Milkyway™ Environment Data Preparation

User Guide

Version A-2007.12, September 2008

Comments?
Send comments on the documentation by going to http://solvnet.synopsys.com, then clicking "Enter a Call to the Support Center."

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, Design Compiler, DesignWare, Formality, HDL Analyst, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM, HSIM^{plus}, i-Virtual Stepper, IICE, In-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

Appendix D. Top Design Format File

Appendix E. Preparing a Library and Generating an LM View

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *Milkyway Environment Release Notes* in SolvNet.

To see the *Milkyway Environment Release Notes,*

1. Go to the Download Center on SolvNet located at the following address:

   http://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

2. Select Milkyway Environment, and then select a release in the list that appears.

About This Guide

The *Milkyway Environment Data Preparation User Guide* describes how to use the Milkyway data preparation tool.

Audience

Users of this manual must be familiar with physical IC implementation concepts (specifically, standard-cell-based design) and with programming in an interactive extension language.

The intended audience uses a Synopsys physical implementation tool. This guide provides information for setting up the place and route process, preparing libraries, translating physical design data, processing standard cells, and setting up required technology data.

Using Swish-e With Physical Implementation Online Help

The Swish-e program, which is provided for online Help, is not "Licensed Software" or "Licensed Product" in your license agreement with Synopsys. The Swish-e program is subject to the license provisions found at http://swish-e.org, where you can also find the source code for Swish-e.

Related Publications

For additional information about Milkyway Environment, see Documentation on the Web, which is available through SolvNet at the following address:

http://solvnet.synopsys.com/DocsOnWeb

You might also want to refer to the documentation for the following related Synopsys products:

- *Milkyway Environment Extension Language Reference Manual*
- *Milkyway Environment C-API Reference Manual*
- *Physical Implementation Getting Started*
- Physical Implementation Online Help

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

http://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr

1

Starting the Data Preparation Process

The data preparation tasks described in this guide are performed with the Milkyway data preparation tool. You can also perform data preparation functions in the Astro tool by choosing Tools > Data Prep.

Besides being the name for the data preparation tool, Milkyway is the name for the unified database the tool interacts with. In this guide, unless otherwise specified, *Milkyway* refers to the tool, not the database.

The Milkyway tool and the data preparation functions in other tools can perform several tasks essential for cell library and data preparation, including the following:

- Creating cell libraries
- Importing cell data
- Translating and loading CLF timing data
- Specifying technology information
- Writing technology information to a file
- Removing cell hierarchy
- Specifying power and ground port types
- Optimizing the standard cell layout
- Extracting pin and blockage information

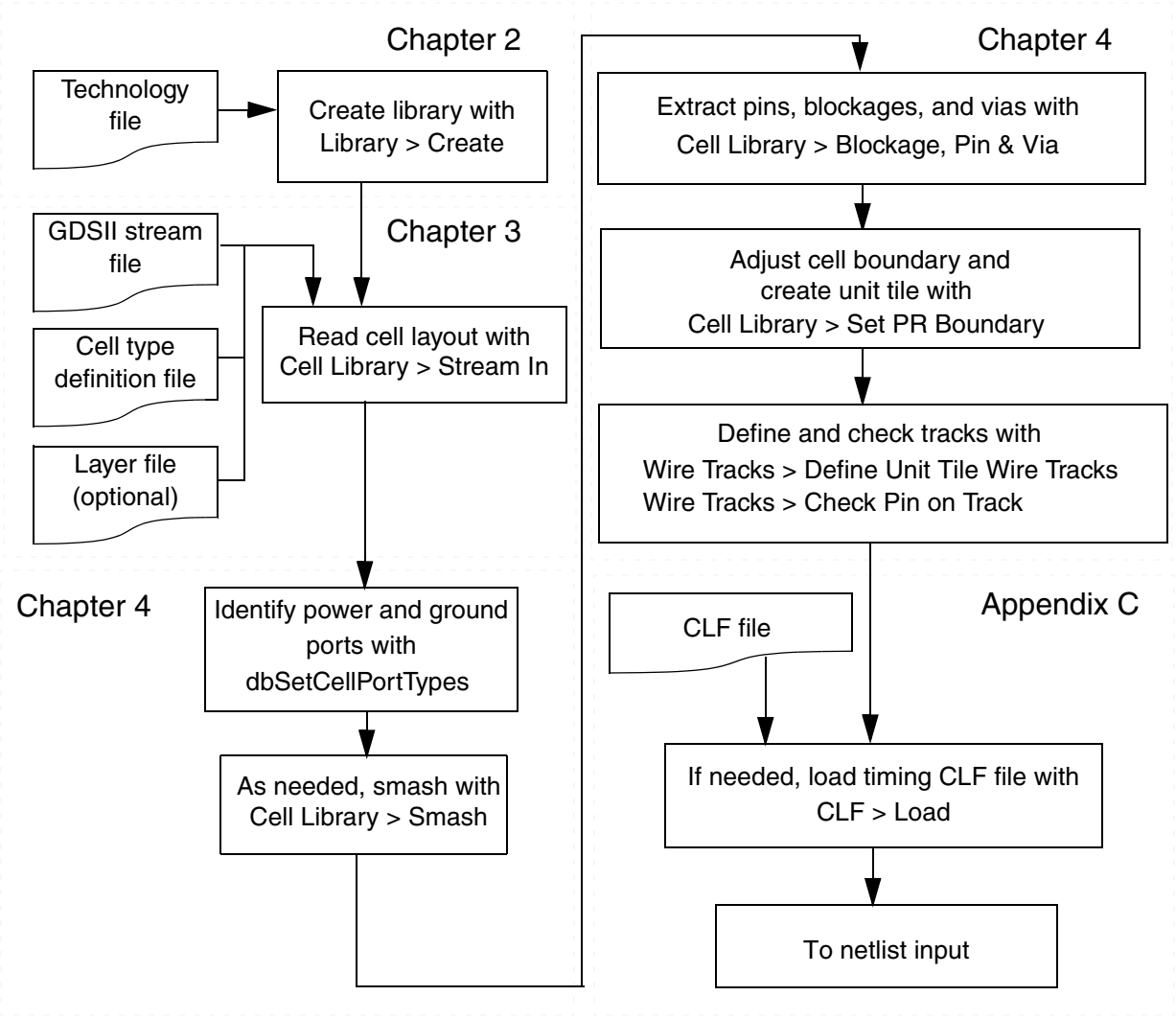
- Setting place and route boundaries
- Defining wire tracks

This chapter contains the following sections:

Activity Flow for Data Preparation

The flow diagram in Figure 1-1 describes the basic steps required to create and load a library and prepare your data for place and route. These steps are documented in the subsequent three chapters and appendixes:

Figure 1-1 Activity Flow for Data Preparation



Starting Milkyway

In the directory containing your design data files, start Milkyway by typing

```
% Milkyway &
```

Note:
It is best to open Milkyway in the directory containing your design data files, although it can be opened in any directory. That way you will be able to more easily access libraries and design cells and the created .cmd and .log files, because they will be in a predictable location.

For information about command-line options and licensed feature sets, see the *Physical Implementation Getting Started*.

Exiting Milkyway

To exit Milkyway,

1. Enter `hiQuit` (see Physical Implementation Online Help) or choose Tools > Quit.

   A dialog box asking if you really want to quit appears.

2. Click OK.

3. You can also exit Milkyway by typing `exit` in the input area.

AServer/AMonitor

AServer is a server process that allows different processes to communicate with each other. Its primary purpose is to allow a tool based on Milkyway to fork a process (run as an executable) through AMonitor.

Each workstation should have only one AServer process per platform running at a time no matter how many instances of a tool you are executing. This rule is necessary because the port number used by AServer is hard-coded, and any tool based on Milkyway tries to connect to that hard-coded port when starting up.

If AServer fails to start up because the port is already in use by another application, you can set the `AServerPort` environment variable to a free port number as shown in the following example:

```
% setenv AServerPort 1978
```

When you set the `AServerPort` variable to a free port number, such as 1978 in the previous example, AServer determines whether the port is available. If it is available, the tool connects to it.

If the port is not available, AServer fails to start. If this happens, you will need to reset the `AServerPort` variable to another free port number until AServer successfully starts up. You can get additional free port numbers from the system administrator.

When all tools connected to the port are disconnected, AServer terminates itself. You can also end the AServer process by issuing the following command:

```
% kill -9 processID
```

Ending the process is not recommended and should be used only in cases where there are no tools based on Milkyway running and AServer does not exit. Each tool process has one AMonitor process associated with it; therefore, AMonitor quits if you terminate the tool process.

Running a Script in Batch Mode

As an alternative to using the GUI, you can run a script in batch mode by using the `-nullDisplay` option. When you enter `-nullDisplay` on the command line of a tool based on Milkyway, the tool launches an X server, and all windows are displayed to that X server.

You can also choose a VNC server as the X server. See “Choosing an X Server” for information on how to specify your own X server. Note that you cannot use the VNC server with the `-nullDisplay` option for debugging purposes.

Using the -nullDisplay option

You can use Tcl or Scheme scripts with the `-nullDisplay` option, as shown in the following examples.

To run a Tcl script without the GUI, enter the following on the command line:

```
% Milkyway -tcl -file tcl_script -nullDisplay
```

To run a Scheme script without the GUI, enter the following on the command line:

```
% Milkyway -replay scheme_script -nullDisplay
```

The Milkyway tool writes out a log file that is named Milkyway.log.<date>, which is standard output and a standard error log. This log file is written to the current working directory.

To find help on `-nullDisplay` and other options, enter the following on the command line:

```
% Milkyway -help
```

Choosing an X Server

You can choose a specific X server by setting the `NullXServerVer` environment variable. To specify an X server (such as version 1 in the following example), enter the following on the command line:

```
% setenv NullXServerVer 1
```

You can choose from the following X servers:

Version 1

Uses the original X server shipped with Milkyway (named NullXServer).

- Advantages of this version

  The version is stable and fast. This is the recommended version for running regressions.

- Disadvantages of this version

  You cannot write out a design layout to a .gif file, and you cannot monitor the job with a VNC viewer.

Version 2

Uses binary Xvnc as the X server. To use the binary Xvnc server, you must obtain the free Xvnc download from RealVNC. Then make a symbolic link named Xvnc in the directory where Astro resides and point it to the downloaded Xvnc. If you do not create this link, the binary named NullXServer2 (Xvnc 3.3.7) is used.

- Advantages of this version

  You can monitor the job with a VNC viewer, and you can write out a design layout to a .gif file.

- Disadvantages of this version

  Xvnc is a third-party, free server; thus there is no guarantee that the server will be stable or fast.

Version 3

Uses a VNC server as the X server. You can enter “vncserver” in a console to see whether a VNC server is installed on your system. If you do not have a VNC server installed, you can download it for free from RealVNC.

Note:

This version is only for advanced users who are familiar with X server configuration.

- Advantages of this version

  You can monitor the job with a VNC viewer and have full control of the VNC server configuration. You can also write a design layout to a .gif file.

- Disadvantages of this version

  Requires you to install a VNC server. Xvnc is a third-party, free server; thus there is no guarantee that the server will be stable or fast.

2

Creating and Loading a Library

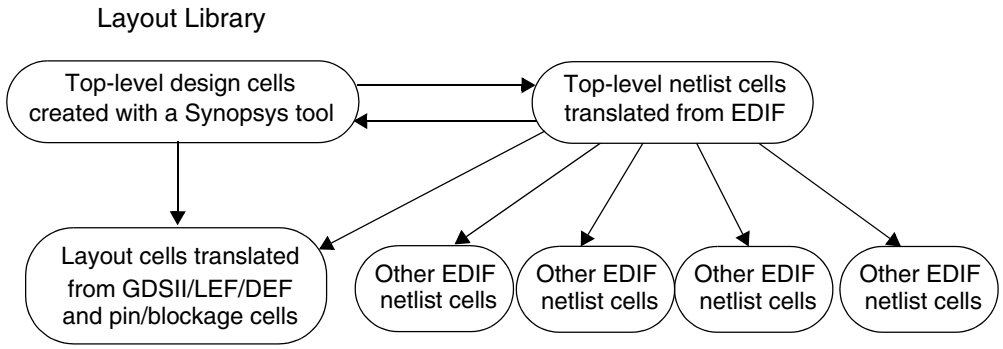
Before you can import and use data for your design in Milkyway, you must have cell libraries in which to place that cell data. The Milkyway tool provides a way of creating and loading libraries for this data. A cell library can contain both top-level designs and the cells used in those designs.

This chapter contains the following sections:

Organizing Libraries

For best performance, organize your libraries as shown in Figure 2-1. The arrows point from libraries to reference libraries. (See “Referencing Cells in Another Library” on page 2-6.)

Figure 2-1 Recommended Library Structure



This structure minimizes the number of reference libraries searched from the design library and therefore improves application performance.

Note:

Through the data preparation and design processes, you create additional cells in these libraries. For example, after pin and blockage extraction, the layout library contains the pin/blockage cells.

Creating a Library

You can create a library with Milkyway. Before you can create a library, you need a technology file. For instructions on creating a technology file, see “Creating and Loading a Technology File” on page A-3.

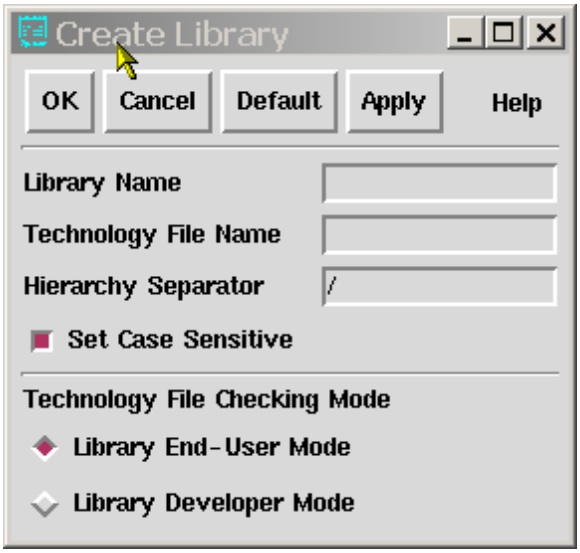
Note:

Set case sensitivity prior to creating the library. Case sensitivity cannot be changed once the library is created. For more information, see the *Physical Implementation Getting Started*.

To create a new library,

1. Enter `cmCreateLib` or choose Library > Create.

   The Create Library dialog box appears.



2. Enter the library name.

   This is the name you want to assign to the new library.

   Few restrictions on the names given to objects are imposed, but if you plan to exchange data with other tools or export data to other formats, you should adhere to the following guidelines when naming your library:

   - The first character must be alphabetic.
   - The remaining characters can be alphabetic, numeric, or the underscore (_).
   - The length must not exceed 31 characters, except for stipple and line style pattern names, which should not exceed 15 characters.

     For more library naming conventions, see “Naming Conventions for Milkyway Library Cells” on page 2-4.

   Caution!

   Failure to conform to these guidelines can result in an inability to export your data.

   Note:

   For information about database naming restrictions, see the Milkyway Environment Extension Language Reference Manual.

3. Enter the technology file name.

   This is the name of the technology file you want to associate with the new library.

4. Enter the hierarchy separator.

   This is the character you want to use as a separator in hierarchical names. If this character is used in cell instance or net names, the name is translated with a backslash (\) preceding the hierarchy separator character.

   The following characters are allowed as hierarchy separators in the Milkyway database:

   - / (forward slash)
   - . (period)
   - | (pipe)
   - # (pound)
   - @ (at)
   - ^ (caret)

5. Select Set Case Sensitive whenever you want case sensitivity for the library. For more information, see *Physical Implementation Getting Started*.
6. Click OK.

   The `cmCreateLib` command creates the library catalog file, which contains the compiled technology file information. The command also creates a directory with the library name you specified to hold the cell files.

Naming Conventions for Milkyway Library Cells

The Milkyway library user interface is structured as follows:

Cellname.Viewname;version

If you open a library and look at a cell inside the library, you can see that the naming convention follows the Cellname.Viewname;version convention. The dot (.) and semicolon (;) are delimiters to identify the cell name and the version of the cell. Anything prior to the dot (.) is the name of the cell, and anything after the semicolon (;) is the version of the cell. These special characters are reserved for this purpose in the Milkyway database. Therefore, you cannot use a dot (.) or a semicolon (;) as part of your cell names in the library.

Technology Checking Modes

The Milkyway tool always performs checks on the technology file for errors. You have two checking levels to choose from.

- Library End-User Mode

This is the default level. The end-user mode is less restrictive. Use this mode if you receive technology files from a library vendor and want to create design libraries. Errors are reported with less severe labeling; therefore, they are not as useful for debugging.

- Library Developer Mode

  This mode is more restrictive. It is for library developers who create technology files before passing them to the end user. This mode provides the maximum level of reported warning messages to help create clean technology files.

