominal power density as specified in the resistor directive.
- `local_temp` is the temperature of the resistor either specified with a TEMP property on the resistor or with the temp directive.

If the control file does not contain a `power_derating` directive, the Resistor Synthesizer cannot apply any derating.

⚠ The `derating_factor` must be provided in watt/degree.

power_units

Specifies the power density unit of measurement used in the `resistor` directive.

```
power_units < MICRON|CM|MM|MIL|INCH >
```

power_units MM means that the power density given in the `resistor` control is in Watt per square millimeter. The default power units are Watt per square millimeter.

resistor_base

Defines the base_name for the `resistor` and `resistor_pads` directives.

```
resistor_base < base_name >
```

The default base_name is `SUBSTRATE`.

⚠ See the `default_base` directive for additional information.

resistor

Describes all resistive layers/ subclasses in your design. This is required by the tool.

`resistor <` R_{lay} `> <` R_{nom} `> <` P_{den} `> <` C_{den} `> <` P_{tol} `> [<` V_{grad} `> [<` *TH_Pt* `> ]]`

`resistor <` R_{lay} `> <` R_{nom} `> <` P_{den} `> <` C_{den} `> <` P_{tol} `> [<` V_{grad} `> ]`

R_{lay}	The layout layer name or subclass name.

R_{nom}	The resistivity in ohms/sq inch of the layer or subclass
P_{den}	The maximum power density (for units, refer to the `power_units` directive)
C_{den}	The maximum current density (for units, refer to the `current_units` directive)
P_{tol}	The process tolerance given as a percentage
V_{grad}	The maximum voltage gradient (for units, refer to the `voltage_units` directive)
TH_Pt	The process tolerance used for tophat resistors and given as a percentage. When this tophat process tolerance is not specified, the same value as the normal process tolerance is taken.

RES1 is the name of a material of 10000 ohms/sq inch that can dissipate 50mW per square mm with a maximum current density of 1mA per 100 microns, a process tolerance of 25%, a voltage gradient of 400V per cm, and a tophat process tolerance of 20%. The required `resistor` control directive would be:

```
resistor res1 10000 50mW 1.0E-5 25 400 res1_id 20

resistor res1 10000 50mW 1.0E-5 25 400 res1_id
```

⚠ The process tolerance refers to a deviation in resistivity {in ohms/sq in} and not in the resulting number of squares.

The film resistor control file must include at least one resistor control, or the program issues an error message and stop execution.

⚠ The `resistor` directive applies to the latest `resistor_base` directive.

resistor_loops

Indicates whether to check for cyclic loops when measuring the resistors during the trimming process. The existence of such loops prevents exact measurement of each resistor, since the surrounding circuitry also influences the resistance value of that specific resistor. The Resistor Synthesizer generates warning messages and a list of the resistors for which the loop problem occurs.

```
resistor_loops < script_file >
```

script_file is the name of the Design Entry HDL script that contains the controls for highlighting

the existing loops (highlighting of resistor bodies plus corresponding nets). The default is disabled, which means cyclic loops are not checked.

⚠ Cadence recommends that you always include the `resistor_loops <script_ file>` directive in your control file to identify errors early in the process.

resistor_pads

Specifies the parameters for creating the two pads of the generated resistors.

```
resistor_pads < pad_layer > < pad_ovl > < pad_encl > [< pad_encl_w > ]
```

pad_layer	The name of the layer or subclass used for pads generation.
pad_ovl	The amount that the *pad_layer* must overlap the RIayer.
pad_encl	The distance between the edges of the RIayer and the edges of the *pad_layer*.
pad_encl_w	Optional (it defaults to *pad_encl*). A different enclosure value in the width direction.

The Resistor Synthesizer expects that you specify *pad_ovl*, *pad_encl*, and *pad_encl_w* in microns unless you used a `scale` directive. This is used with the `allegro_pad_shaped` directive. It generates a pad either as a shape on the chosen subclass or automatically creates a padstack of the correct size.

⚠ This `resistor_pads` directive applies to the latest `resistor_base` directive.

resistor_specs

Specifies the name of the resistor specification file that contains resistor information to be used instead of a Design Entry HDL schematic.

```
resistor_specs < resistor_specs_file >
```

resistor_specs_file is the name of the resistor specification file.

The Resistor Synthesizer assumes a Design Entry HDL schematic is being provided unless you include this `resistor_specs` directive in the film resistor control file.

⚠ You can also enter any other property that may influence resistor generation in the resistor specification file.

resistor_table

Specifies the name of a file where the resistor table information will be output.

```
resistor_table < resistor_table_file >
```

resistor_table_file is the name of the output file. This file is not generated unless this directive is included in the control file.

⚠ In addition to user-defined values for each generated resistor, this table contains calculated value, including chosen ink, width, and height.

resolution

Specifies the internal units used.

```
resolution < resolution_value >
```

resolution_value is a scale factor between the *user_units* and the internally used units for calculation purposes. The default value is equal to 100.

root_drawing

Specifies the name of the Design Entry HDL root drawing.

```
root_drawing < root_drawing_name >
```

The Resistor Synthesizer verifies that Packager-XL outputs match the name specified in this directive. The Resistor Synthesizer checks to see if Packager-XL `pst` files have a more recent date than the top design. If out-of-date files exist, the `pst` files do not exist, the Resistor Synthesizer re-executes Packager-XL before interpreting the `pst` files.

scale_factor_file

Specifies the name of a file that contains a set of scale factors different from the SPICE standard used in component properties tables.

```
scale_factor_file < scale_name >
```

scale_name is the name of the file that contains the redefined scales.

square_correction

Corrects the number of squares computed.

```
square_correction < coef_a > < coef_b >
```

coef_a and *coef_b* are used in the following equations:

$$n\_fix = R\_val / R_{max} - coef\_a / R\_width * R_{min} / R_{max} * (R\_val / R_{max} + coef\_b)$$

$$ncc = R\_val / R_{max} - coef\_a / width * R_{min} / R_{max} * (R\_val / R_{max} + coef\_b)$$

coef_a	Is in microns unless a scale control is used
coef_b	Is a real number without dimension

suppress

Prevents the display of warning and oversight messages.

```
suppress < suppress_number > [< suppress_number > ... ]
```

⚠ Error messages cannot be suppressed.

temp

Specifies the operating temperature of the resistor.

```
temp < temperature >
```

The Resistor Synthesizer uses this temperature to calculate maximum power density (see the power_derating directive).

⚠ You can specify the temperature for a resistor by adding the TEMP property to the resistor. Temperature values on the resistor override the value specified by this directive.

tophat_number_of_squares

Specifies the minimum and maximum number of squares of the generated tophat resistors.

```
tophat_number_of_squares < min_square > < max_square >
```

min_square and *max_square* are two real numbers that respectively fix the minimum and maximum values for the ratio:

$$\frac{R\_length}{R\_width}$$

⚠ For tophats, the maximum trimming is taken into account when calculating the number of squares. It is the number of squares through which the current flows.

Min_square	Determines when to start looking for a top hat solution (a normal value being 4 or 5). The minimum value of *min_square* depends on geometry constraints. User errors are reported.
Max_square	A measure of the allowed height of the tophat, compared to its width.

⚠ An overlap region in number_of_squares may exist, for which a rectangular and a tophat solution exist for a resistor on a specific ink. The resistor tolerance, shape, and area determine which solution is selected by the Resistor Synthesizer.

Tophat resistors are frequently used to reduce the number of inks. For example, allowing 20 squares as the maximum number of squares may result in generating a particular resistor on an ink with a smaller resistivity. This means you do not need to use a second ink.

tophat_max_width

Specifies the maximum width of generated tophat resistors.

```
tophat_max_width < max_width_value >
```

max_width_value is the maximum width of the generated resistor in microns, unless you use a `scale` directive.

The Resistor Synthesizer uses this value in the following equation:

```
tophat_R_width < = max_width_value
```

⚠ If you do not specify this directive, the Resistor Synthesizer does not check maximum dimensions.

This tophat maximum width value may differ from the rectangular maximum width value. You can specify a different value for rectangular maximum widths than the tophat maximum widths.

tophat_min_width

Specifies the minimum width of generated tophat resistors.

```
tophat_min_width < min_width_value >
```

min_width_value is the minimum width of the generated resistor in microns, unless you use a `scale` directive.

The Resistor Synthesizer uses this value in the following equation:

```
tophat_R_width  > = min_width_value
```

The *min_width_value* must be on the grid or the Resistor Synthesizer stops processing and issues an error message.

This tophat minimum width value may differ from the rectangular minimum width value. The paste end effects may have a lesser impact on the resistivity when tophat shapes are used. You can specify a different value for rectangular minimum widths than for tophat minimum widths.

tophat_leg_length

Specifies the actual length of the tophat resistor legs in relation to its width.

```
tophat_leg_length < leg_length_value >
```

leg_length_value is the percentage of the width of the generated tophat resistor.

The Resistor Synthesizer uses the *leg_length_value* in the following equation:

```
tophat_res_leg_length = leg_length_value * R_width
```

⚠ The *leg_length_value* should be defined so that its multiplication with the width gives a dimension on grid. Otherwise the dimension is forced to the nearest grid.

This directive is optional. If you do not specify this directive, the synthesizer chooses leg lengths

that fit various criteria specified in the film resistor control file.

tophat_correction_curves

Indicates whether correction curves are to be used in tophat resistor geometry calculation. When disabled, the Resistor Synthesizer does not correct any resistivity due to the resistor dimensions.

```
tophat_correction_curves < ON | OFF >
```

When this directive is enabled, the Resistor Synthesizer considers the correction curves for each ink when establishing the dimensions of a tophat resistor on that ink. The correction curves contain the correction factor on the resistivity for a specific length and width of a resistor on a certain ink.

The default for `tophat_correction_curves` is `ON`.

⚠ For a tophat resistor, this width is the width of the base component. The length is the length of the unfolded tophat resistor with maximum trimming and is equal to the maximum number of squares of the tophat times its width.

trimming

Specifies trimming information.

```
trimming < t_layer > < t_width > < t_step > < t_space > < t_min >
```

t_layer	The layer/subclass where the trimming information will be written
t_width	The width of the trimming tool
t_step	The minimum step displacement of the trimming tool
t_space	The minimum spacing of the laser cut with the pads
t_min	The minimum width of the layer/ subclass after trimming

⚠ *t_min* is a percentage of the total width of the resistor. You should specify *t_width*, *t_step*, and *t_space* in microns unless you use the `scale` directive.

The `t_space` value is also used in the following equation:

```
res_length = ncc * Rmax / Rmin * width + laser_space
```

```
R_width  > = width * Rmax / Rmin + ( laser_space / ncc )
```

This directive is required. If it is not specified, the Resistor Synthesizer issues an error message and stops processing.

trim_check

Indicates whether checking for trimming is to be suppressed.

```
trim_check < ON | OFF >
```

If enabled, the geometry of the resistors is also adapted (enlarged) to include trimmability constraints. The default is `OFF`.