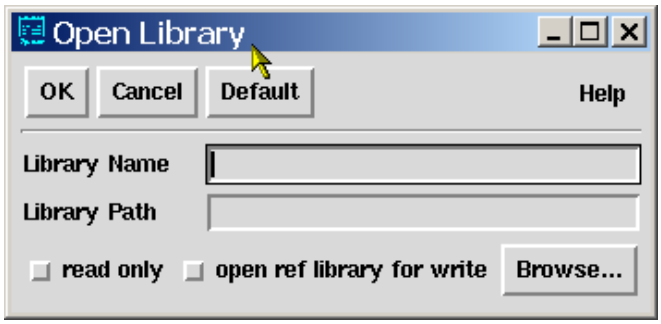
Opening a Library

You must open a library before you can open a cell. To open a library, start Milkyway in the directory containing your libraries.

To open an existing library,

1. Enter `geOpenLib` or choose Library > Open.

   The Open Library dialog box appears.



2. Fill in the Open Library dialog box. For detailed descriptions of the command options, see Physical Implementation Online Help.
3. Click OK.

   The `geOpenLib` command opens the library, and a message appears in the message area of the application window.

Referencing Cells in Another Library

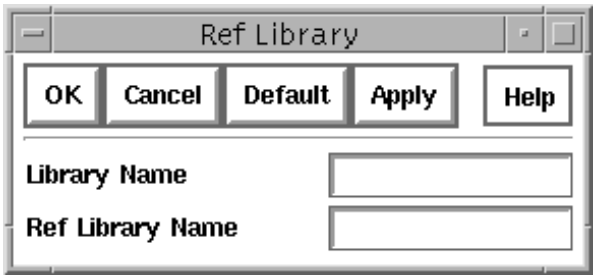
You can work in only one library at a time. However, you can access cells in another library by making a reference library of the library you are working in. When working with reference libraries, you should understand the following:

- The reference you create when you associate a reference library with a library is a one-way reference. Therefore, if B is the reference library of A, you cannot access cells in A from B unless you make A a reference library of B.
- The reference you create when you associate a reference library with a library is a one-level reference. In other words, if C is a reference library of B and B is a reference library of A, you cannot access cells in C from A unless you make C a reference library of A.

To associate a reference library with a library,

1. Enter `cmRefLib` or choose Library > Add Ref.

   The Ref Library dialog box appears.



2. Fill in the Ref Library dialog box. For detailed descriptions of the command options, see Physical Implementation Online Help.
3. Click OK.

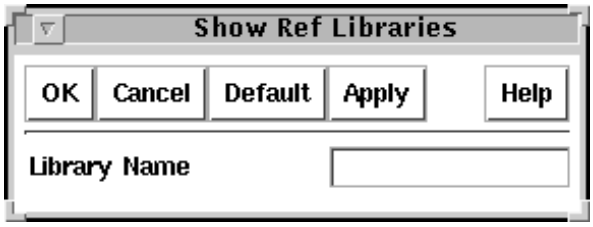
   The libraries become associated so that anytime you open the library specified by Library Name, you can access cells in the library specified by Ref Library Name.

Note:
You can open cells in a reference library only in read-only mode.

To list the reference libraries associated with a library,

1. Enter `cmShowRefLib` or choose Library > Show Refs.

   The Show Ref Libraries dialog box appears.



2. Enter the library name.

   This is the name of the library for which you want to list reference libraries.

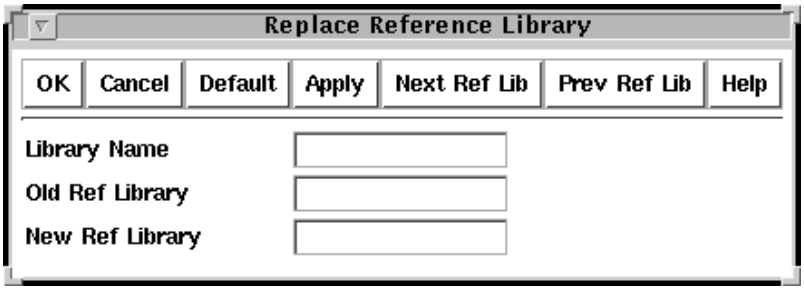
3. Click OK.

   The reference libraries are listed in the message area.

To change a reference library associated with a library,

1. Enter `cmReplaceRefLib` or choose Library > Replace Ref.

   The Replace Reference Library dialog box appears.



2. Fill in the Replace Reference Library dialog box. For detailed descriptions of the command options, see Physical Implementation Online Help.
3. Click OK.

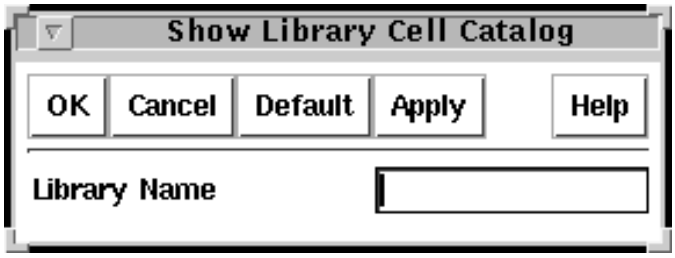
Listing the Cells in a Library

When you are working in a library, you might want to see a list of the cells in that library.

To see a list of cells in the open library,

1. Enter `cmShowLibCat` or choose Library > Show Cells.

The Show Library Cell Catalog dialog box appears.



Note:

This dialog box lists only the cells in the open library. It does not list cells in reference libraries.

2. Use the scroll bars to view the list.

   You can also save the list of cells to a file by using File > Save As from the Library Cell Catalog dialog box.

3. Select File > Close from the Library Cell Catalog dialog box.

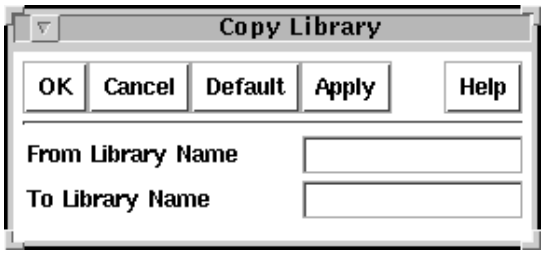
Copying a Library

Copying a library creates a new library catalog file and a new library directory containing all the cells in the original library. You can copy a library only when no libraries are open.

To copy a library,

1. Enter `cmCopyLib` or choose Library > Copy.

   The Copy Library dialog box appears.



2. Fill the Copy Library dialog box. For detailed descriptions of the command options, see Physical Implementation Online Help.
3. Click OK.

   The new library is created.

Controlling Access to a Library

You can run Synopsys applications in either of the following access modes:

- Shared access mode allows other users to open a library while you have that library open. To use Milkyway in shared mode, enter the following on the command line:

  ```
  % Milkyway &
  ```

  The Synopsys application uses the UNIX system locks to enforce library access control. When you open a library, the Synopsys application places a write lock on the library (and a read lock on any reference libraries).

- In shared access mode, the lock remains only during read and write operations. The Synopsys application releases the lock at the completion of a read or write operation, so that other users can access the library.

In the case of a system crash, a cell might remain locked. The following procedure unlocks a cell if your system crashes. For more information about unlocking a cell, see the *Physical Implementation Getting Started*.

To unlock a library locked by a system crash,

1. From the Synopsys directory, make a copy of the library information file, by typing the following at the UNIX prompt:

   ```
   cp libName/lib temp
   ```

   where *libName* is the name of the library you want to unlock.

2. Replace the original library information file with the copy, by typing the following:

   ```
   mv temp libName/lib
   ```

   where *libName* is the name of the library you want to unlock.

Closing a Library

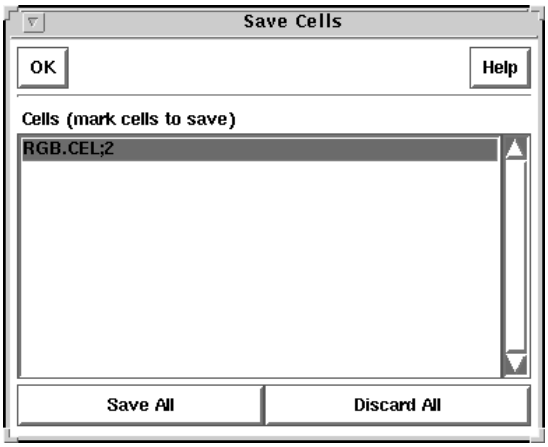
You can work in only one library at a time. If you have a library open and want to open a cell in another library, you must close the current library before you open the library containing the cell you want to open.

When you close a library, all open cells are automatically saved and closed.

To close a library,

1. Enter `geCloseLib` or choose Library > Close.

   If any open cells have been changed, the Save Cells dialog box appears.



2. Select the cells you want to save from the list of cells.

   The cells you select are highlighted.

   You can also select a cell by typing the name of the cell you want included in the Selection and clicking Select. Or you can use pattern matching to select all cells with names that match a pattern. For example, typing .*CEL and clicking Select selects all cells with names that end with CEL. For more information about pattern matching, see the *Physical Implementation Getting Started*.

3. Click OK.

   The cells you specify are saved, and the library is closed.

3

GDSII and OASIS Data Preparation

To prepare your data, you must first import the physical cells into one or more libraries, which can be done with GDSII or OASIS data. The steps for importing the physical cells are covered in this chapter, which contains the following sections:

Specifying Cell Types

GDSII Stream files do not designate the type of a cell. However, your design tool might require the cell type information for a cell. Milkyway allows you to map cell types to cells in the layout by creating a cell type definition file.

After creating this file, you specify it in the Cell Type Definition File box (see gds2Arcs.map, shown in Figure 3-1 on page 3-4). This text file contains cell type statements, in the following format:

```
cellType {cellName}
cellType {cellName}
cellType {cellName}
...
```

where

cellType is a keyword (not enclosed by quotation marks) that indicates the cell type of the cells specified by *cellName*.

Valid values:

- `gdsMacroCell` marks the specified cells as macro cells.
- `gdsStandardCell` marks the specified cells as standard cells.
- `gdsCornerCell` marks the specified cells as corner cells.
- `gdsPadCell` or `gdsIOCell` marks the specified cells as pad cells.
- `gdsStdFillerCell` marks the specified cells as standard filler cells.
- `gdsPadFillerCell` marks the specified cells as pad filler cells.
- `gdsFCDriverCell` marks the specified cells as flip chip drivers.
- `gdsFCPadCell` marks the specified cells as flip chip pads (bumps).
- `gdsOtherCell` marks the specified cells, such as feed-through cells, vias, transistors, and top-level cells, as cells you want ignored.

Valid values for ApolloGAII:

- `macroCell` marks the specified cells as macro cells
- `coverCell` marks the specified cells as cover cells.
- `imageCell` marks the specified cells as image cells.
- `moduleCell` marks the specified cells as module cells.
- `cornerCell` marks the specified cells as corner cells.
- `padCell` marks the specified cells as pad cells.
- `otherCell` marks the specified cells, such as feed-through cells, vias, transistors, and top-level cells, as cells you want to ignore.

cellName is a name of a cell

Valid values: Name of any cell in the library, or an asterisk (*) to indicate that you want all unspecified cells to be marked as indicated by *cellType*. When specifying multiple cell names, separate the names with spaces.

Because the asterisk (*) indicates the default cell type used to mark cells not specified as any other type, you cannot use an asterisk for more than one cell type.

Note: You can change the type of a cell after translation by using Layout > Mark Cell Type.

Valid cell names can be the name of any cell in the library or an asterisk (*), which indicates all unspecified cells to be marked by the indicated cell type.

You cannot use the asterisk (*) for more than one cell type. This is because the asterisk indicates the default cell type used to mark all cells not specified with a cell type. You can change the type of cell after translation by using Layout > Mark Cell Types in Milkyway.

When specifying multiple cell names, you must separate the names with spaces. You can also add comments to the file by preceding them with a semicolon (;).

Note that unless specified in a cell type definition file, cells are marked as standard cells by default. In other words, cells are marked as standard cells if you do not supply a definition file.

Example

```
gdsMacroCell BLOCK1 BLOCK2 BLOCK3
gdsOtherCell TEST VIA1 RCAP FEED2
gdsStandardCell *
```

Using this cell type definition file, the translation marks the cells as specified in the file. If the GDSII Stream file contains cells not specified in the file, the translation marks them as standard cells.

Translating GDSII Stream Data

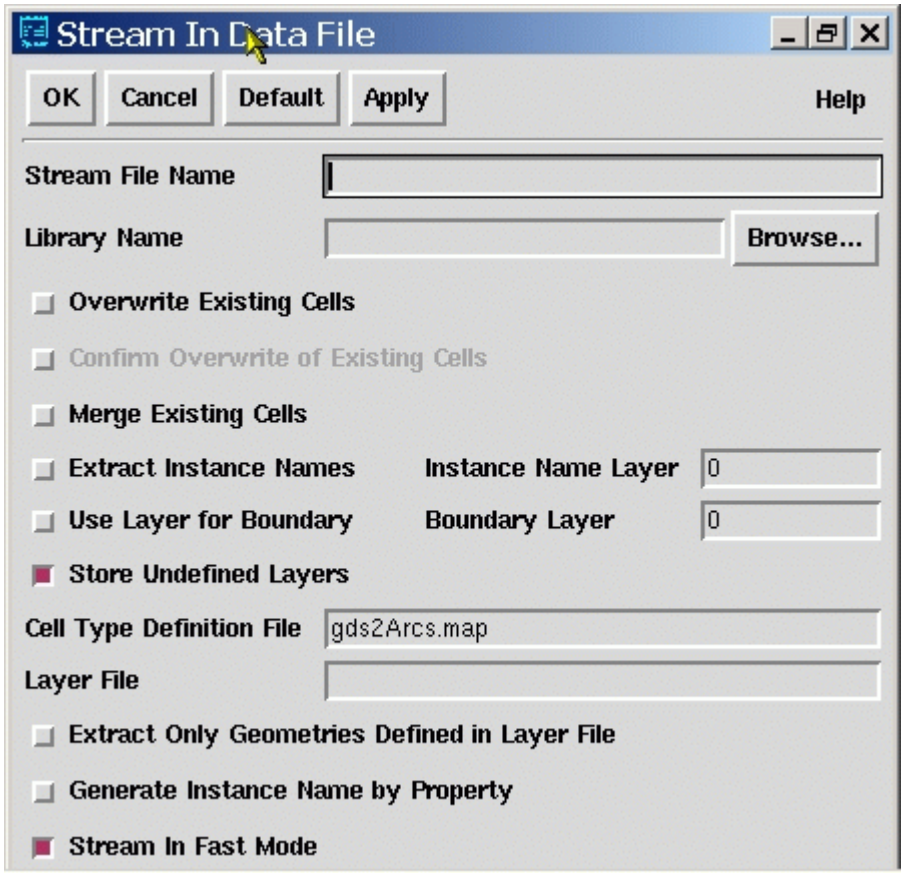
After you have the input files you need, you are ready to translate your GDSII Stream files. Using Layout > Stream In, you can translate all or part of the cells in the layout.

To import a library in GDSII Stream format,

1. Enter `auStreamIn` or choose Cell Library > Stream In.

The Stream In Data File dialog box appears, as shown in Figure 3-1.

Figure 3-1 Stream In Data File Dialog Box



2. Fill in the Stream In Data File dialog box. Specify the Stream file name and the library name. It is recommended that you also specify cell types for mapping in the Cell Type Definition File box and specify layer mapping in the Layer File box. For detailed descriptions of command options, see Physical Implementation Online Help.
3. Click OK.

The `auStreamIn` command translates the cells, creates a Synopsys layout cell for each cell, and places them in the library specified.

If a cell type is marked as “other,” `auStreamIn` displays a message to let you know that a cell type is not set for the cell. The `auStreamIn` command ignores a cell with no cell type in further processing.

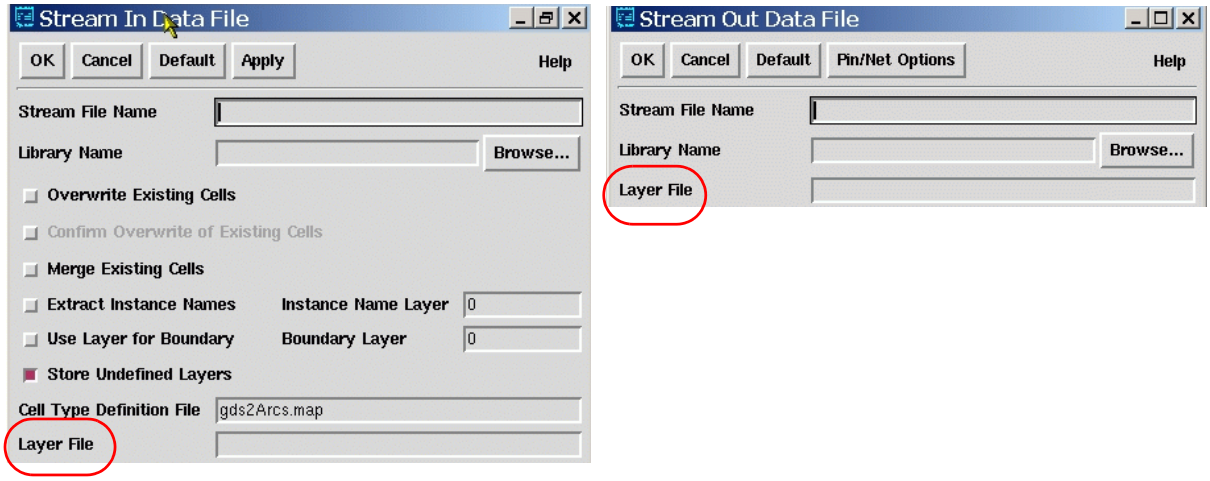
If you mark a cell type or tile class incorrectly, you can change it after the translation. To change a cell type, use the `cmMarkCellType` command. For more information, see Physical Implementation Online Help.