⚠ If you use this directive, a tolerance property must be attached to the resistors.

units

Specifies the units that are used in the film res control file.

```
units <MICRON|CM|MM|MIL|INCH >
```

The units used in the current design and the film res control file must be the same, otherwise the resistor symbols are not generated.

use

Specifies an alternate name for `dir` or `directory`. See the description of the `dir` directive for more information.

```
use < scald_dir > [< scald_dir > ... ]
```

voltage_units

Specifies the voltage gradient unit used in the `resistor` control.

```
voltage_units <MICRON|CM|MM|MIL|INCH >
```

`voltage_units CM` means that the voltage gradient specified in the resistor control is in Volts per centimeter. The default value assumes that the voltage gradient is in Volts per centimeter.

warnings

Determines whether warning messages are displayed.

```
warnings < ON | OFF >
```

The default is `ON` for warnings.

wider_serpentine

Indicates whether to increase the serpentine width to enhance trimmability.

```
wider_serpentine < ON | OFF >
```

The default is `OFF`. The Resistor Synthesizer generates an error message when trimmability checks cannot be met for the smallest possible serpentine width.

Related Topics

- Running the Thick/Thin Film Resistor Synthesizer
- Setting Resistor Generation Controls

Film Resistor Error Handling

In this section, you will find:

- A list of errors that might appear in the `film_res.log` file
- A table listing the obsolete commands due to the change from using `film_res.cmd` to the `film res` command

Error Messages

The `film res` command directive does not run unless a valid `f`ilm_res.rcf control file is present.

Any errors in the `film_res.rcf` file are found after you click the *Run* button on the Thick/Thin Film Resistor Generator Controls dialog box. All error, warning, and status information is recorded in the `film_res.log` file, which you can view by running the `viewlog` command.

If there are errors in the control file the following message displays in the command window:

```
Errors in generation. Please see film_res.log file for details.
```

The following table lists possible error and warning messages in the `film_res.rcf` file.

Message	Reason	Solution
`Error (200) Mandatory command(s) not specified in command file. The following command(s) should be added to your command file: UNITS command.`	The `film_res.rcf` control file does not include the UNITS command.	Add the UNITS command to the `film_res`.rcf control file.
`Drawing type must be Layout for Resistor Generation. Change drawing type and try again.`	You can only run `film res` when you are editing a layout. The file should have been created using `new` .	Change the drawing type to *Layout*.
`Component <`*component name*`> has been edited. Cannot generate symbol <`*symbol name*`>. Please delete instances with padeditdb and run film_res again.`	When you edit a previously generated resistor symbol to shave/change its pad boundaries, `film res` ignores the symbol.	Delete instances of the component/symbol and run the `film res` command again.

`Symbol <`*symbol name*`> at <x,y> has been edited. Cannot generate symbol <`*symbol name*`>. Please delete instances with padedits and run` *film_res* `again.`	When you edit a previously generated resistor symbol to shave/change its pad boundaries, `film res` ignores the symbol.	Delete instances of the component/symbol and run the `film res` command again.
`Warning 273 Obsolete command used in resistor control file....`	You are using a resistor control file that you used with a previous version of the tool.	Remove the obsolete command from the resistor control file.

Obsolete Commands

The following table lists the warning messages for obsolete commands.

Obsolete Command	**Warning Message**	**Solution**
`allegro_doc_symbol`	The command ‘ `ALLEGRO _DOC_SYMBOL` ’ is obsolete, ignoring. Documentation symbol is no longer generated. See the log file and resistor table files for the resistor details.	Remove the command from the control file. Read details in the `film_res.log` file.
`allegro_execution`	The command ‘ `ALLEGRO _EXECUTION` ’ is obsolete, ignoring. Resistors are generated on the fly and there is no need to specify this.	Remove the command from the control file.
`allegro_number_of_colors`	The command ‘ `ALLEGRO _NUMBER_OF_COLORS` ’ is obsolete, ignoring.	Remove the command from the control file.
	The command ‘ `ALLEGRO _OVERWRITE_CONFIRM` ’ is obsolete, ignoring. Resistors are generated on the fly and they overwrite the existing symbols. In case you need otherwise, please run `film_res` on an empty design. Then use ‘ `dump libraries` ’ and ‘ `redraw symbols` ’ to redraw symbols selectively.	Remove the command from the control file.

allegro_script	The command ‘ ALLEGRO _SCRIPT’ is obsolete, ignoring. Resistors are generated on the fly and Scripts are not used.	Remove the command from the control file.
	The command ‘ ALLEGRO _SUBSTRATE_OUTLINE ’ is obsolete, ignoring. There is no need to specify the substrate outline. film_res uses current design extents for resistor symbols.	Remove the command from the control file.
allegro_symbols_ dir	The command ‘ ALLEGRO _SYMBOLS_DIR ’ is obsolete, ignoring. Resistor symbols are generated as part of the current design. You can use ‘ DumpLibrary ’ command to save these symbols to disk.	Remove the command from the control file.
allegro_text_size	The command ‘ ALLEGRO _TEXT_SIZE ’ is obsolete, ignoring. Text size is no longer read from the control file. You can choose appropriate text block from the form.	Remove the command from the control file. Select the text size on the Thick/Thin Film Resistor Generator Controls form.
scale	The command ‘ SCALE ’ is obsolete, ignoring. Use the new command ‘ UNITS ’ to specify the units/scale	Remove the command from the control file. You must include the ‘UNITS’ command in the resistor control file.

Related Topics

- Control File Directives for Film Resistors

Properties for Resistor Generation

ASPECT

This property specifies the width/height ratio of a serpentine resistor.

```
ASPECT < aspect_value > (instance)
```

aspect_value enforces a width/height ratio of the serpentine resistor.

You can attach the ASPECT property to any resistor. If you do not use this property, the `thick` program tries an aspect ratio of 1 (min_width and min_length is considered).

⚠ The ASPECT property can only be put on an instance.

CURRENT_COEF

This property specifies a safety current coefficient.

```
CURRENT_COEF < cc_value > (instance, library)
```

cc_value is used by the Resistor Synthesizer to multiply the local current value.

You can attach the CURRENT_COEF property to any resistor. If you do not use this property, the `thick` program sets the cc_value to `current_coef_value` specified in the `current_coef` directive.

⚠ You can specify this property on an instance of a resistor or in the library description of the component.

GENERATE

This property prevents the generation of this resistor. In this case, the program considers the resistor as an add-on component. If the resistor is not to be created with the Resistor Synthesizer, you must describe this component in a physical component table (PACK_TYPE or JEDEC_TYPE information).

```
GENERATE TRUE | FALSE ( instance )
```

Set this property to `FALSE` and the Resistor Synthesizer does not generate the resistor.

You can attach the GENERATE property to any resistor. If you do not use this property, the Resistor Synthesizer generates the resistor.

⚠ The GENERATE property can only be put on an instance.

I or CURRENT

This property specifies the current required by the resistor.

⚠ Attach this property to all resistors in the design.

```
I < current_value > ( instance )

CURRENT < current_value > ( instance )
```

current_value is the current in Amperes that the resistor must be able to draw. The *current_value* can be an integer, a floating point, an integer or floating point number followed by an integer exponent (for example, 1.0E3), or either an integer or floating point number followed by one of the following scale factors:

```
T = 1E12

G = 1E9

MEG = 1E6

K = 1E3

M = 1E-3

U = 1E-6
```

The Resistor Synthesizer ignores letters immediately following a number that are not scale factors. It also ignores letters immediately following a scale factor. For example, each of the following samples represent the same value:

- 0.001
- 0.001Amperes
- 1mA
- 1E-3

All resistors must have an I or CURRENT property. If a resistor does not have this property, the `thin` program issues an error message.

⚠ The I or CURRENT property can only be put on an instance.

JEDEC_TYPE

This property specifies which component symbol is used during design layout.

```
JEDEC_TYPE < jedec_type_name > ( instance )
```

You can attach the JEDEC_TYPE property to any resistor. If you do not use this property, a layout or component symbol with the same name as the body is expected to exist, unless you define a *jed ec_type value* in the component properties tables.

⚠ The JEDEC_TYPE property can only be put on an instance.

The JEDEC_TYPE property is typically used for assigning a symbol from a library to a body. The Resistor Synthesizer interprets the JEDEC_TYPE property the same way as the PACK_TYPE property.

SUBCLASS

This property specifies the subclass on which the Resistor Synthesizer will generate the resistor.

```
SUBCLASS < subclass_name > ( instance )
```

subclass_name is the name of a material described in a *resistor* directive. If the program cannot find a description for the subclass, an error message is issued.

⚠ The Resistor Synthesizer uses the specified subclass_name even if the generated resistor violates the min_square you have specified in the `number_of_squares` command. In this case the Resistor Synthesizer issues a warning message.

You can attach the SUBCLASS property to any resistor. If you do not use this property, the Resistor Synthesizer selects the subclass that optimizes the area of the resistor.

⚠ The SUBCLASS property can only be put on an instance.

LOCATION

This property specifies the location value to use when generating the resistor or add-on component. The location value is enclosed in parenthesis, and concatenated to the body name. This becomes the new cell-name. This location enforcement uniquely identifies a resistor, capacitor, or other component.

```
LOCATION < location_name > ( instance )
```

location_name is the name to use for special component identification.

You can attach the LOCATION property to any component. If you do not use this property, the Packager assigns a soft_location to each component, which is not guaranteed to stay the same value between different Packager executions.

⚠ The LOCATION property can only be put on an instance.

MAX_LENGTH

This property allows you to control the maximum length of rectangular resistors generated by the Resistor Synthesizer.

```
MAX_LENGTH < max_length_value > ( instance,library )
```

max_length_value is the maximum length of the generated resistor. The value is given in microns unless you use a `scale` directive.

You can attach the MAX_LENGTH property to any resistor. If you do not use this property, the maximum length for the resistor is set to max_length_value specified in the *max_length* directive in the command file.

If no maximum length is specified, checking against maximum dimensions is not performed.

⚠ You can specify the MAX_LENGTH property on an instance of a resistor or in the library description of the component.

MAX_WIDTH

This property allows you to control the maximum width of rectangular resistors generated by the Resistor Synthesizer.

```
MAX_WIDTH < max_width_value > ( instance, library )
```

max_width_value is the maximum width of the generated resistor. The value is given in microns unless you use a `scale` directive.

The MAX_WIDTH property can be attached to any resistor. If this property is not specified, the maximum width for the resistor is set to the value specified in the *max_width* directive in the command file.

If no maximum width is specified, checking against maximum dimensions is not performed.

⚠ You can specify the MAX_WIDTH property on an instance of a resistor or in the library description of the component.

MIN_LENGTH

This property allows you to control the minimum length of rectangular resistors generated by the Resistor Synthesizer.

```
MIN_LENGTH < min_length_value > ( instance, library )
```

min_length_value is the minimum length of the generated resistor. The value is given in microns unless you use a scale directive.

You can attach the MIN_LENGTH property to any resistor. If you do not use this property, the minimum length for the resistor is set to min_length_value specified in the `min_length` directive in the command file.

⚠ You can specify the MIN_LENGTH property on an instance of a resistor or in the library description of the component.