About Layer Mapping Files

The following sections discuss the use of the layer mapping files to control the translation of GDSII data into and out of Milkyway.

The Stream In and Stream Out commands (`auStreamIn` and `auStreamOut`) provide options for specifying a layer mapping file to manipulate Stream data. In Scheme, for example, you can enter the layer mapping file in the forms indicated in Figure 3-2. If you are using tool command language (Tcl) equivalents of Stream In and Stream Out commands, you can specify the layer file on the command line.

Figure 3-2 Layer Mapping File Option in auStreamIn and auStreamOut



GDSII Data Format

The GDSII Stream format is the standard file format for transferring or archiving two-dimensional graphical design data. One of the benefits of GDSII is that it is in binary format and is platform- independent. GDSII contains elements associated with layers, such as text, path/wire, boundary/polygon, structure references, and structure array references.

To translate GDSII layers to Milkyway database layers, you can use the `auStreamIn` command or the Stream In Data File dialog box to create a Synopsys layout cell for each cell as well as to place the cells in the library you specified. (For more information, see "Translating GDSII Stream Data" on page 3-3). For greater manipulation of stream data, create a layer mapping file, as described in the following section.

After completing a place and route or chip assembly design in Milkyway, you output the completed design back to GDSII Stream format, using the `auStreamOut` command or the Stream Out Data File dialog box. The GDSII data can then be used for mask creation or as input to other application tools, such as layout verification or chip assembly.

Specifying Layer Mapping in Milkyway

A layer mapping file can be used as an option to the Milkyway Stream In and Stream Out commands. Specifying a layer mapping file allows you to manipulate Stream data to and from the Milkyway database.

The layer mapping file controls the mapping of GDSII layer and data type conversions to a Milkyway layer (and data type) that you specify. You can also map to a Milkyway system reserved layer such as metal1Blockage for standard cell or macro cell place and route applications.

The syntax for the Stream In and Stream Out layer mapping files differs slightly. The syntax and examples for both file types are discussed in "Mapping GDSII Layers to Milkyway Database Layers."

Mapping GDSII Layers to Milkyway Database Layers

This section discusses the syntax used to map GDSII Stream layers to Milkyway database layers and lists the variables, followed by examples.

Syntax

```
MilkywayLayer[:MilkywayDataType]
  GDSIILayer[:GDSIIDataType]
```

Note that you can add comments to the file by preceding them with a semicolon (;). Milkyway ignores all text on the same line following a semicolon.

Table 3-1 lists and describes the variables for the Stream In layer mapping file.

Table 3-1 Variables for Mapping GDSII Layers for Stream In

Variable	Description
`MilkywayLayer`	The number of the Milkyway layer to which you want the GDSII layer translated. Valid values: The number of any layer defined in the technology file.

Table 3-1 Variables for Mapping GDSII Layers for Stream In (Continued)

Variable	Description
	For mapping blockage layers, use the following predefined Milkyway layer numbers: 212 for polyBlockage 216 for metal4Blockage 217 for via3Blockage 218 for metal1Blockage 219 for metal2Blockage 220 for metal3Blockage 223 for polyContBlockage 224 for via1Blockage 225 for via2Blockage
`MilkywayDataType`	The data type of the Milkyway data to be translated. Valid values: Any integer from 0 to 255
`GDSIILayer`	The number of the GDSII Stream layer to be translated. Valid values: Any integer from 0 to 255
`GDSIIDataType`	The data type of the GDSII Stream data to be translated. Valid values: Any integer from 0 to 255

Each geometric object has one layer number (ranging from 0 to 255) and one data type (ranging from 0 to 255). You use layer mapping files to perform layer and data type conversion between GDSII files and Milkyway. Each line in the file maps a GDSII Stream layer to a Milkyway database layer, as shown:

GDSII Stream layer #0 (any data type) maps to Milkyway database layer #0.

GDSII Stream layer #1 (any data type) maps to Milkyway database layer #1.
...
GDSII Stream layer #255 (any data type) maps to Milkyway database layer #255.

Example

```
44 36:3          ; converts GDSII data of type 3 on layer
                 ; #36 to MilkywayLayer #44

45:3 36:6        ; converts GDSII data of type 6 on layer
                 ; #36 to MilkywayLayer #45 dataType 3
```

Discarding Objects on Layers

If you have geometry on layer 12 with data types 1 and 2, and you want to stream in only geometry of data type 1, `auStreamIn` converts both data types to the same layer in Milkyway by default.

To translate layers differently, you specify the mapping for those layers in the layer mapping file. For example, if you do not want to keep geometry with data type 2, you can prevent it from being streamed in by selecting the layer mapping file and deselecting Store Undefined Layers after running the `auStreamIn` command.

For example, first prepare a layer mapping file and do not define layer 130 in your technology file during cell library creation. Then edit a layer mapping file such as the following:

```
130  12:2 ; convert GDSII layer12, datatype 2 to Milkyway
layer 130
```

Finally, run `auStreamIn` and deselect Store Undefined Layers in `auStreamIn`. Because layer 130 is not defined in the technology file, all objects mapped to that layer are discarded.

Mapping Milkyway Layers to GDSII Stream

This section discusses the syntax used to map Milkyway layers to GDSII Stream layers and lists the variables followed by examples.

Syntax

```
MilkywayObjType[MilkywayNetType]
  MilkywayLayer[:MilkywayDataType] GDSIILayer
  GDSIIDataType
```

Note that you can add comments to the file by preceding them with a semicolon (;). Milkyway ignores all text on the same line following a semicolon.

Table 3-2 lists and describes the variables for the Stream Out layer mapping file.

Table 3-2 Variables for GDSII Stream Out

Variable	Description
`MilkywayObjType`	Names the code for the type of object in the data to be translated. Valid types: A for all types T for text D for data

Table 3-2 Variables for GDSII Stream Out (Continued)

Variable	Description
`MilkywayNetType`	Names the code for the net type of the object in the data to be translated. Valid net types: S for signal P for power G for ground C for clock U for Up conduction layer (upper layer in a via). Example: metal1 in an m1/m2 via D for Down conduction layer (lower layer in a via). Example: metal2 in a nm1/m2 via X for power and ground wires or contacts with a “signal” or “tie-off” routing type Note that U and D are used only for via and smashed via objects and override any other net type set for the layer when the layer is in a via. If `MilkywayNetType` is omitted, Milkyway will map all Milkyway data of the specified object type to the specified GDSII layer. If `MilkywayNetType` is included, Milkyway will examine every object of the object type to determine its net type and map it to the layer you specify. If an object has no net (or a net has no type), Milkyway assumes that it is a signal net type. If a layer file contains contradictory lines, the last line will overwrite any previous line. For example, if an earlier line of a layer file specifies mapping for a particular object/net type and a later line specifies mapping for the same object type but no net type, Milkyway will translate all data of the specified object type as specified by the later line.
`MilkywayLayer`	Specifies the name or number of any layer, from 0 to 225, defined in the technology file to be translated.
`MilkywayDataType`	The data type of the Milkyway data to be translated. Valid values: Any integer from 0 to 255

Table 3-2 Variables for GDSII Stream Out (Continued)

Variable	Description
`GDSIILayer`	Specifies the number of the GDSII Stream layer to which you want objects of `MilkywayObjType` on `MilkywayLayer` to be translated. Use any integer from 0 to 255, or use a minus (–) to omit the Milkyway layer during translation.
`GDSIIDataType`	Specifies the number of the GDSII Stream data type to which you want objects of `MilkywayObjType` (and `MilkywayNetType`, if given) on `MilkywayLayer` to be translated. Use any integer from 0 to 255.

If you want Milkyway to translate layers differently, you must specify the desired mapping of those layers in a layer mapping file. Each line in the layer file specifies a particular type of object (text, data, or both) on a specified Milkyway layer that you want translated to a particular GDSII Stream layer with a given data type. You can also omit a Milkyway layer during translation using a mapping file.

Milkyway outputs net and pin text on the same layer as the associated net or pin. For example, if Milkyway translates a pin on layer 5, the pin's text is also on layer 5. However, if you specify a layer file, Milkyway maps the text layer according to the specification in the layer file.

For example, if Milkyway translates a pin on layer 5, using a layer file that maps Milkyway layer 5 to GDSII stream layer 10, the text is on layer 10 instead. Specifying a minus (–) prevents transfer of all text and data on the Milkyway layer.

The following example shows the mapping of Milkyway layers to GDSII Stream.

Example

```
T METAL:2 3 5          ; converts text on Milkyway layer
                       ; METAL data Type 2 to GDSII Stream
                       ; layer #3 data type 5
                       ; Note: If you stream back into
                       ; Milkyway database without using
                       ; the layer map file, this will
                       ; switch your text on layer METAL
                       ; to layer METAL5

TS 16 31 6             ; converts text associated with
                       ; signal nets on Milkyway layer #16
                       ; to GDSII Stream layer #31 and
                       ; data type 6

TP METAL3 16           ; converts text associated with
```

```
                        ; power on Milkyway layer METAL3
                        ; to GDSII Stream layer #16
TG 28 16                ; converts text associated with
                        ; ground on Milkyway layer #28
                        ; to GDSII Stream layer #16
D METAL2 45             ; converts data on Milkyway layer
                        ; METAL2 to GDSII Stream layer #45

DS 45:2 18 5            ; converts data associated with
                        ; signal nets on Milkyway layer #45
                        ; data Type 2 to GDSII Stream layer #18
                        ; data Type 5

A METAL6 3 4            ; converts text and data on
                        ; Milkyway Layer METAL6 to GDSII
                        ; Stream layer #3 and datatype 4
A METAL4 -              ; omits to transfer all text and
                        ; data on Milkyway layer METAL4
```

When you stream data out of Milkyway, the first character of the map file is the code for the type of object to be translated. Use A for all,
T for text, and D for data. The optional second character specifies the net type. For example, S for signal, P for power, and G for ground. For more information about the net types, see Table 3-2.

Objects in Milkyway That Can Be Streamed Out

Only physical information can be streamed out from the Milkyway database to the GDSII stream file. Because the GDSII stream format contains only layout information, connectivity information cannot be streamed out.

Similarly, Milkyway cannot stream out place and route (abstract) objects, such as port, net, port instance, region, and cell row. In addition, extension type objects, such as group, property, and attached file, cannot be streamed out.

Table 3-3 lists Milkyway objects that can be streamed out.

Table 3-3 Stream Out Objects in Milkyway

Milkyway object	GDSII Stream object
CellInstance	SREF (structure reference element)
CellInstArray	AREF (array reference element)
Contact	SREF or boundary element
ContactArray	SREF or boundary element

Table 3-3 Stream Out Objects in Milkyway (Continued)

Milkyway object	GDSII Stream object
HorizontalWire	Path
Path	Path
Pin	Boundary element
Polygon	Boundary element
Rectangle	Boundary element
Text	Text
VerticalWire	Path

Note:

If you choose the Flatten Devices or Device Arrays option in `auStreamOut`, the contact or contact array will be converted into several GDSII boundary elements. By default, Milkyway outputs devices or device arrays as a structure reference element (SREF) with an optional device name prefix, if specified in the dialog box.

C-API Application Considerations

There are limitations on which Milkyway objects can be streamed out to GDSII. However, using the Milkyway C-API, there is no limit to what you can convert directly from a Milkyway database to a third-party database.

You can use C-API functions to create text from abstract objects such as ports, nets, and port instances. For example, you can create text objects with the origin located on a wire object region by extracting the text name.

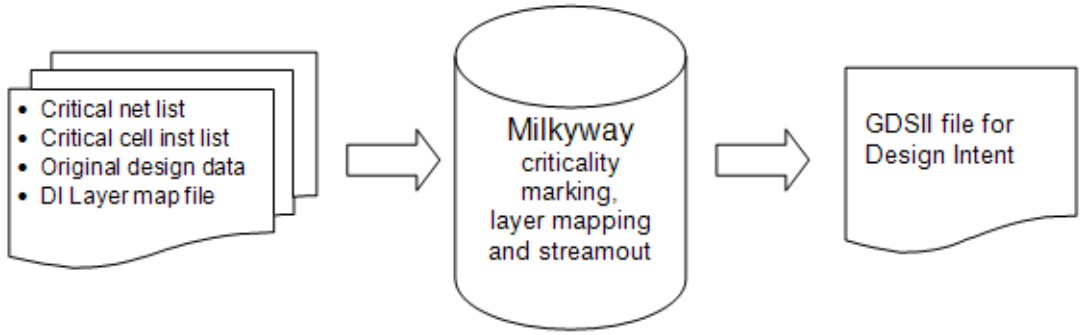
You can also use a C-API function to extract the text name from the net name of the wire object. The text objects can then be streamed out. When you use this method, the net name information for the wire objects can be preserved and passed on to the Stream file.

Design Intent Considerations

Design intent data are extra geometries on top of the regular GDSII design data. These geometries are used to stream out data for critical nets and cell instances in different layer-datatype pairs. Each extra layer-datatype pair is associated with one or more original design layers.

Milkyway supports design intent using the following flow, see Figure 3-3.

Figure 3-3 Design Intent Flow



One of the main uses for design intent is to focus optical proximity correction (OPC) only on critical nets and instances in order to reduce design data processing.

To find out more about using design intent, see the `auStreamOut` command in Physical Implementation Online Help. This command covers the critical net and instance list and the design intent layer-mapping file, which from the format of the layer-mapping file that maps between the Milkyway database and the GDSII layer and datatype.

The Milkyway OASIS Interface

OASIS stands for Open Artwork System Interchange Standard, which defines an interchange and encapsulation format for hierarchical integrated circuit mask layout information. The OASIS reader and writer flow allows you to annotate the Milkyway database with physical information. As a replacement for GDSII, OASIS removes all 16- and 32-bit integer width restrictions. It is a concise data format and provides a richer information palette to facilitate interchange of layout-related data between design and manufacturing.

This section includes the following subsections:

- OASIS Import and Export Commands
- Importing OASIS Into Milkyway Using read_oasis
- Exporting OASIS From Milkyway Using write_oasis

OASIS Import and Export Commands

You translate an OASIS design to and from the Milkyway database by using the following Milkyway commands:

- `read_oasis`

  Imports (reads in) the basic OASIS data into a runtime data structure and then annotates the information from the input to the Milkyway database.

- `write_oasis`

  Exports (writes) a Milkyway design cell to OASIS format by traversing the objects in the database to be output, storing them in a runtime data structure, and then writing the information to OASIS format.

The `read_oasis` and `write_oasis` commands are available in both Scheme and Tcl modes.

For more information about the Scheme and Tcl modes, see the *Milkyway Environment Extension Language Reference Manual*. The details of these commands are described later in this section.

Importing OASIS Into Milkyway Using read_oasis

The OASIS reader (version 1.0) enables you to annotate the Milkyway database with physical information in the input OASIS file by using the `read_oasis` command. (For information about the write oasis dialog box and options used to export OASIS from the Milkyway environment, see “Exporting OASIS From Milkyway Using write_oasis” on page 3-17.)

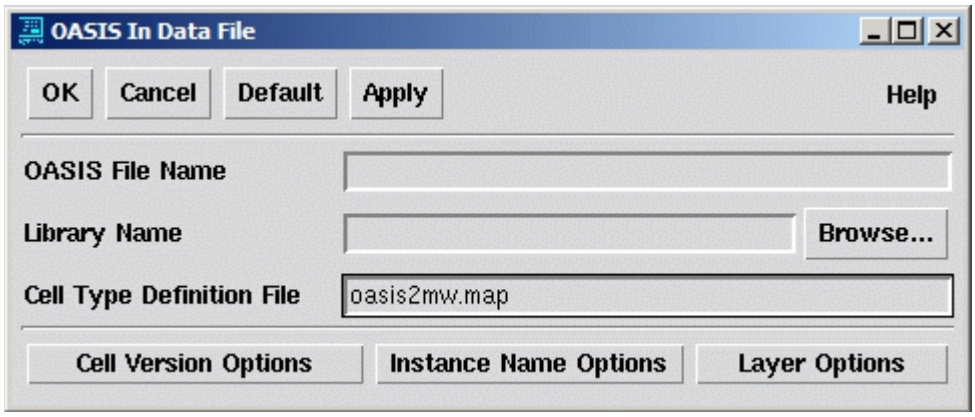
Steps for Importing OASIS Files

The steps for translating a design from OASIS Stream format to Milkyway format by using the `read_oasis` command are as follows:

1. Run the `read_oasis` command, by entering `read_oasis` on the command line.

   When you’ve invoked the OASIS In Data File dialog box, enter the OASIS file name (the name of the OASIS Stream file that contains the layout definition), as shown in Figure 3-4.