MIN_WIDTH

This property allows you to control the minimum width of rectangular resistors generated by the Resistor Synthesizer.

```
MIN_WIDTH < min_width_value > ( instance, library )
```

min_width_value is the minimum width of the generated resistor. The value is given in microns unless you use a scale directive.

You can attach the MIN_WIDTH property to any resistor. If this property is not specified, the minimum width for the resistor is set to the value specified in the `min_width` directive in the command file.

⚠ You can specify the MIN_WIDTH property on an instance of a resistor or in the library description of the component.

ORIENTATION

This property specifies the orientation of a generated resistor.

```
ORIENTATION HORIZ | VERT ( instance, library )
```

If you specify `HORIZ`, the Resistor Synthesizer generates a horizontal resistor. If you specify `VERT`, the Resistor Synthesizer generates a properly rotated resistor. If you specify an invalid value, the Resistor Synthesizer ignores this property and issues an error message.

You can attach the ORIENTATION property to any resistor. If you do not use this property, the Resistor Synthesizer generates either a horizontal or vertical resistor, depending on which one gives the best results.

When correction_curves are not introduced, a vertical layout coincides with a 90-degree rotation and vertical solutions and alternate symbols are not necessary.

⚠ You can specify the ORIENTATION property on an instance of a resistor or in the library description of the component.

P or POWER or MAX_POWER_DISS

These properties specify the power that must be dissipated by the resistor.

```
P < power_value > ( instance )

POWER < power_value > ( instance )

MAX_POWER_DISS < power_value > ( instance )
```

power_value is the power in watt that the resistor must be able to dissipate. The *power_value* can be an integer, a floating point, an integer or floating point number followed by an integer exponent (for example, 1.0E3), or either an integer or floating point number followed by one of the following scale factors:

```
T = 1E12

G = 1E9

MEG = 1E6

K = 1E3

M = 1E-3

U = 1E-6
```

The Resistor Synthesizer ignores letters immediately following a number that are not scale factors. It also ignores letters immediately following a scale factor. For example, each of the following samples represent the same value:

- 0.001
- 0.001Watt
- 1mW
- 1E-3

All film resistors must have a P or POWER or MAX_ POWER_DISS property. If a resistor does not have this property, the program issues an error message.

⚠ The P, POWER, and MAX_POWER_DISS properties can only be put on an instance.

POWER_COEF

This property allows you to specify a safety power coefficient.

```
POWER_COEF < pc_value > (instance, library)
```

pc_value is used by the Resistor Synthesizer to multiply the local power value.

You can attach the POWER_COEF property to any resistor. If you do not use this property, the Resistor Synthesizer sets the *pc_value* to the *power_coef_value* specified by the *power_coef* directive in the command file.

⚠ You can specify the POWER_COEF property on an instance of a resistor or in the library description of the component.

R or VALUE

You must attach this property to all resistors in the design to specify their nominal value.

```
R < R_value > ( instance )

VALUE < R_value > ( instance )
```

R_value is the nominal resistor value in Ohms. The R_value can be an integer, a floating point, an integer or floating point number followed by an integer exponent (for example, 1.0E3), or either an integer or floating point number followed by one of the following scale factors:

```
T = 1E12

G = 1E9

MEG = 1E6

K = 1E3

M = 1E-3

U = 1E-6
```

The Resistor Synthesizer ignores letters immediately following a number that are not scale factors. It also ignores letters immediately following a scale factor. For example, each of the following items represent the same value:

- 1000
- 1000ohms
- 1Kohms
- 1.0E3

All resistors must have an R or VALUE property attached to them. If a resistor does not have this property then the program issues an error message.

⚠ The R and VALUE properties can only be put on an instance.

RES_TYPE

This property allows you to prevent the generation of a resistor during processing. When the Resistor Synthesizer detects a RES_TYPE property, it considers the resistor as an add-on component. If the resistor is not to be created, you must describe the component in a physical component table (pack_type or jedec_type information).

```
RES_TYPE THICK | THIN | IC ( instance )
```

THICK	Is the default if no property is specified, ensures that the resistor is generated as a film resistor during processing
THIN	Marks the resistor as only to be generated in thin-film technology
IC	Specifies that the Resistor Synthesizer does not generate the resistor

⚠ You can only put the RES_TYPE property on an instance or on the drawing of the top design, meaning that all resistors not specifically labeled by this property inherit the value of the drawing property.

SHAPE

This property specifies the shape of a generated resistor.

```
SHAPE RECT | TOPHAT ( instance, library )
```

`RECT` indicates that the Resistor Synthesizer generates a rectangular resistor. The `TOPHAT` option generates a tophat resistor.

If you provide an invalid value, the Resistor Synthesizer ignores this property and issues an error message.

You can attach the SHAPE property to any resistor. If you do not use this property, the Resistor Synthesizer generates either a rectangular resistor or a tophat, depending on which produces the best results.

⚠ You can specify the SHAPE property on an instance of a resistor or in the library description of the component.

TEMP

This property specifies a local temperature for the resistor.

```
TEMP    < temperature > ( instance )
```

temperature is the local temperature of the resistor in degrees. The temperature can be an integer, a floating point, an integer or floating point number followed by an integer exponent (for example, 1.0E3), or either an integer or floating point number followed by one of the following scale factors:

```
T = 1E12

G = 1E9

MEG = 1E6

K = 1E3

M = 1E-3

U = 1E-6
```

The Resistor Synthesizer ignores letters immediately following a number that are not scale factors. It also ignores letters immediately following a scale factor. For example, each of the following items represents the same value:

- 100
- 100Degree
- 0.1KDeg
- 1.0E2

You can attach the TEMP property to any resistor. If you do not specify this property, the Resistor Synthesizer uses the global temperature specified by the temp directive in the command file.

⚠ The TEMP property can only be put on an instance.

T or TOL

This property allows you to specify the tolerance for the resistor.

```
T   < tolerance > ( instance )

TOL   < tolerance > ( instance )
```

tolerance is the final tolerance required for the resistor. The tolerance can be an integer, a floating point, an integer or floating point number followed by an integer exponent (for example, 1.0E3), or either an integer or floating point number followed by one of the following scale factors:

```
T = 1E12

G = 1E9

MEG = 1E6

K = 1E3

M = 1E-3

U = 1E-6
```

The Resistor Synthesizer ignores letters immediately following a number that are not scale factors. It also ignores letters immediately following a scale factor. For example, each of the following items represents the same value:

- 10
- 10%

- 0.01K%
- 1.0E1

You can attach the T or TOL property to any resistor, unless you use a `trim_check` directive in the command file. If you do not specify this property, the Resistor Synthesizer sets the tolerance to 0 and no trimmability check is performed.

⚠ The T or TOL property can only be put on an instance.

TOPHAT_MAX_WIDTH

This property allows you to control the maximum width of tophat resistors generated by the Resistor Synthesizer.

```
TOPHAT_MAX_WIDTH   < max_width_value > ( instance, library )
```

max_width_value is the maximum width of the generated tophat resistor. The value is given in microns unless you use a scale directive.

This property does not force the resistor to be laid out as a tophat. (Refer to the SHAPE property.)

The TOPHAT_MAX_WIDTH property can be attached to any resistor. If this property is not specified, the maximum width for the resistor is set to the value specified in the `tophat_max_width` directive in the command file.

If no value is specified (default), maximum dimensions are not checked. This property does not force the resistor to be layout out as a tophat.

⚠ You can specify the TOPHAT_MAX_WIDTH property on an instance of a resistor or in the library description of the component.

TOPHAT_MIN_WIDTH

This property allows you to control the minimum width of tophat resistors generated by the Resistor Synthesizer.

```
TOPHAT_MIN_WIDTH   < min_width_value > ( instance, library )
```

min_width_value is the minimum width of the generated tophat resistor. The value is given in microns unless you use a scale directive.

This property does not force the resistor to be laid out as a tophat. (Refer to the SHAPE property.)

The TOPHAT_MIN_WIDTH property can be attached to any resistor. If this property is not specified, the minimum width for the resistor is set to the value specified by the `tophat_min_width` directive in the command file.

⚠ You can specify the TOPHAT_MIN_WIDTH property on an instance of a resistor or in the library description of the component.

TOPHAT_MIN_VALUE

This property allows you to control the minimum value of tophat resistors generated by the Resistor Synthesizer.

`TOPHAT_MIN_VALUE < min_value > ( instance, library )`

min_value is the minimum resistance value of the generated tophat resistor. It is imperative that this minimum value can be reached with little or no trimming, even in the case of maximum resistivity.

Without trimming, the tophat part of the resistor decreases the overall resistance. Just neglecting this tophat part does not harm the less-than criterion. For the remaining rectangular part, it is dimensioned in such a way that the design number of squares meets the minimum resistance value, resulting in a maximum length of the legs of the tophat resistor.

This property does not force the resistor to be laid out as a tophat. (Refer to the SHAPE property.)

The TOPHAT_MIN_VALUE property can be attached to any resistor. If this property is not specified, other criteria (for example, tophat_leg_length) are taken into account for dimensioning the legs of the resistor.

The TOPHAT_MIN_VALUE property has precedence over the TOPHAT_LEG_LENGTH property.

⚠ You can specify the TOPHAT_MIN_VALUE property on an instance of a resistor or in the library description of the component.

TOPHAT_LEG_LENGTH

This property allows you to control the length of the legs of tophat resistors generated by the Resistor Synthesizer.

`TOPHAT_LEG_LENGTH < leg_length_value > ( instance, library )`

leg_length_value is the length of the legs of the generated tophat resistor as a percentage of its width.

The TOPHAT_LEG_LENGTH property can be attached to any resistor. If this property is not specified, the *leg_ length_value* is set to the value specified by the `tophat_leg_length` directive in the command file.

This property does not force the resistor to be laid out as a tophat. (Refer to the SHAPE property.)

⚠ You can specify the TOPHAT_LEG_LENGTH property on an instance of a resistor or in the library description of the component.

TRIM

This property specifies the type of trimming for a resistor.

```
TRIM S_CUT | L_CUT | D_CUT ( instance, library )

TRIM S_CUT | L_CUT   ( instance, library )
```

S_CUT	Causes the Resistor Synthesizer to generate a resistor with a single-plunge laser cut (serpentine shape)
L_CUT	Causes the Resistor Synthesizer to generate a resistor with an L-shaped laser cut
D_CUT	Causes the Resistor Synthesizer to generate a resistor with a dual plunge laser cut

The Resistor Synthesizer ignores this property if you give an invalid value and it also generates an error message.

You can attach the TRIM property to any resistor. If you do not use this property, the Resistor Synthesizer generates a resistor with default_trim information (if this directive is specified in the command file) or looks at all trim-types, and selects the one with the smallest area. The Resistor Synthesizer first tries to generate a serpentine resistor and if this is not possible (because of insufficient length), it generates a rectangular resistor design.

The TRIM property only relates to rectangular resistors. When the tophat shape is chosen, a single plunge cut is performed, regardless of the value of this property.

⚠ You can specify the TRIM property on an instance of a resistor or in the library description of the component.