Figure 3-4 OASIS In Data File Dialog Box



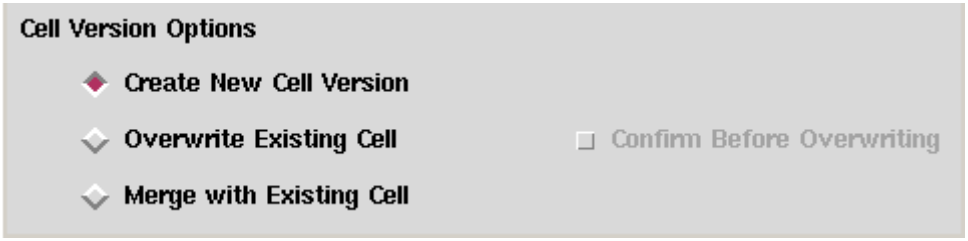
2. Enter the name of the Milkyway library in which you want to place the layout cells in the Library Name field.

   Click the Browse button for help with locating the library files.

3. (Optional) Enter the name of a file that lists the cells by type in the Cell Type Definition File box (for example, oasis2mw.map, as shown in Figure 3-4). If you do not specify a cell type, Milkyway imports all cells as standard cells. For information about creating a cell type definition file, see "Specifying Cell Types" on page 3-2.

4. (Optional) Select Cell Version Options if you want to control the cell version, as shown in Figure 3-5.

Figure 3-5 Cell Version Options



The options are as follows:

- Create New Cell Version (Selected by default)

  Select this option to create a new cell.

- Overwrite Existing Cell

  Select this option to overwrite any cells already in the library with the same name as the cells being imported.

- Confirm Before Overwriting (active if the Overwrite Existing Cell option is selected)

Select this option if you want Milkyway to require confirmation before it overwrites a cell.

- Merge with Existing Cell

  Select this option if you want to merge with existing cells.

5. (Optional) Select Cell Instance Name Options to select the cell instance name options you want to use in the translation, as shown in Figure 3-6.

Figure 3-6 Cell Instance Name Options

The options are as follows:

- Extract Instance Names

  Select this option to extract the names of the cell instances. If you select this option, you must also specify the instance name layer used for the OASIS Stream instance names.

- Instance Name Layer (active if the Extract Instance Names option is selected)

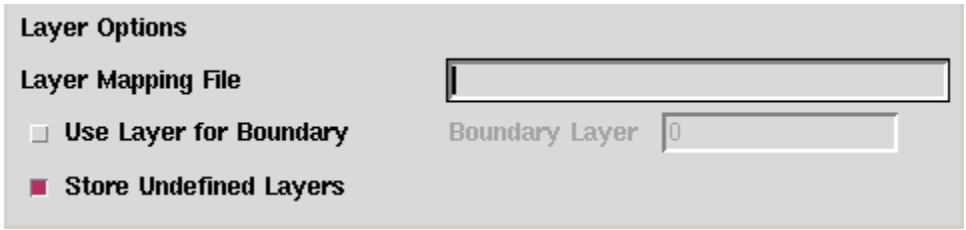
  If you select Extract Instance Names, enter the number of the Synopsys layer. (Milkyway will extract the OASIS Stream instance names and place the data on the specified Synopsys layer.)

- General Instance Name by Property

  Select this option to generate the cell instance names according to the PROPERTY record in the OASIS file. This option is *not* selected, by default.

6. (Optional) Select Layer Options to select the layer options you want to use in the translation, as shown in Figure 3-7.

Figure 3-7 Layer Options



The options are as follows:

- Layer Mapping File

  Enter the name of the file containing special layer mapping instructions (if any). For information about creating a layer file, see “About Layer Mapping Files” on page 3-5.

- Use Layer for Boundary

  Select this option if you want your Synopsys tool to use the cell boundaries in the OASIS layout when placing cells in cell row. If you select this option, you must also specify the boundary layer used for the OASIS Stream boundary.

- Boundary Layer (Active if Use Layer for Boundary is selected)

  If you selected Use Layer for Boundary, enter the number of the Synopsys layer used for OASIS Stream boundaries.

- Store Undefined Layers (Selected by default)

  Select this option to import and store layers not defined in the technology file. Otherwise, any data layer not defined in the technology file will not be imported into Milkyway.

Exporting OASIS From Milkyway Using write_oasis

This section discusses the OASIS Out Data File dialog box and options used to export OASIS from the Milkyway environment. (For information about the `read_oasis` dialog box and options used to import OASIS into the Milkyway environment, see “Importing OASIS Into Milkyway Using read_oasis” on page 3-14.)

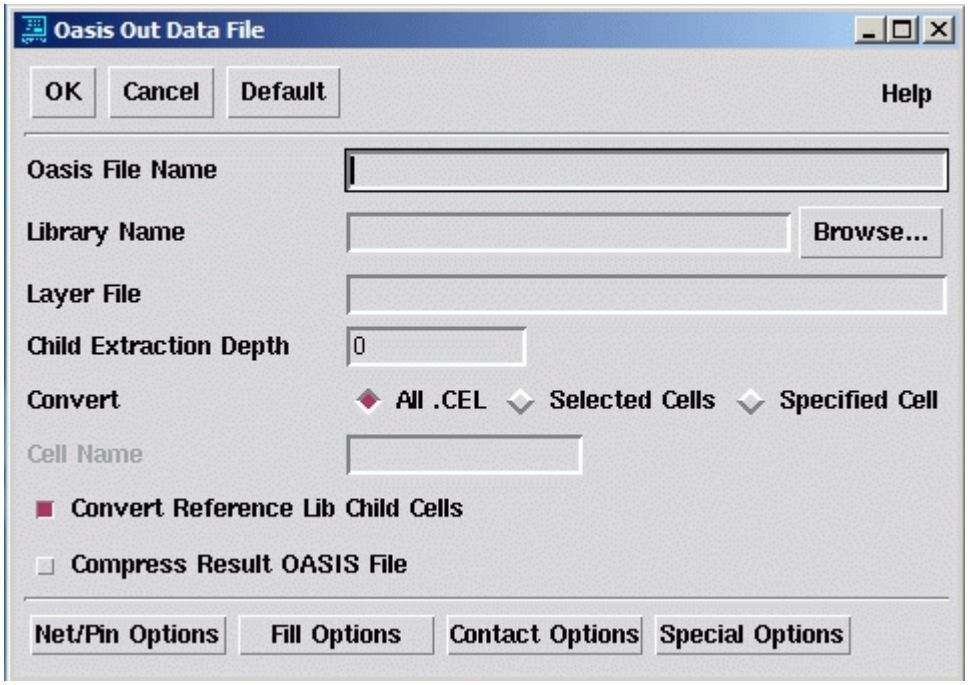
Step 1: The Main Dialog Box

The steps for translating a design from Milkyway format to OASIS Stream format by using the `write_oasis` command are as follows:

1. Run the `write_oasis` command by entering `write_oasis` on the command line.

   When you’ve invoked the OASIS Out Data File dialog box, enter the name you want to assign to the output file in the OASIS File Name box, as shown in Figure 3-8.

Figure 3-8 OASIS Out Data File Dialog Box



2. Enter the name of the library containing the layout cells in the Library Name field.

   Click the Browse button for help locating the library files.

3. (Optional) Enter the name of the file containing the layer mapping instructions, if any, in the Layer File field.

4. Enter the level to which you want to convert child cells in the Child Extraction Depth box.

   If you are exporting the design to perform a design rule checking (DRC) check, enter a large number, such as 20, to ensure extraction of all child cells. Milkyway exports the .CEL version of each child cell.

5. Specify which cells you want to translate by selecting a Convert option. The options are as follows:

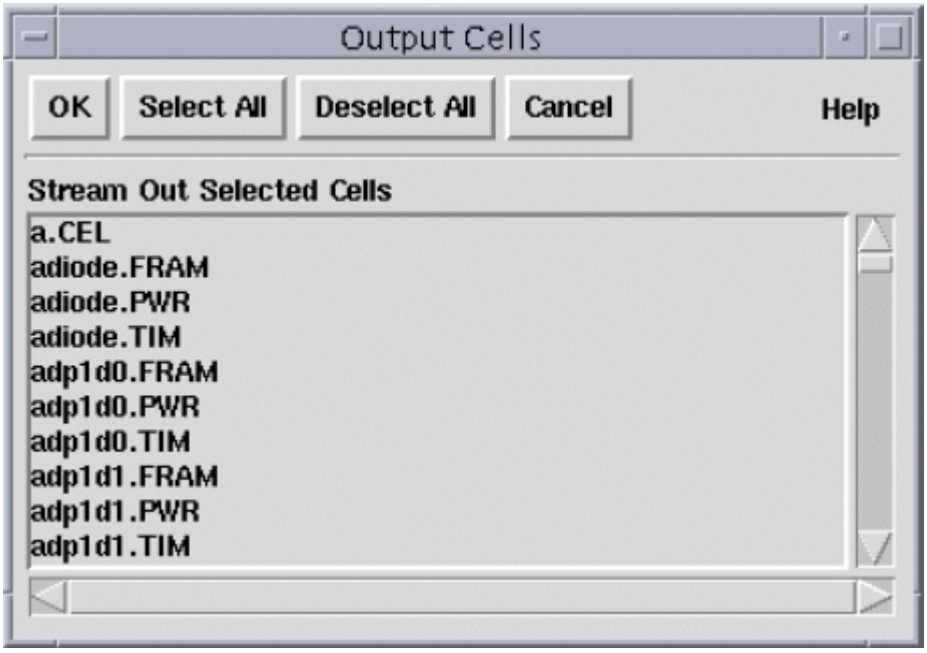
   - All .CEL (Selected by default)

     Select this option to translate all the layout cells in the specified library.

   - Selected Cells

     Select this option to translate only the cells you specify. When you select Selected Cells and click OK, Milkyway displays the Output Cells window, as shown in Figure 3-9. Select the cells you want to translate from a list of all the cells in the library. Then click OK.

Figure 3-9 Output Cells



- Specified Cells

  Select this option to translate a cell you specify by entering its name in the Cell Name box. Select Specified Cells, and enter the name of the cell you want to translate.

6. Select Convert Reference Lib Child Cells, to convert child cells in reference libraries. This option is selected by default.

   Milkyway converts child cells in reference libraries only if the Child Extraction Depth is sufficient to convert cells at that level.

7. (Optional) Select Compress result OASIS File, to compress the result of the OASIS file.

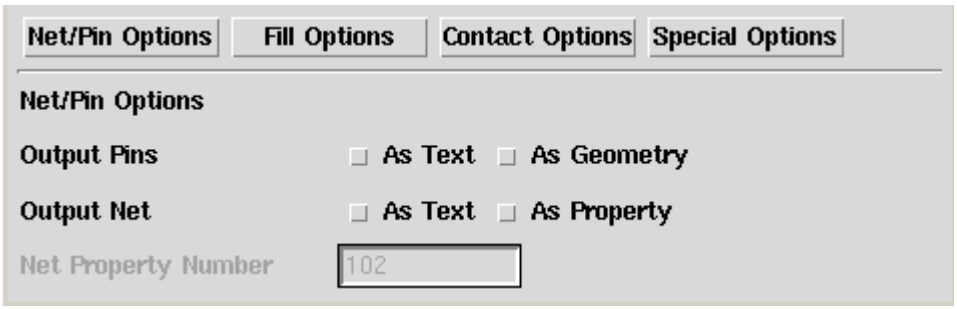
Step 2: (Optional) Net/Pin Options

Note that Milkyway outputs net and pin text on the same layer as the associated net or pin. For example, if Milkyway translates a pin on layer 5, the text is placed on layer 5. However, if you specify a layer file, Milkyway maps the text layer according to the specification in the layer file. For example, if Milkyway translates a pin on layer 5, by using a layer file that maps Synopsys layer 5 to OASIS Stream layer 10, the text is placed on layer 10. (For information about mapping layer files, see “About Layer Mapping Files” on page 3-5.)

To specify net and pin options,

1. Select the options you want to use for translating pins, as shown in Figure 3-10.

Figure 3-10 Net and Pin Options



The Output Pins options are as follows:

- As Text

  Select to translate text associated with pins.

- As Geometry

  Select to translate geometry associated with pins.

Note:

Milkyway outputs pins for selected cells only (not for child cells of selected cells).

2. Select the options you want to use for translating nets. The Output Net options are as follows:

   - As Text

     Select to generate one text string per net and place the text over an arbitrary wire associated with that net.

   - As Property

     Select to generate a property for each object associated with a net. If you select this option, enter a property number in the range of 0 to 127 in the Net Property Number box.

     The property is equal to the net name, unless the net name is greater than 126 characters. If a net name exceeds 126 characters, Milkyway displays a message and does not generate a property for that net. In the case of contacts, Milkyway attaches the property to the contact SREF (if not flattened) and the individual geometries (if flattened).

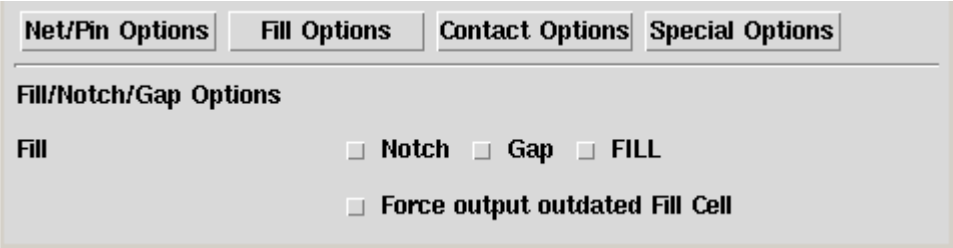
   Note:

   Milkyway outputs nets for selected cells only (not for child cells of selected cells).

Step 3: (Optional) Fill Options

To specify fill options, select the fill options you want to use in the translation, as shown in Figure 3-11.

Figure 3-11 Notch, Gap, and Fill Options



The Notch, Gap, and Fill options are as follows:

- Notch

  Select this option to fill notches that are smaller than the minimum distance required between objects on the same layer.

- Gap

  Select this option to fill gaps that are smaller than the minimum distance required between objects on the same layer.

- FILL

  Select this option to fill notches and gaps that are stored in the .FILL cell (generated by the `geNewFillNG` command). Filling patterns include both notch and gap.

Note:
The Notch and Gap options are designed for .NOTC and .GAP cells. You must generate gap and notch fill information by using the `geNewShortLocator` command before selecting the Notch or Gap option. The FILL option is designed for .FILL cells and is generated by the `geNewFillNG` command.

Note also that Milkyway fills notches and gaps for selected cells only (not for child cells of selected cells).

Step 4: (Optional) Contact and Contact Array Options

To specify contact and contact array options (as shown in Figure 3-12), select the contact and contact array options you want to use in the translation. Use the Flatten options to specify the objects you want to break down into component parts.

Figure 3-12 Contact and Contact Array Options

The Contact and Contact Array options are as follows:

- Devices

  By default, this option is *not* selected. Milkyway therefore translates contacts as references to the technology file.

- Device Arrays

  By default, this option is *not* selected. As a result, Milkyway creates cells with the following naming convention from a device array:

  prefix_contactName_xViaPitch_yViaPitch_xReps_yReps

  where

  prefix specifies the Device Name Prefix you specified in the Contact/ContactArray Options dialog box.

  contactName specifies the name assigned to the contact in the deviceTable section of your technology file.

  xViaPitch specifies the horizontal distance (in database units) between centers of contacts in the array.

  yViaPitch specifies the vertical distance (in database units) between centers of contacts in the array.

  xReps specifies the number of columns (contacts in the horizontal direction) in the array.

  yReps specifies the number of rows (contacts in the vertical direction) in the array.

  If the contact name is V1 and you specified a device name prefix of $$ (as shown in Figure 3-12), the *prefix* and contactName are $$V1.

  The name $$V1_2000_3000_4_4 therefore specifies a four-by-four array of V1 contacts. This contact array is spaced with 2,000 database units from center to center in the horizontal direction and 3,000 database units from center to center in the vertical direction.

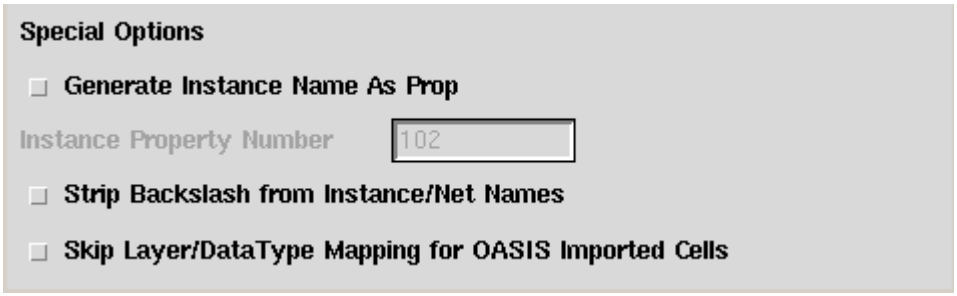
- Device Name Prefix

Enter the prefix you want to add to the contact names. If you use the default prefix ($$), Milkyway translates each contact as $$deviceName, where deviceName is the name assigned to the contact in the deviceTable section of the technology file. For example, a contact named via1 translates to the OASIS file as $$via1.