Related Topics

- Setting Resistor Generation Controls

16

APD: Die Text File Format Specification

The APD die text format was designed as a flexible, spreadsheet-type format for specifying the die pad placement information necessary for representing an IC component inside a package design. Its primary objectives are readability and easy editing inside a spreadsheet tool such as Microsoft Excel. It is composed of 4 optional and one mandatory section:

- Pin Definition (Required)
- Header Section (Optional)
- Padstack Definitions (Optional, used to describe size and shape of pads for pins)
- Shape Definitions (Optional, used to describe additional geometries like pre-shrink outlines that may be desirable context for the package designer).
- Hierarchical Grid Specification (Optional, written by APD only, not user editable).

Additionally, the format supports comment lines identified by a leading `#` line.

Header Section

The header information stores information about the die, such as the size (given as width+height or extents) and definition name for the component. All of this information is optional and, if not present, will be queried during the import process.

Following is a sample of a header section:

```
File: E:/RefFlowData/DIE_data.txt

Date: Wed Sep 28 17:34:06 2005

Units: microns, 2 decimal places

Name: UNNAMED_DIE

DEF Design: DIE

RefDes: DIE
```

```
DieType: FlipChip

DieOrient: ChipDown

Origin: (0.00 0.00)

Rotation: 0.000

Extents: ((-5000.00 -5000.00) (5000.00 5000.00))
```

The header section contains the following elements:

- *File*: Lists the full path of the original file. This is only for information.
- *Date*: Contains the date on which this file was last edited. This is only for information.
- *Units*: Lists the design units (`microns`, `mils`, and so on) and accuracy (example, `2 decimal places`) for all measurement and placement data contained in the file.
- *DEF Design*: Lists the original, case-sensitive IC design name for this component.
- *Name*: Lists the default package symbol name to use when importing into APD.
- *DieType*: Specifies the attachment type by which this die connects to the package. Currently, two types are supported, `Wirebond` and `FlipChip`.
- *DieOrient*: Specifies the default mounting orientation for an instance of this component. Maybe be one of `ChipUp` or `ChipDown`.
- *Size*: Specifies the size of the IC component, given as width and height measurements.
- *Extents*: Specifies the extents of the die boundary, given as a list of lower-left and upper-right extents. If present, overrides any value set for *Size*.
- *RefDes*: Specifies the default instance reference designator to use when adding this die to APD.
- *Origin*: Specifies the default instance placement origin for this component.
- *Rotation*: Specifies the default rotation for a placed instance of this component.
- *Pad Layer*: Specifies the layer on which pads should be placed by default for this component in the layout database.
- *DieAdhesive*: Information about the adhesive material used (primarily for wire bond dies) in mounting of the component to the substrate or element below it in the stack.

Pin Definition Section

The pin definition section lists all die pad locations and logical data required by the package tool. Each die pad is listed in a separate line of the die text file, with each piece of information separated from the last by a column separation character, typically one or more spaces or tab characters, but you can select any unique character.

The pin definition section is preceded by an (optional) information line which defines the order of the columns and the data contained in each. If the information line is not provided, you user must identify the columns during import. Following is an example of an information line:

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

The available columns of information which can appear in this section are:

`Pin Number`	◦ Required ◦ Blank entries not allowed.	The physical pin number for this pin. This value must be unique.
`X Coord`	◦ Required ◦ Blank entries not allowed.	The X-coordinate value of the physical pin relative to the origin of the die symbol definition.
`Y Coord`	◦ Required ◦ Blank entries not allowed.	The Y-coordinate value of the physical pin relative to the origin of the die symbol definition.
`Rotation`	◦ Optional ◦ Blank entries not allowed ◦ Default of 0.00 degrees if not supplied.	The rotation of the physical pin around its X,Y coordinate location in the definition.

`Pin Use`	◦ Optional ◦ Blank entries allowed. ◦ Default of Bidirectional if not supplied.	This is the logical pin use of this pin, and may be one of Power, Ground, Bidirectional, Tristate, In, Out, No-Connect, Unspecified, OCA, or OCL.
`Swap Code`	◦ Optional ◦ Blank entries allowed. ◦ Default matches pin use.	The swap code is used to define groups of pins which may have their net assignments swapped to improve routability. By default, pins are grouped by their pin use (ie power pins can swap amongst themselves, bidirectional pins amongst themselves, etc).
`Padstack`	◦ Optional ◦ Blank entries allowed. ◦ User will be prompted to define or select a padstack to use for all pins without a set padstack name.	The pad definition (defines pad size, layer, and shape for the pin).

`Net Name`	◦ Optional ◦ Blank entries allowed. ◦ Pin placed on a dummy net (unassigned) if no column or blank entry in this column	The net to which this pin is connected.
`RefDes`	◦ Optional ◦ Blank entries allowed. ◦ Must be paired with a package pin column	The instance reference designator of the package component to which this die pin is associated (connected to the same net). Paired with the package pin column, this provides the full description of the package pin.
`Package Pin`	◦ Optional ◦ Blank entries allowed only if refdes column is blank on same row. ◦ Must be paired with a package pin column	The physical pin number of the package pin to which this die pin is associated (connected to the same net). Paired with the refdes column, this provides the full description of the package pin.

`Pin Prop Name`	◦ Optional ◦ Blank entries allowed. ◦ Property not added if column is blank. ◦ Multiple columns of this type are permitted. ◦ Must be paired to a pin prop value column.	The name of a property which should be attached to this physical pin instance. This may be a user-defined property or a system property. The next column in the file defines the value for this property if not a boolean value property.
`Pin Prop Value`	◦ Optional ◦ Blank entries allowed. ◦ Multiple columns of this type are permitted. ◦ Must be paired to a pin prop name column.	The value of a property which should be attached to this physical pin instance. This column is blank for Boolean properties.