Step 5: (Optional) Special Options

To specify Special Options (as shown in Figure 3-13), select the options you want to use in the translation.

Figure 3-13 Special Options



The special options are as follows:

- Generate Instance Name As Prop

  Select this option if you want Milkyway to attach a property specifying the instance name to each cell instance in the design.

  Note:

  Milkyway generates instance name properties for selected cells only (not for child cells of selected cells).

- Instance Property Number (active if the Generate Instance Name As Prop option is selected)

  Enter a number in the range of 0 to 127 to indicate the property number of the instance name property.

- Strip Backslash from Instance/Net Names

  Select to remove backslashes from hierarchical cell instance and net names that might conflict with hierarchical names.

- Skip Layer/Data Type Mapping for OASIS Imported Cells

  Select to ignore layer and the data type mapping for those cells created from the OASIS file during the `read_oasis` process. The layer number and data type are written to the OASIS file without any conversion.

4

Data Preparation Using LEF and DEF

The steps for importing LEF and DEF files are covered in this chapter, which contains the following sections:

Library Preparation Using LEF

In addition to GDSII, the Library Exchange Format (LEF) is another method of providing library information from a third-party database to Milkyway. LEF defines the elements of an IC process technology and associated library of cell models and contains library information for a class of designs. This library data includes layer, via, placement site type, and macro cell definitions. Milkyway supports all LEF versions, including version 5.6.

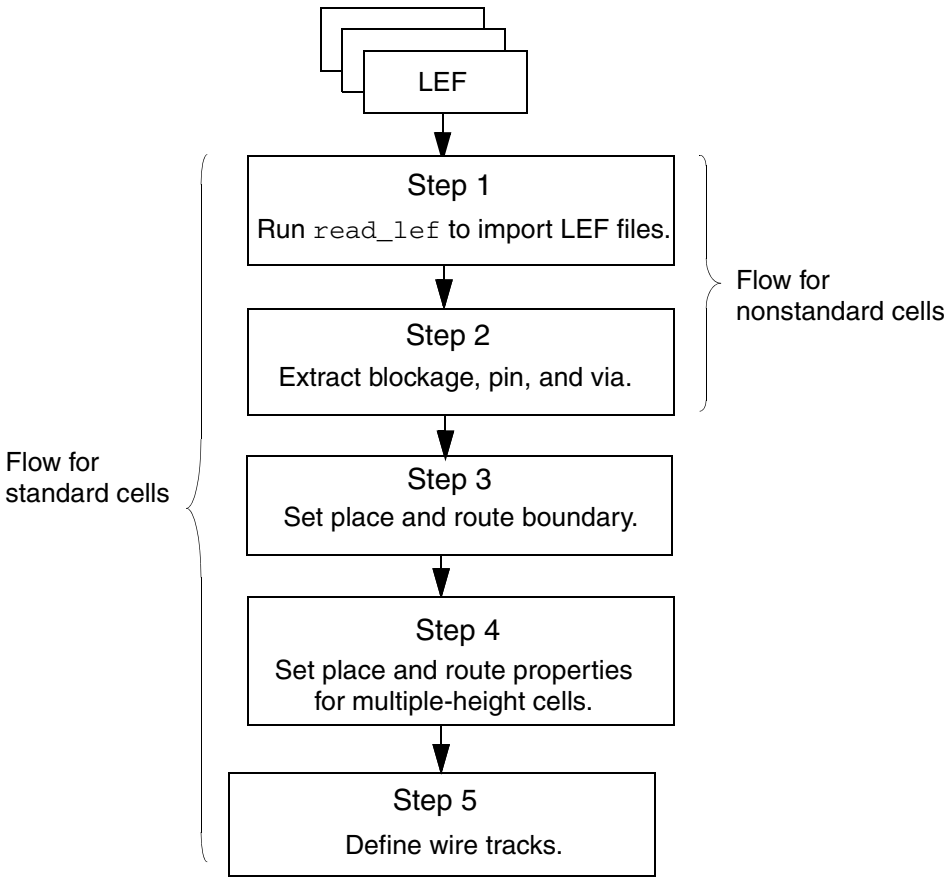
The commands for importing LEF data are `read_lef` (automated flow) and `read_lib` (advanced flow) for standard and nonstandard cells. Both commands provide complete translation for LEF syntax.

This section describes the process of importing LEF data to create a Milkyway reference library and contains the following subsections:

- Creating a Milkyway Library Using read_lef
- Creating a Milkyway Library Using read_lib
- Mixed Flows: Technology File and LEF or LEF and GDSII
- Generated Files
- Comparison Between auNLIApi and read_lef

Figure 4-1 shows an overview of the Milkyway library preparation flow using `read_lef` and `read_lib`. Importing LEF data to create a Milkyway reference library is a five-step process.

Figure 4-1 Flow for Creating a Milkyway Library From LEF



Overview of read_lef

The `read_lef` command (Cell Library > LEF In) automates the five-step flow required to create a ready for place and route Milkyway reference library from LEF data. The `read_lef` command automates LEF importing for standard and nonstandard cells. To begin LEF importing, using `read_lef`, see "Step 1: Importing LEF Files" on page 4-6.

Note:

The `read_lef` command has replaced the `auNLIApi` command. The `auNLIApi` command is no longer supported.

Figure 4-2 shows the `read_lef` flow for creating a Milkyway library for standard cells. The `read_lib` advanced flow follows the same five steps; however, you must manually perform each step.

Figure 4-2 read_lef and read_lib Flow for Standard Cells

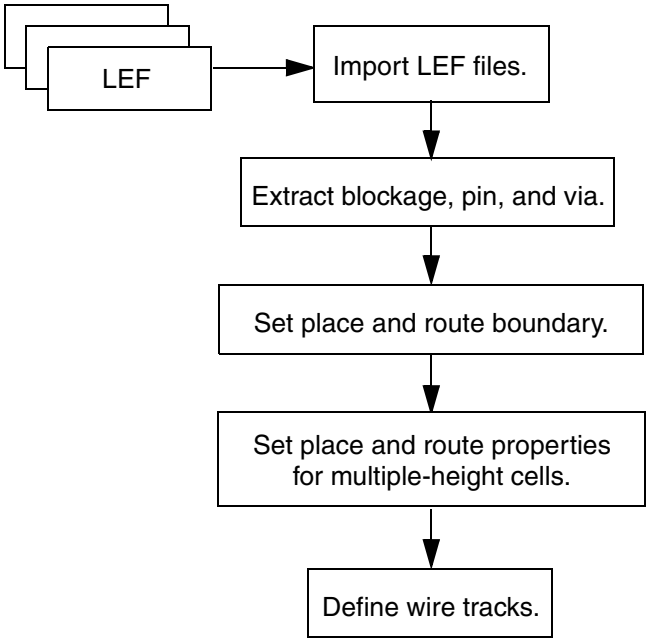
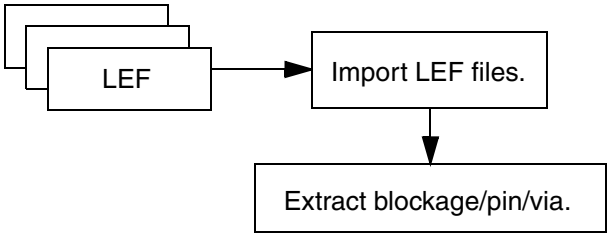


Figure 4-3 shows the `read_lef` flow for nonstandard cells using a two-step automated process. Note that the `read_lef` flow runs through *all* steps shown in Figure 4-2. Using the automated `read_lef` flow for nonstandard cells, you can ignore any error messages that might be reported in the last three steps. The `read_lib` advanced flow uses the same two steps (Import LEF files and Extract blockage, pin, and via) for nonstandard cells. However, you must manually perform the steps.

Figure 4-3 read_lef and read_lib Flow for Nonstandard Cells



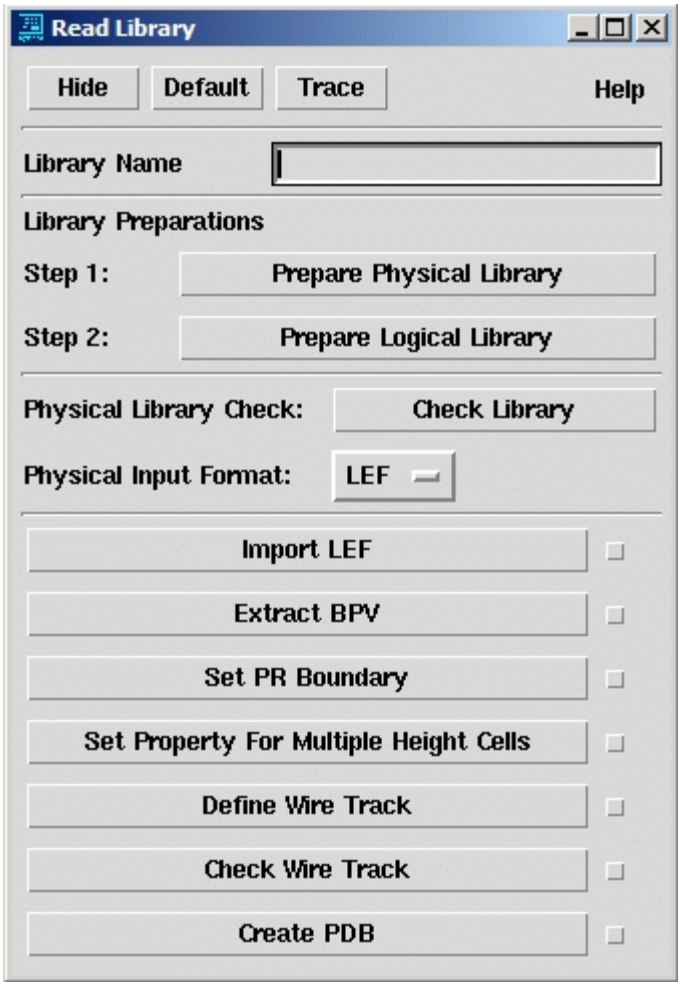
Overview of read_lib

The `read_lib` command provides steps for manually importing LEF data to create a Milkyway reference library that is ready for place and route. Use the `read_lib` advanced flow if you do not want to use the default settings or if you need to run a different sequence of steps.

To use the advanced flow to create a Milkyway library, you start by entering `read_lib` on the Milkyway command line. This invokes the Read Library dialog box, shown in Figure 4-4.

Alternatively, you can select the Advanced Library Prep Mode option in the Read LEF dialog box, shown in Figure 4-5. After Milkway imports the LEF data, the Read Library dialog box opens with the first step (Import LEF) selected, indicating that the first step is complete. You can now use the Read Library dialog box to manually complete the remaining steps, as shown in Figure 4-4. The steps for using `read_lib` are covered in "Creating a Milkyway Library Using read_lib" on page 4-13.

Figure 4-4 Read Library Dialog Box



Note:

The Create PDB step is used in Physical Compiler and is not covered in this chapter.

Creating a Milkyway Library Using read_lef

The `read_lef` command automatically runs all five steps required to create a Milkyway reference library ready for place and route.
Step 1 in the following section guides you through the initial step that automates the library preparation flow for standard and nonstandard cells. (To run the `read_lib` advanced flow, see “Creating a Milkyway Library Using read_lib” on page 4-13.)

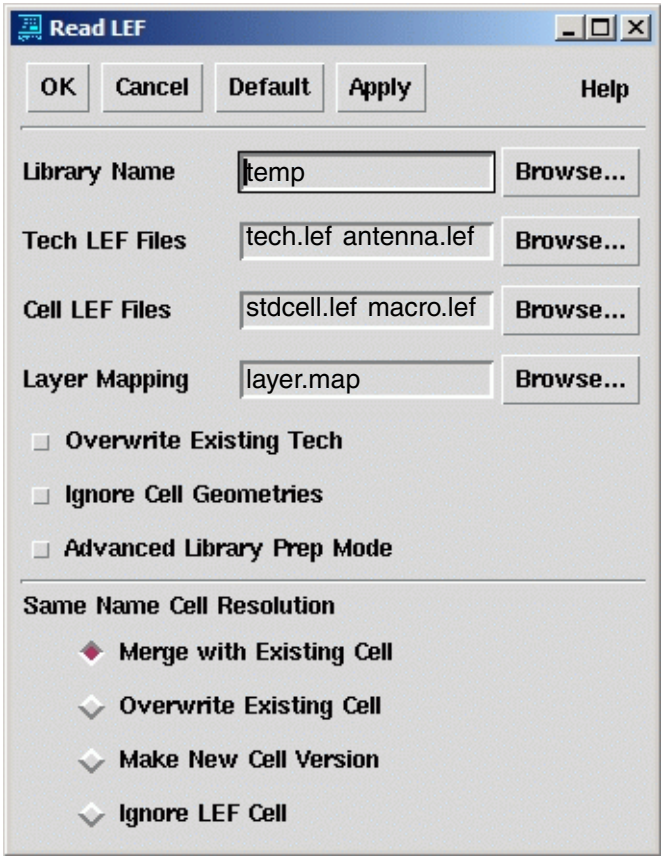
Step 1: Importing LEF Files

The steps for importing LEF files, using the `read_lef` command, are as follows:

1. Run the `read_lef` command, by entering `read_lef` on the command line or by choosing Cell Library > LEF In in the GUI.

   When you invoke the Read LEF dialog box, you must first specify the library name. For example, Figure 4-5 shows the Read LEF dialog box with a library named temp.

Figure 4-5 Read LEF Dialog Box



You do not need to have an existing library to perform `read_lef`. If you do not have an existing library, the `read_lef` command creates a new reference library based on the LEF information.

If a library already exists, you can perform an incremental LEF import. During an incremental LEF import, all LEF files must have the same version number if you are using LEF version 5.4 or later.

Note:

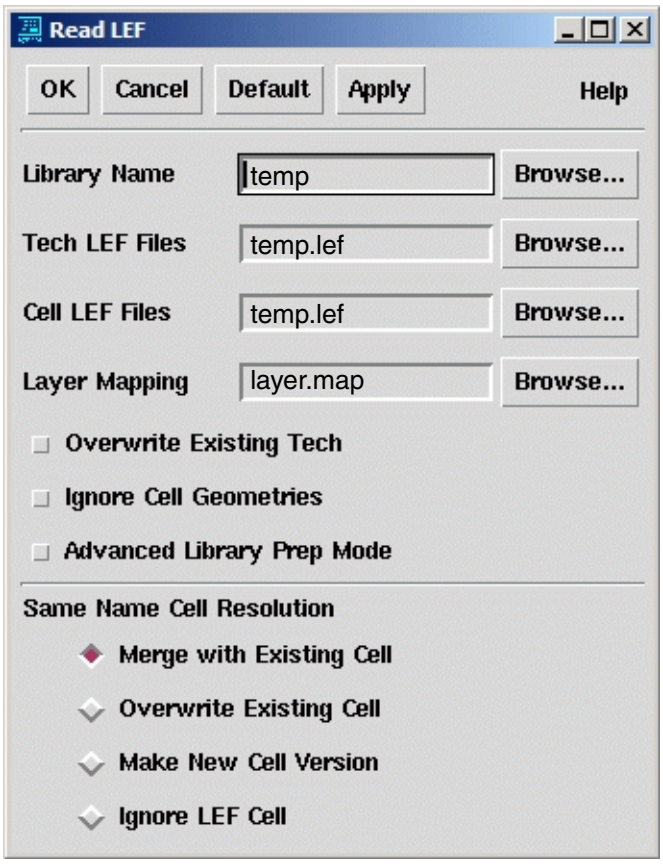
If you select Advanced Library Prep Mode in the Read LEF dialog box and click Apply, Milkyway opens the Read Library dialog box. When you click OK in the Read LEF dialog box, Milkyway completes the first step (Import LEF) and the status check box next to Import LEF in the Read Library dialog box is selected. You can now run the LEF import advanced flow manually, beginning with step 2 (Extracting Blockage, Pin, and Via). For more information, see "Creating a Milkyway Library Using read_lib" on page 4-13.

2. (Optional) Specify the LEF files in the Tech LEF Files and Cell LEF Files boxes. Keep in mind that there can be multiple methods for reading in LEF files. For example, you can

   - Import a single LEF file with the technology and cell information
   - Import multiple LEF files for technology and cell information

   If you import a single LEF file with all the technology and cell information, you should specify the same LEF file in the Tech LEF Files and Cell LEF Files boxes in the Read LEF dialog box. That is, if the LEF file is called temp.lef in the Tech LEF Files box, specify temp.lef in the Cell LEF Files box as well, as shown in Figure 4-6.

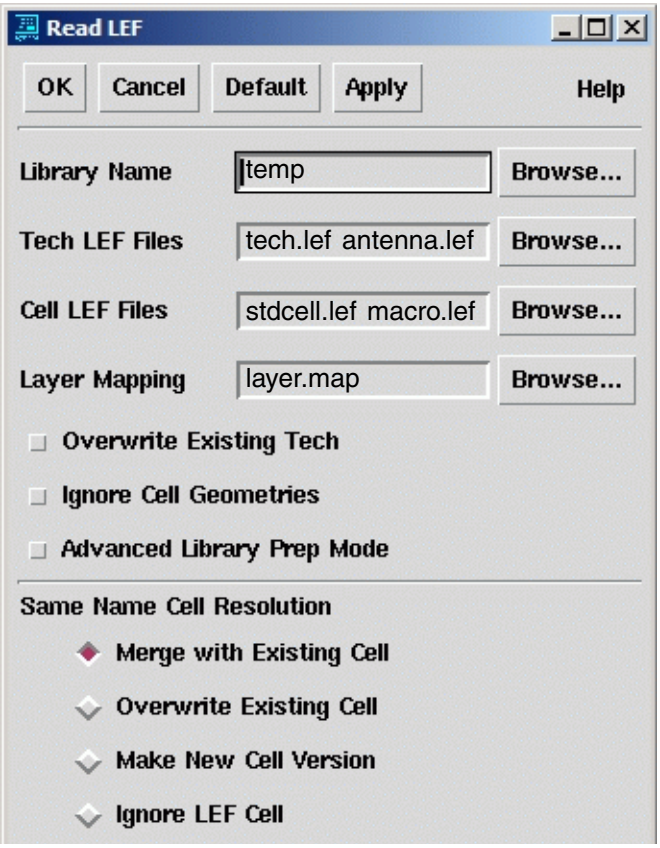
Figure 4-6 Read LEF for Single Tech and Cell LEF Files



If you have multiple LEF files for technology information (tech.lef, antenna.lef) and multiple LEF files for cell information (stdcell.lef, macro.lef), you must specify multiple files in the Tech LEF Files section and multiple files in the Cell LEF Files section.

For example, you should specify tech.lef and antenna.lef in the Tech LEF Files box and stdcell.lef and macro.lef in the Cell LEF Files box, as shown in Figure 4-7. Use a blank space or comma as a separator when specifying multiple files.

Figure 4-7 Read LEF for Multiple Tech and Cell LEF Files



Note that only the technology section from the files specified in the Tech LEF Files box is used. The cell-level information is not used. Similarly, only the cell-level information from the files specified in the Cell LEF Files box is used. The technology information from the files specified in the Cell LEF Files box is not used.

3. Use the Layer Mapping box to specify a layer mapping file that contains the LEF layer name and Milkyway layer number.

   Although specifying the layer mapping file is optional, it is strongly recommended. If no file is specified, Milkyway follows its internal default order.

   For example, Milkyway follows the LEF layer order with the first layer in LEF mapping to layer 1 in Milkyway and the second layer in LEF mapping to layer 2 in Milkyway, and so on.

   You can use the lef_layer_tf_number_mapper.pl script as follows to create a layer mapping file. For a current version of this Perl script see the SolvNet article “Using the lef_layer_tf_number_mapper.pl script.

   ```
   lef_layer_tf_number_mapper.pl tech_file_name
   lef_file_name
   ```

For example, if gold_x.tf and test_y.lef are the technology file (tf) and LEF files, respectively, run this script at the shell prompt:

```
% lef_layer_tf_number_mapper.pl gold_x.tf test_y.lef
```

As a result, the test_y_gold_x.log and test_y_gold_x.map files are generated.

The following is an example of a layer mapping file:

```
LEF layer name  Mikyway Layer number
METAL1           10
VIA12            11
METAL2           20
VIA23            21
METAL3           30
VIA34            31
METAL4           40
```

It is recommended that you provide a complete mapping of all LEF layers to the corresponding Milkyway layer number. Without this mapping, Milkyway assigns default layer numbers for you.

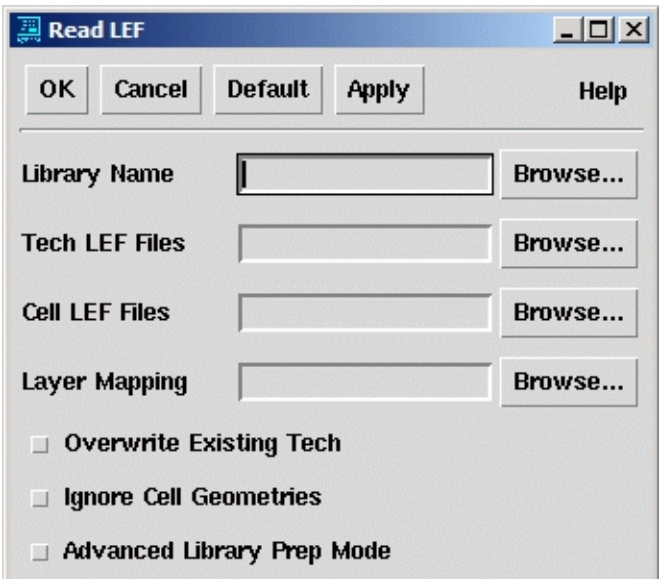
4. Click OK.

   Only the first step (invoking the Read LEF dialog box and specifying the library name) is required to create a library. The Read LEF options are described in the following section.

read_lef Options

The `read_lef` dialog box is divided into two sections: the main options and the Same Name Cell Resolution options. Figure 4-8 shows the main options, followed by a brief description of each one.

Figure 4-8 Read LEF Dialog Box: Main Options



Library Name (Required)

Specifies the Milkyway library in which the library information and reference cells will be created. The string can include the path of the library.

If no path is specified, the library is resolved from the current working directory. The `read_lef` command creates a new reference library based on LEF information if you do not have an existing library.

Tech LEF Files (Optional)

Specifies the Tech LEF input files. You can also specify multiple LEF files if, for example, you have separate LEF files for technology information and antenna information.

Cell LEF Files (Optional)

Specifies the Cell LEF input files. You can also specify multiple LEF files if, for example, you have separate LEF files for standard cells and macros.

Layer Mapping (Optional)

Specifies a layer mapping file that contains the LEF layer name and Milkyway layer number. If you do not specify this file, Milkyway follows the LEF layer order with the first layer in LEF mapping to layer 1 in Milkyway and the second layer in LEF mapping to layer 2 in Milkyway, and so on. Specifying a layer mapping file is recommended.

Overwrite Existing Tech (Optional)

This option is *not* selected by default. When you select this option, all the existing technology information from the library is removed and the new technology information from the Tech LEF Files is used instead.

Ignore Cell Geometries (Optional)

By default, this option is *not* selected. When you select this option, the pin, rectangle, polygon, and via objects are not created. After LEF input, you must input the GDSII data to create the physical information. This option is typically used for the Tech LEF + Physical Cell GDS flow.

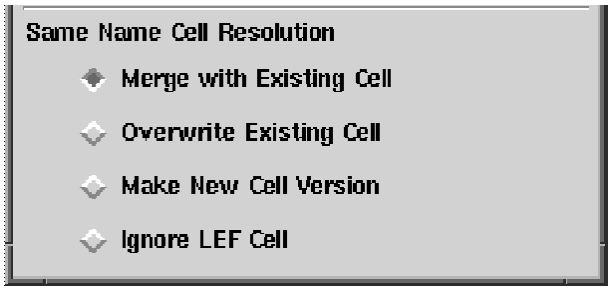
Advanced Library Prep Mode (Optional)

By default, this option is *not* selected. In the default mode, `read_lef` automatically completes all library preparation steps and creates a Milkyway library ready for place and route.

If this option is selected, you must manually run the steps, such as Extract BPV, Set PR Boundary, Set Property for Multiple Height Cells, and Define Wire Track, that follow Import LEF to create a Milkyway Library. Use this option for special cases or for advanced library preparation.

Figure 4-9 shows the Same Name Cell Resolution options, followed by a brief description of each option.

Figure 4-9 Read LEF Dialog Box: Same Name Cell Resolution Options



Merge with Existing Cell

By default, this option is selected. When selected, the latest cell version will include the information defined in the LEF file and the existing database information. Cell-related information such as macro pin and macro obstructions can be appended to the existing information.

Overwrite Existing Cell

By default, this option is *not* selected. If this option is selected, the latest cell versions will be overwritten by the LEF macro cells.

Make New Cell Version

By default, this option is *not* selected. If this option is selected, the new cell versions will be created by the macro cells defined in LEF. The old version will be maintained but a new version based on the LEF information will be created.

Ignore LEF Cell

By default, this option is *not* selected. If this option is selected, macros defined in the LEF file with the same name will be ignored when you import the LEF file.

Creating a Milkyway Library Using read_lib

You can manually run the five-step flow to create a Milkyway reference library by running the `read_lib` command. Use the manual flow if the `read_lef` automated flow does not work for your library or if you want to change the default settings used in `read_lef`. Perform the following five steps to run the `read_lib` advanced flow.

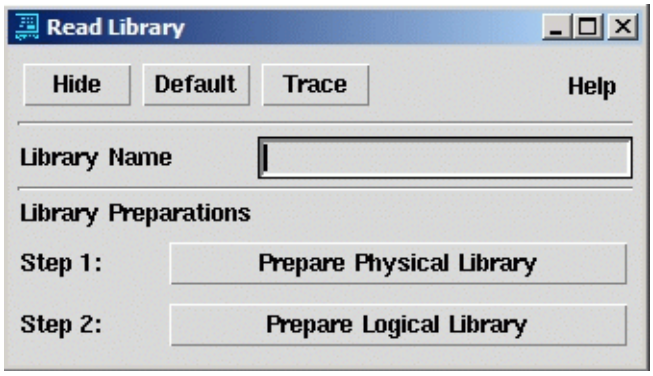
Note that if you select the Advanced Library Prep Mode option in the Read LEF dialog box and click Apply, Milkyway opens the Read Library dialog box. When you click OK in the Read LEF dialog box, Milkyway completes the first step (Import LEF) and the status button next to Import LEF in the Read Library dialog box is selected. You can now run the LEF import advanced flow manually, beginning with step 2 (Extracting Blockage/Pin/Via). You must complete the remaining steps (2 through 5) in the `read_lib` advanced flow to create a Milkyway reference library.

Note:

Step 1 (importing LEF files) has the same default options for `read_lib` as it does for `read_lef`.

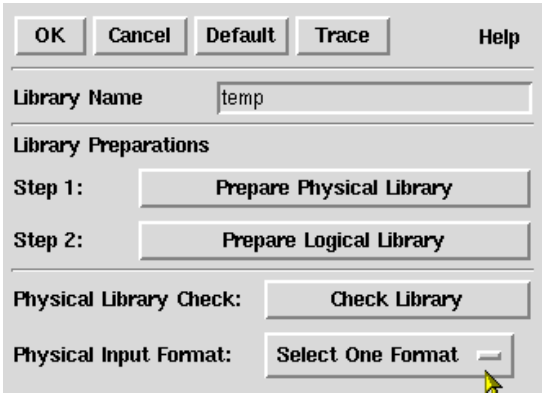
Step 1: Importing LEF Files

1. Invoke the Read Library dialog box, by entering `read_lib` on the Milkyway command line. This command guides you through the steps required to create a Milkyway library ready for place and route.



2. Specify the library name in the Library Name box.
3. Click Prepare Physical Library.

4. Click the Select One Format menu, and choose LEF.



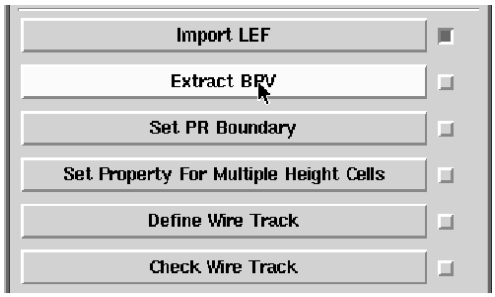
   The library preparation option settings appear at the bottom of the dialog box, as shown in Figure 4-4 on page 4-5.

5. Click OK.

Step 2: Extracting Blockage, Pin, and Via and Generating FRAM View

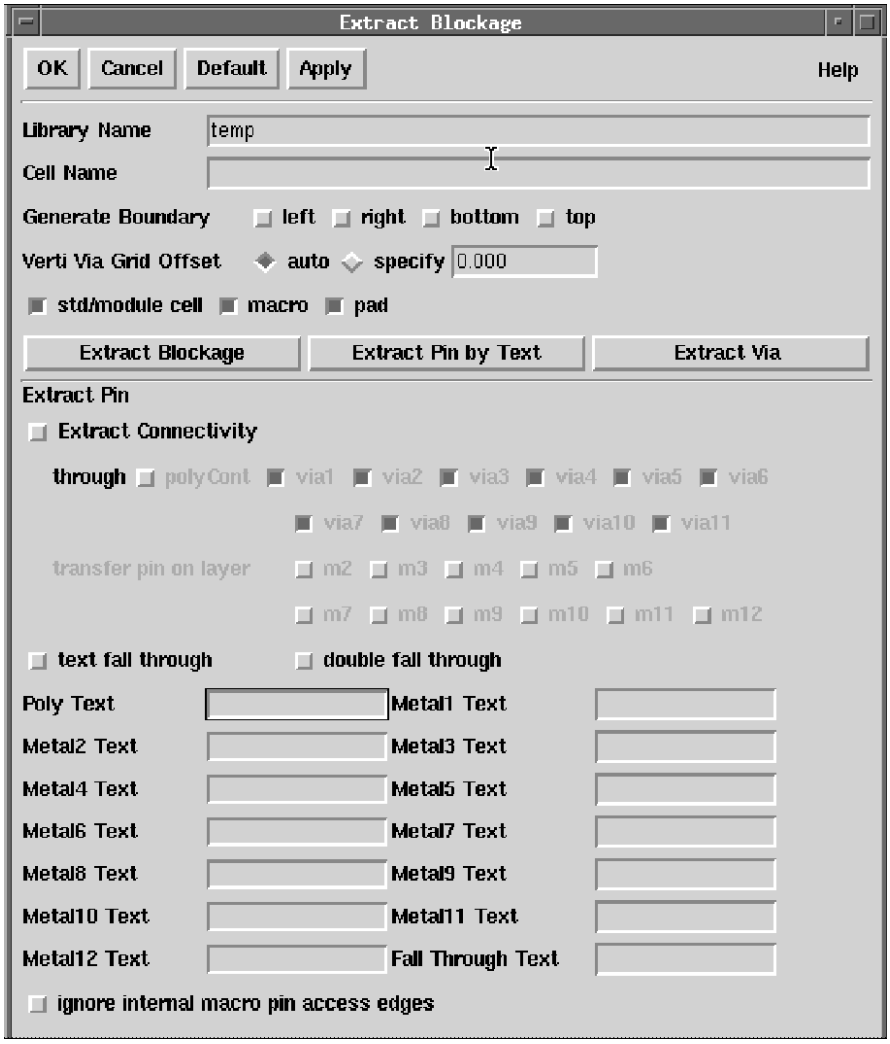
The `read_lef` command automates the steps required to create a ready for place and route Milkyway library. However, if you are using the `read_lib` advanced flow, you must manually run steps 2 through 5. This allows you to change the default options for each step.

1. Run `auExtractBlockagePinVia`, and generate a FRAM view (read_lib > Extract BPV) on the current Milkyway reference library to extract blockage, pin, and via.



   Selecting read_lib > Extract BPV brings up the Extract Blockage dialog box, shown in Figure 4-10. Electrical equivalence for pins (EEQ) information is extracted, based on the EEQ information provided from the LEF file.