`Net Prop Name`	◦ Optional ◦ Blank entries allowed. ◦ Property not added if column is blank. ◦ Multiple columns of this type are permitted. ◦ Must be paired to a net prop value column.	The name of a property which should be attached to the net specified for this pin. This may be a user-defined property or a system property. The next column in the file defines the value for this property if not a boolean value property.
`Net Prop Value`	◦ Optional ◦ Blank entries allowed. ◦ Multiple columns of this type are permitted. ◦ Must be paired to a net prop name column.	The value of a property which should be attached to this net. This column is blank for boolean properties.

An example series of lines in the pin definition section which match the information line provided earlier would be:

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

1 -4700.00 4700.00 0.000 BI

2 -4600.00 4700.00 0.000 BI

3 -4500.00 4700.00 0.000 POWER VDD

4 -4400.00 4700.00 0.000 BI
```

```
5 -4300.00 4700.00 0.000 BI

6 -4200.00 4700.00 0.000 GROUND VSS
```

Padstack Definitions Section

This optional section of the file allows you to directly specify the size and shape of padstack pads referenced by the pins section in the file. If the padstack is already defined in the destination layout database, that will be used. This allows you to control the pad geometries independent of any library pad definitions which may conflict with the name of padstacks referenced in the file.

Any number of padstacks can be defined for the die, each with its own shape and size. Padstacks are listed one per file line, using keyword:value pairings separated by | characters.

The definition attributes for a padstack are:

- *PADSTACK*: Identifies this as a line defining a shape.
- *Name:<string>* -- The name of the padstack. Should be referenced in the padstack column of the pins section.
- *Shape:<string>* -- The shape of the padstack's pad (SQUARE, RECTANGLE, OCTAGON, CIRCLE, etc). Any of the pad primitive names supported are allowed.
- *Width:<DB value>* -- The width dimension of the pad shape.
- *Height:<DB value>* -- The height dimension of the pad shape. For symmetric shapes, like square and circle pads, this should match the width.

The following section defines padstack information for pins of this symbol. Pad layer will be determined at the time of import.

```
Begin padstacks:

PADSTACK: NAME:DIE_PAD|SHAPE:SQUARE|WIDTH:70.000|HEIGHT:70.000

End padstacks.
```

Shape Definitions Section

This optional section defines additional shape objects that are a part of the physical symbol definition. These may be reference elements, keep-out areas, etc. Shape definitions go on specific layers identified in the lines in this section; those layers, if they do not already exist in the design, will be added automatically.

Any number of shapes can be defined for the die, each with its own extents and layer mapping. Shapes are listed one per file line, using keyword:value pairings separated by | characters.

The definition attributes for a shape are:

- SHAPE: Identifies this as a line defining a shape.
- Type:<string> -- The type of shape definition (currently supports only rectangle).
- Class:<string> -- The class name on which the shape is to be placed.
- Subclass:<string> --The subclass name on which the shape is to be placed. If the layer does not exist in the design yet, it will automatically be created for you.
- Extents: <2-point list> -- The lower left and upper right coordinates defining this rectangle in the symbol's definition.

An example of this section for a die with a single additional reference shape would be:

```
Begin shapes:

SHAPE: TYPE:RECTANGLE|CLASS:COMPONENT GEOMETRY|SUBCLASS:DEMO|EXTENTS:((100.00 100.00)
(200.00 200.00))

End shapes.
```

Hierarchical Grid Specification Section

The grid specification is private to APD, and defines the details of the hierarchical grid structures to which the pins in the pin definition section are snapped. These are used by the APD symbol editor environment for snapping pins, automatically maintaining physical pin numbering patterns during pin pattern manipulation, etc.

Any number of grids can be defined for the die, each with a priority relative to the others. Grids are listed one per file line, using keyword:value pairings separated by | characters.

As of revision 2, the items defined for a grid are:

- *GRID*: Identifies this as a line defining a grid.
- *Rev:<integer>* – The format revision for the grid line.
- *Name:<string>* – The user name for this grid region.
- *Extents:<2-point list>* – The lower left and upper right coordinates definition this grid region.
- *Priority:<integer>* – The priority of this grid relative to others. The -1 priority is reserved

for the base grid, whose size matches that of the die component extents.

- *Scheme:<string>;<string>;<string>;<flags>* – The automatic pin numbering scheme for this grid. The first string represents the pattern name, the second is the location of the first pin in the grid for numbering purposes, the third column is a prefix string which should be applied to all pins in this region.
- *Pitch:<DB value>;<DB value>* – This section defines the horizontal and vertical pin pitch by which pins should be snapped.
- *Offset:<DB value>;<DB Value>;<string>* – The offset from the grid boundary to the first legal grid point. This allows the grid to define a keepout region around its border. These values represent the X offset, Y offset, and the corner from which this offset applies.
- *Flags:<String>* – Boolean flags used internally.

An example of this section for a die with a single, uniform pin grid, would be:

```
The following section defines your saved grid parameters for this symbol.

Editing of these values is NOT supported. Doing so may corrupt your design.

Begin grids:

GRID: REV:2|NAME:BASE_GRID|EXTENTS:((-5000.00 -5000.00) (5000.00 5000.00))|PRIORITY:-
1|SCHEME:NUMSPRCCW;TOPLEFT;;0 0 0
0|PITCH:100.00;100.00|OFFSET:300.00;300.00;BOTLEFT|FLAGS:TFFTF

End grids.
```