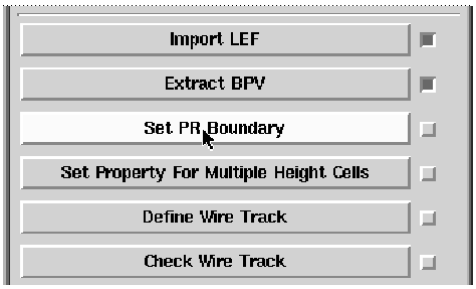
Figure 4-10 Extract Blockage Dialog Box



2. Specify the library name if it is not provided.
3. Click OK.

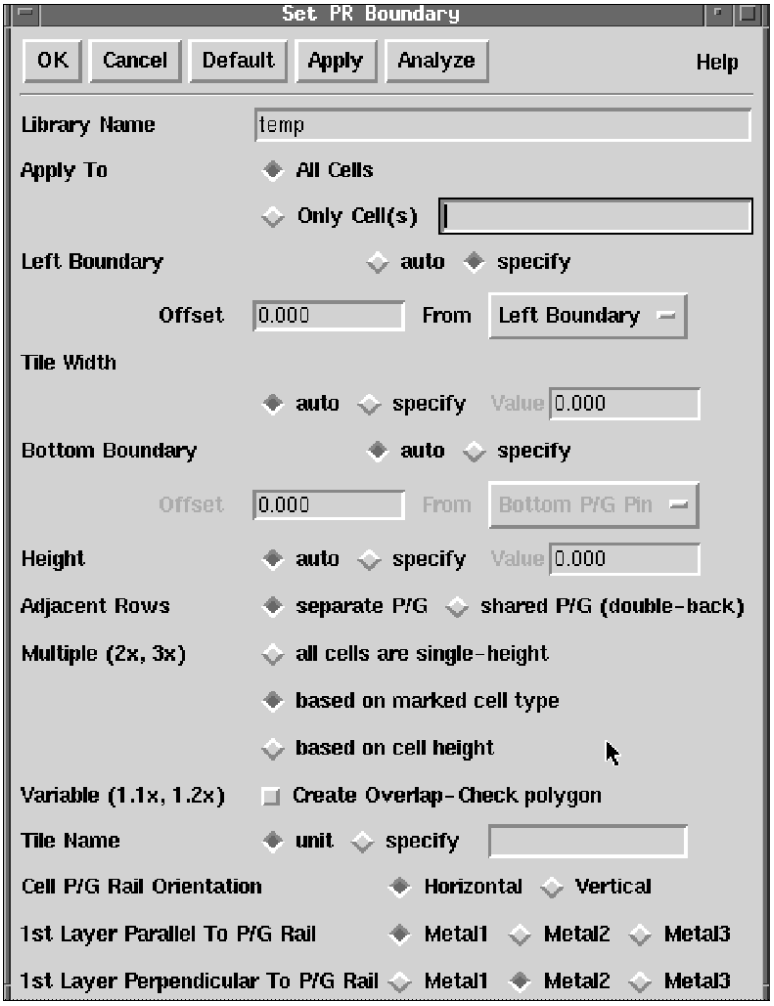
Step 3: Setting the Place and Route Boundary

1. Run the `auSetPRBdry` command (read_lib > Set PR Boundary) on the current Milkyway reference library.



   Selecting read_lib > Set PR Boundary opens the Set PR Boundary dialog box, shown in Figure 4-11.

Figure 4-11 Set PR Boundary Dialog Box

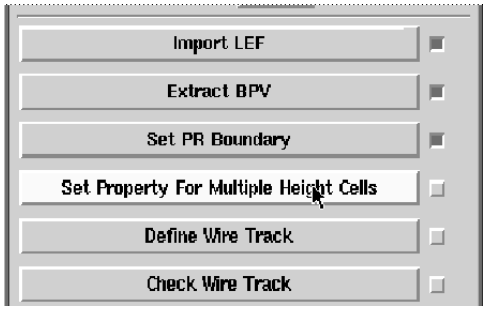


Milkyway uses “based on marked cell type” if your library has been created with LEF. The LEF format provides enough information to determine the place and route boundary. You must specify Library Name in this dialog box for the marked cell type to be effective. Once you specify the library name, you will see that “based on marked cell type” is selected in the dialog box.

2. Select the options you want.
3. Click OK.

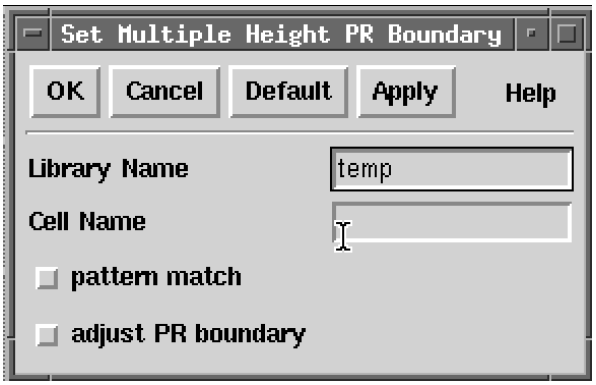
Step 4: Specifying the Multiple-Height Place and Route Boundary

1. Run the `cmSetMultiHeightProperty` command (read_lib > Set Property For Multiple Height Cells) on the current Milkyway reference library.



   This command sets place and route properties for multiple-height cells, shown in Figure 4-12.

Figure 4-12 Set Multiple Height PR Boundary Dialog Box



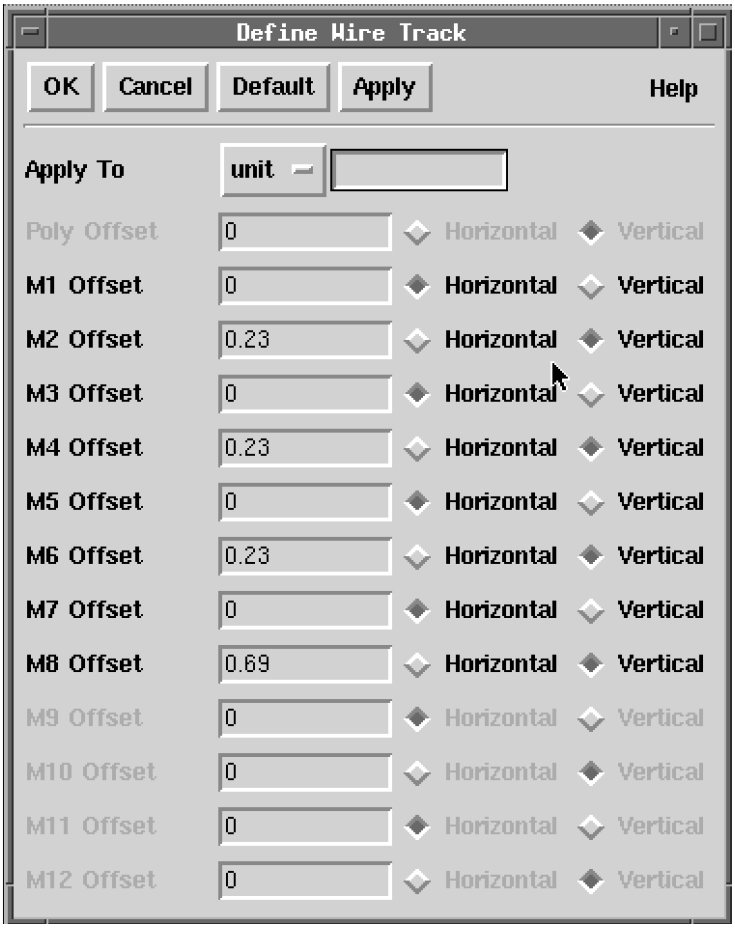
2. Click OK.

Step 5: Define Wire Tracks

1. Run the `axgDefineWireTracks` command (read_lib > Define Wire Track) on the current Milkyway reference library.

   Selecting read_lib > Define Wire Track opens the Define Wire Track dialog box, shown in Figure 4-13.

Figure 4-13 Define Wire Track Dialog Box

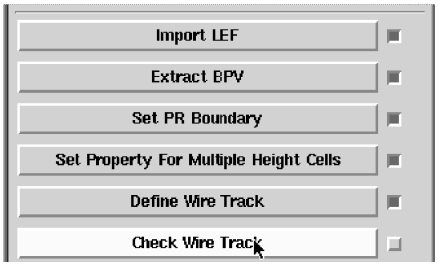


Note that the offsets and the metal directions are derived from the input LEF file.

2. Click OK.

Step 6: (Optional) Check Wire Tracks

1. Run the `axgCheckWireTrack` command (read_lib > Check Wire Track) on the current Milkyway reference library.
   Selecting read_lib > Check Wire Track runs the check.



   Note:

   The `read_lef` command in the default mode does not run this step. Therefore, if you need to check your wire tracks, you must manually run this step. Figure 4-14 shows the Check Wire Track dialog box.

Figure 4-14 Check Wire Track Option

   If you see pins without a desirable access point on the Grid, redo Define Wire Track (modify the outing layer's offset). Note that it is acceptable to have a small percentage of pins that are not accessible. (Note that a higher percentage of inaccessible pins can result in the router's taking longer to complete the design.)

2. Click OK.

Mixed Flows: Technology File and LEF or LEF and GDSII

For mixed flows, use the Read LEF dialog box and select the appropriate options. For more information on the `read_lef` options, see "read_lef Options" on page 4-10 and also Chapter 4, "Processing the Cells."

If you have a golden technology file and a macro LEF file, follow these steps to create a library:

1. Create a Milkyway library, using the golden Milkyway technology file.
2. Invoke the Read LEF dialog box, and specify the current library name in the Library Name box.
3. Import the LEF files, using `read_lef` to set the appropriate command options. You should leave the Tech LEF Files box blank and specify the Layer Mapping File.
4. Click OK.

If you have a Tech LEF file and a GDSII file for cell geometries, follow these steps:

1. Invoke the Read Library dialog box, by entering `read_lib` on the Milkyway command line.
2. Click Prepare Physical Library.
3. In the option for Physical Input Format, click the Select One Format menu and choose LEF + GDSII, then follow the steps mentioned in the flow, using the appropriate options.
4. Click OK.

Generated Files

During the LEF input process, the following files are generated automatically and are used by other functions. This information is for reference only.

The following files are created as side files, which you can refer to after you run `read_lef`:

cell.lefdef.alias.sum

Contains a list of all alias or define statements within the LEF file.

lefFileName.alias_exp

Copy of the LEF input file with expanded alias names and define variables. Each alias name is replaced by its corresponding alias definition. Each define variable is replaced by its corresponding define expression.

libraryName.wtdSet.script

Scheme file that sets the Milkyway design cell wire track directions. This file is created from the LEF layer direction statements and is referred to during the Define Wire Track step.

lefFileName.site_def

Site Definition File containing the placement site definitions derived from the LEF site statements. This file is optionally passed to the DEF input and output functions.

The format of this file is as follows:

```
SITENAME TYPE SYMMETRY WIDTH HEIGHT
```

where valid symmetry values are

- 1 for X symmetry
- 2 for Y symmetry
- 4 for rotate 90
- 8 for rotate 180
- 16 for rotate 270
- 32 for asymmetry

lefFileName.site_info

Site Information File containing the macro placement site information derived from the LEF macro site statements.

lefFileName.defineVarRoute.sc

Scheme file that defines the Milkyway design cell variable route rules. This file is created from the LEF nondefault rule statements. It is used by the router to define variable route rules for a design.

lefPinOrder2def

List of macros and their pins that describes the macro pin order defined in the LEF file.

lefFileName.tech.clf

Scheme (CLF) file that sets the Milkyway cell library's technology antenna rules. This file is created from the LEF LAYER antenna rule statements. This file should be loaded after the design is loaded.

lefFileName.constrain.script

Scheme file that sets the Milkyway defaultGateSize of droute. This file is created from the LEF INPUTPINANTENNASIZE statement (for LEF v5.3 syntax) and should be loaded after you create a FRAM view of the cell.

Comparison Between auNLIApi and read_lef

Table 4-1 compares the old `auNLIApi` and new `read_lef` commands and lists the differences and changes between the two. (The `auNLIApi` command is no longer supported.)

Table 4-1 auNLIApi and read_lef Option Comparison

`auNLIApi` options	**`read_lef` options**
Library Name	Library Name
Specifies an opened library name.	Specifies an existing or brand-new library name. The library does not need to be open.
LEF File	Tech LEF Files Cell LEF Files
Specifies a LEF file.	Specifies two lists of LEF files: • Tech LEF files, such as technology, antenna, and varRouteRule • Cell LEF files LEF In reads only tech information from those files specified in Tech LEF Files and reads only cell information from files specified in Cell LEF Files. If there is only one LEF file, specify this file name in both the tech LEF files and cell LEF files fields.
(no equivalent option)	Layer Mapping If nothing is specified, `read_lef` designates an internal default Milkyway layer number. If a mapping file is specified, the layer number is designated by the definition in the mapping file format: layerName layerNumber /* layerName is in LEF layer section*/ /* layer number is Milkyway layer number*/ example: M1 10 VIA1 11 M2 20 ...

Table 4-1 auNLIApi and read_lef Option Comparison (Continued)

auNLIApi options	**read_lef options**
Bus Naming Style Default is blank. If specified, sets the library's busNamingStyle accordingly. Otherwise, use BUSBITCHAR in the LEF file as the library bus naming style.	(no equivalent option) Use BUSBITCHAR in the LEF file as the library bus naming style.
Cell Type Definition File Default is blank. If specified, the cell type is marked according to the map file. Otherwise, the cell type is determined by MACRO CLASS in Cell LEF files.	(no equivalent option) Cell type is determined by MACRO CLASS in Cell LEF files.
GDSII Default is off. When selected, macro geometry and obstruction in cell LEF files are ignored.	Ignore Cell Geometries Default is off. When selected, macro geometry and obstruction in cell LEF files are ignored.
annotate GDSII text Default is off. This option is available when the GDSII option is selected (pins are not created). When selected, pin texts are created. Otherwise, GDSII text is not annotated.	(no equivalent option) `read_lef` does not take any stream-in-related options.
via pin (on) Default is on: via pins are created. When turned off, via pins are not created.	(no equivalent option) `read_lef` always creates via pins.
apply library case-sensitive mode (on) Default is on. If the LEF file's NAMECASESENSITIVE is not the same as the library case-sensitive mode, the library case-sensitive value is used during LEF importing.	(no equivalent option) `read_lef` determines the library's case-sensitive mode from the LEF NAMECASESENSITIVE statement. If not specified in the LEF file, case-sensitivity is set to ON.

Table 4-1 auNLIApi and read_lef Option Comparison (Continued)

auNLIApi options	**read_lef options**
Layer Options:	
>handle overlap layer Default is off. Overlap layer is ignored.	(no equivalent option) read_lef always handles the overlap layer if it's in the LEF files.
Tech Options:	Tech Option: >Overwrite Existing Tech If selected, all existing technology information in the library is removed and new technology information is loaded in from the Tech LEF files.
>skip techfile Default is off. Library technology information is replaced by technology information defined in the LEF file.	(no equivalent option) If you want to skip technology information in the LEF file, you can specify only Cell LEF files. Even if there is technology information in the LEF file, it is ignored.
>append layers to technology Default is off. If selected, the existing layer's attributes are updated according to the LEF file. New layers defined in the LEF file are created as a dummy layer (no maskName) in the library.	(no equivalent option) Not supported.
>append vias to technology Default is off. Available when the techfile option is on. If selected, via information is imported to the library, although other technology information is skipped.	(no equivalent option) Not supported.

Table 4-1 auNLIApi and read_lef Option Comparison (Continued)

auNLIApi options	**read_lef options**
Cell options:	
>LEF TO FRAM Default is off. If selected, triggers a complete library preparation process with only default settings.	Advanced Library Prep Mode Default value: OFF In default value, after read_lef is finished, the tool automatically calls extract BPV, set PRboundary, set multiple-height property, and defineWireTrack on the library created during read_lef and creates a complete library ready for place and route. If this option is set to ON, the program stops after read_lef and you have to manually run other scheme commands such as extract BPV, and set PR boundary. This is useful for advanced library prep users.
Same cell resolution:	Same Name Cell Resolution
>revise	>Merge with Existing Cell
Updates (merges) library cell information according to the definition in the LEF file.	Updates (merges) library cell information according to the definition in the LEF file.
>overwrite	>Overwrite Existing Cell
Removes the latest-version cell and creates the cell according to the macro definition in the LEF file.	Removes the latest-version cell and creates the cell according to the macro definition in the LEF file.
>new version cell	>Make New Cell Version
Creates a new-version cell according to the macro definition in the LEF file.	Creates a new-version cell according to the macro definition in the LEF file.
>skip	>Ignore LEF Cell
Ignores the macro definition in the LEF file.	Ignores the macro definition in the LEF file.

Exporting LEF Data From Milkyway Using write_lef

This section discusses the Write LEF dialog box and the options used to export LEF data from the Milkyway environment. (For information about the Read LEF dialog box and the options used to import LEF data into the Milkyway environment, see “Library Preparation Using LEF” on page 4-2.)

The `write_lef` command outputs a LEF file with complete LEF information in a single process. LEF defines the elements of an IC process technology and associated library of cell models and contains library information for a class of designs. This library information includes layer, via, placement site type, and macro cell definitions.

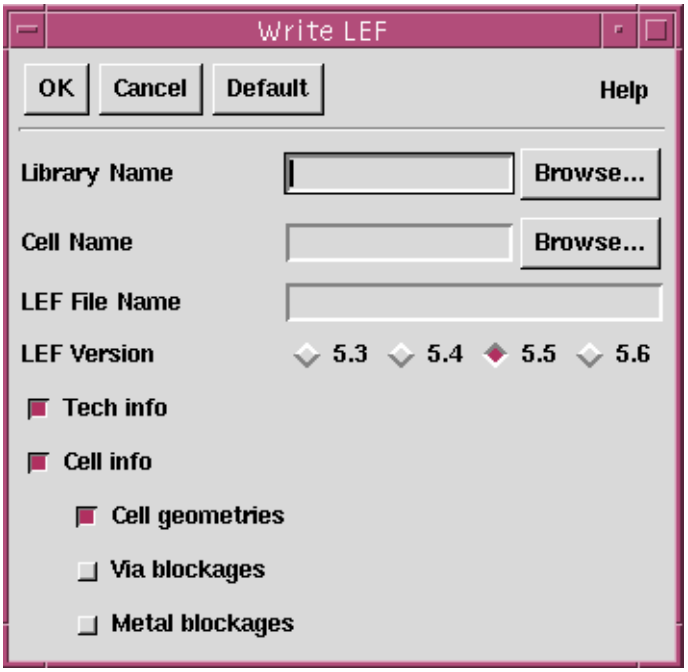
Note:
This release continues to support the `auNLOApi` command.

To open the Write LEF dialog box, enter the following on the Milkyway command line:

```
% write_lef
```

The Write LEF dialog box opens, as shown in Figure 4-15.

Figure 4-15 Write LEF Dialog Box



The Write LEF dialog box contains the following options:

Library Name (Required)

Specifies the Milkyway library that contains the design cell. The text box can include the path to the library; otherwise, the library path is resolved from the current working directory. You can also browse to the library by using the Browse button.

Cell Name (Required)

Specifies the name of the design cell. You can browse to the design cell by using the Browse button.

LEF File Name (Required)

Specifies the name of the LEF file to which the design cell will be written. If the file already exists, Milkyway overwrites it. If the file does not exist, Milkyway creates it. The text box can include the path to the output file; otherwise, the file is written to the current working directory.

LEF Version (Optional)

The default is 5.5.

The LEF interface supports versions 5.3, 5.4, 5.5, and 5.6.

Tech info (Optional)

This option is selected by default. In the default mode, Milkyway writes a technology LEF file. You can also

- Write a complete LEF file by selecting the “Tech info” option, the “Cell info” option, and the “Cell geometries” option.
- Write a LEF file with technology and cell information, but without geometry information, by selecting the “Tech info” option and the “Cell info” option and deselecting the “Cell geometries” option.

Cell info (Optional)

This option is selected by default. In the default mode, Milkyway writes a LEF file with cell information. You can

- Write a LEF file with cell information and geometries, but without technology information, by selecting the “Cell info” option and deselecting the “Tech info” option.
- Write a LEF file with technology and cell information, but without geometry information, by selecting the “Tech info” option and the “Cell info” option and deselecting the “Cell geometries” option.
- Write a LEF file with cell information but without technology information and cell geometries by selecting the “Cell info” option and deselecting the “Tech info” option and the “Cell geometries” option.

Cell geometries

This option is selected by default. When it is selected, Milkyway outputs cell geometry information. This option is not available if you output technology information only.

Via blockages

If this option is selected, Milkyway output via blockages are written to the LEF file. By default, the option is not selected and output via blockages are not written to the LEF file. This option is not available if you output technology information only.

Metal blockages

If this option is selected, Milkyway output metal blockages are written to the LEF file. By default, the option is not selected and output metal blockages are not written to the LEF file. This option is not available if you output technology information only.

The Milkyway DEF Interface

The Design Exchange Format (DEF) defines the elements of an IC design relevant to the physical layout, including the netlist and design constraints. It contains the design-specific information for a circuit and is a representation of the design at any point during the layout process.

The Milkyway DEF interface discussed in this section uses a common new DEF reader and writer to provide consistency between Synopsys tools.

Note:
The `auNDIApi` and `auNDOApi` commands are no longer supported.

The DEF interface supports versions 5.3, 5.4, 5.5, and 5.6. This section includes the following subsections:

- DEF Import and Export Commands
- Importing DEF Data Into Milkyway Using read_def
- Exporting DEF From Milkyway Using write_def
- Recommended DEF Flows
- Supported DEF 5.6 Syntax

DEF Import and Export Commands

The following commands read and write in the DEF interface:

- `read_def`

Imports (reads) a DEF file to a Milkyway design cell.

- `write_def`

  Exports (writes) a Milkyway design cell to a DEF file.

The `read_def` and `write_def` commands are available in both Scheme and Tcl modes. For more information about Scheme and Tcl modes, see the *Milkyway Environment Extension Language Reference Manual*. The details of these commands are described later in this section.

DEF files compressed in gzip format (*.gz) are supported in the DEF interface. The `read_def` command does not have a gzipped option but supports files with the *.gz extension. The `write_def` command provides an option that, if selected, outputs DEF in gzipped format.

The DEF reader and writer are designed to work in Tcl mode. This section covers the Tcl DEF import and export interface commands.

Importing DEF Data Into Milkyway Using read_def

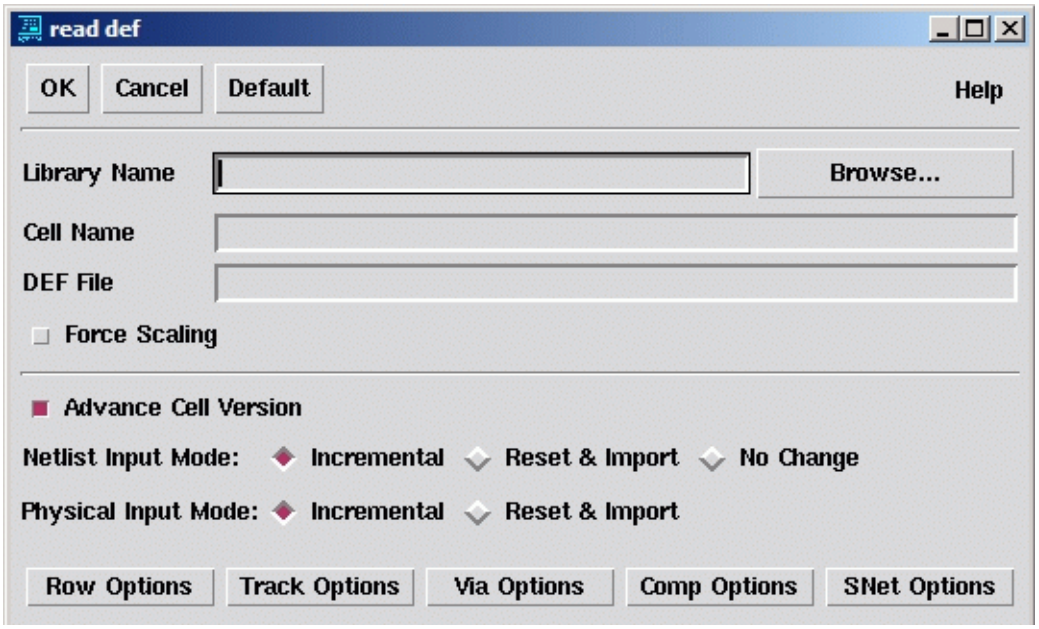
This section discusses the “read def” dialog box and the options used to import DEF data to the Milkyway environment. For information about the “write def” dialog box and the options used to export DEF data from the Milkyway environment, see “Exporting DEF From Milkyway Using write_def” on page 4-38.

To open the “read def” dialog box, enter the following on the Milkyway command line:

```
Milkyway > read_def
```

The “read def” dialog box opens, as shown in Figure 4-16.

Figure 4-16 read def Dialog Box



The “read def” dialog box contains the following options:

Library Name (Required)

Specifies the name of the library to which the design information is loaded.

Cell Name (Required)

Specifies the name of the cell to annotate with physical data.

DEF File (Required)

Specifies the DEF input file. You can specify single or multiple DEF files. Use a blank space as a separator when specifying multiple files. By default, multiple DEF files are imported in incremental mode.

Important:

The order in which you enter multiple DEF files in the DEF File text box is important because DEF syntax defined in previous sections is sometimes used in later sections. For example, suppose your first DEF file (a.def) includes syntax for the REGION section, and your second DEF file (b.def) includes syntax for the GROUP section. If the second file includes [+ REGION regionName] syntax in the GROUP section, the correct file order is a.def and then b.def. However, if the second file (b.def) does not include [+ REGION regionName] syntax, you can enter a.def and b.def in either order.

Note:

The `read_def` command supports the gzip format (*.gz). There is no explicit option in the “read def” dialog box for gzip support. But if the DEF File box has input with a *.gz extension, such as my_gizpd_input.def.gz, Milkyway determines that the input DEF file is gzipped. It then automatically unzips the file and begins to read in the DEF file.

Force Scaling (Optional)

If this option is selected, `read_def` enforces scaling if the Milkyway database precision is not a multiple of the input DEF unit. If not selected, `read_def` quits with an error if the Milkyway database precision is not a multiple of the input DEF unit. This option is not selected by default.

Advance Cell Version (Optional)

If this option is selected, the most recent Milkyway design cell is increased by the input DEF file. This option is selected by default.

The `read_def` command provides five netlist and physical input mode options to give better support for Verilog plus DEF input flow as well as pure DEF input flow.

Netlist Input Mode: Incremental; Physical Input Mode: Incremental

Mode 1: `-netl_phys incr_incr`

In this mode, `read_def` incrementally reads the netlist and physical layout information and issues information when a netlist object is added.

Netlist Input Mode: Incremental; Physical Input Mode: Reset & Import

Mode 2: `-netl_phys incr_rimport`

In this mode, `read_def` incrementally reads the netlist information and resets and imports the physical layout information. It also issues information when a netlist object is added.

Netlist Input Mode: Reset & Import; Physical Input Mode: Reset & Import

Mode 3: `-netl_phys rimport_rimport`

In this mode, `read_def` resets and imports both netlist and physical layout information; however, it does not issue warnings or information.

Netlist Input Mode: No Change; Physical Input Mode: Incremental

Mode 4: `-netl_phys nochange_incr`

In this mode, `read_def` drops new netlist objects and issues warnings. It also incrementally reads physical layout information.

Netlist Input Mode: No Change; Physical Input Mode: Reset & Import

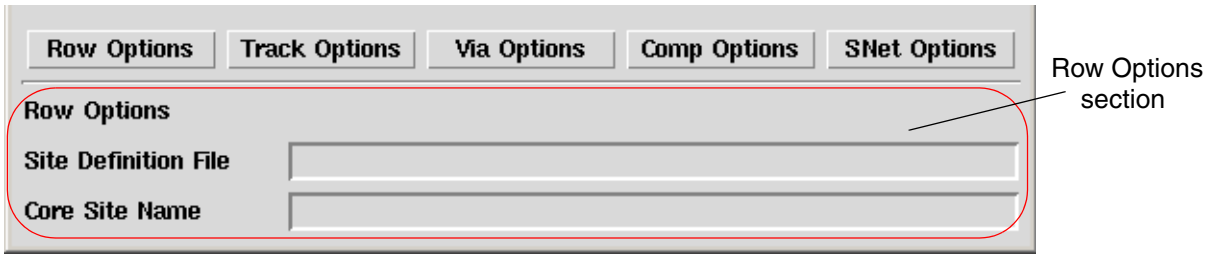
Mode 5: `-netl_phys nochange_rimport`

In this mode, `read_def` drops new netlist objects and issues warnings. It also resets and imports physical layout information.

Row Options

Clicking the Row Options button opens the Row Options section, shown in Figure 4-17.

Figure 4-17 read_def Row Options



The Row Options section contains the following options:

Site Definition File (Optional)

This file contains the core and pad site definitions and is generated during DEF input from the site statements. Sites are created in the Milkyway design cell with these definitions.

If the site definition file is not specified, sites are created in the design cell with sites and tiles in the library. Searching begins in the top library and proceeds through the library reference hierarchy.

If the library has no site definition, the file should be provided in the Site Definition File box. In the normal LEF/DEF flow, this option is not required, because the site information is defined in the DEF file.

Core Site Name (Optional)

Specifies the name of the core site used in the design. This is an optional argument. If a name is not specified, the site specified in the input DEF file is used.

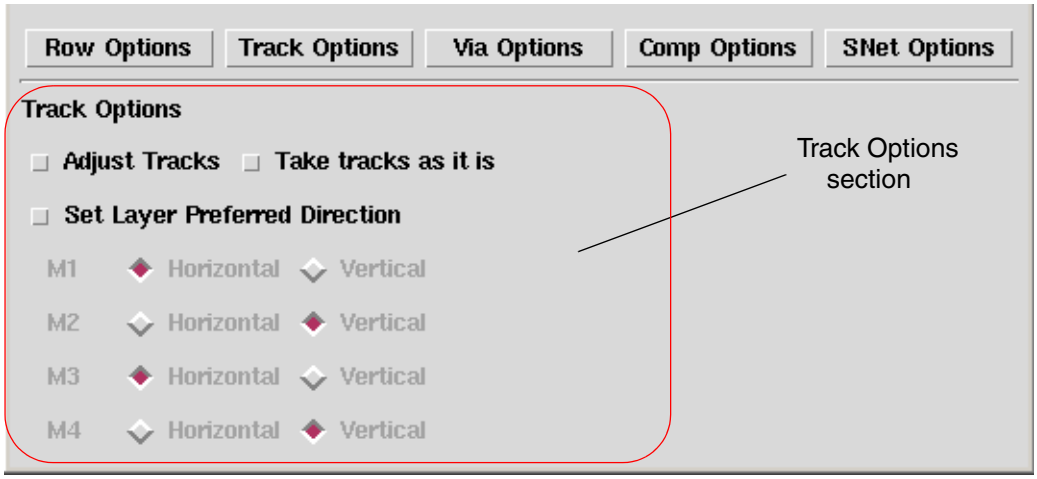
If the Site Definition File box is empty and the name in the Core Site Name box is “core” (which implies that the design library or the reference library has a site named “core”), Milkyway creates rows according to these specified options.

If the library has a tile named “core,” Milkyway uses the “core” tile. Otherwise, Milkyway finds and uses the “unit” tile. In all other cases, Milkyway fails to create rows.

Track Options

Clicking the Track Options button opens the Track Options section, shown in Figure 4-18.

Figure 4-18 read_def Track Options



The Track Options section contains the following options:

Adjust Tracks

If the spacing between the tracks is less than the minimum width plus the minimum spacing of the routing layer, `read_def` automatically adjusts the track spacing.

Take tracks as it is

Takes tracks as they are defined in the DEF file.

Set Layer Preferred Direction

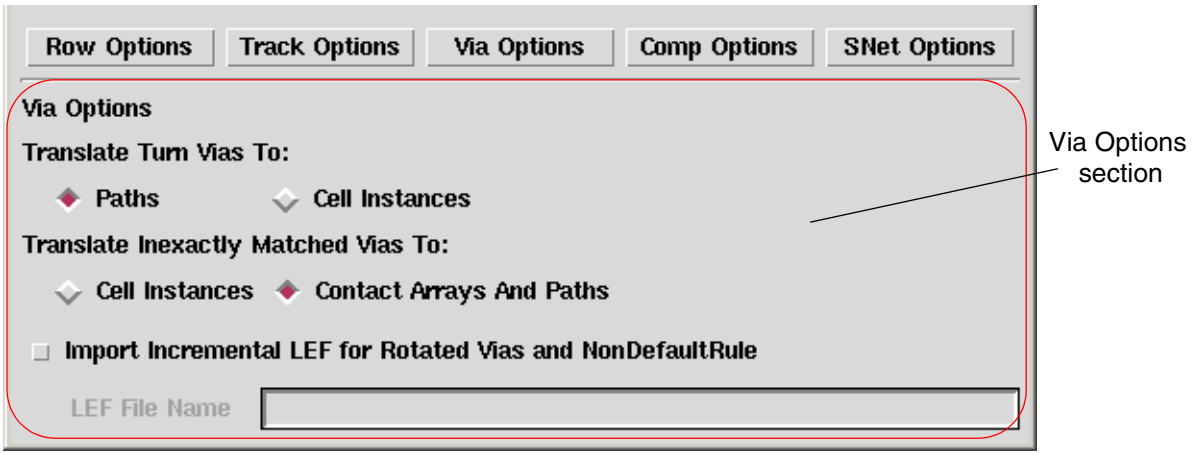
If this option is selected, the wire track direction is set for metal layers M1, M2, M3, and M4. This option is not selected by default.

Tracks of metal layers higher than M4 are set to alternating directions with respect to the M4 layer.

Via Options

Clicking the Via Options button opens the Via Options section, shown in Figure 4-19.

Figure 4-19 read_def Via Options



For each definition in the DEF via section, if a match to a Milkyway technology contact code is found, the via is translated into the design as either a contact or a contact array. If the match to any technology contact code is not found, it is translated as a via cell instance.

To judge whether a via matches a contact code, Milkyway attempts to match the via cell to any contact code, using 0- or 90-degree rotations to try to find the best match.

Milkyway checks simple vias and via arrays differently. A simple via (single cut rectangle) is checked by name or by its cut dimensions and enclosure dimensions. A via array (multiple cut rectangles) is checked by its cut dimensions and enclosure dimensions.

Reasons for not matching include the following:

- The via cell can’t be recognized (stack via cell).
- The via cut dimensions do not match any contact code’s cut dimensions.
- The via cut spacing is smaller than any contact code’s minimum cut spacing.
- The via lower enclosure is smaller than any contact code’s lower-layer enclosure requirements.
- The via upper enclosure is smaller than any contact code’s upper-layer enclosure requirements.

There are two exceptions to the above rules for turn vias and inexactly matched vias. Use the following options for these exceptions.

Translate Turn Vias To

Default: Paths selected. Specifies how turn vias are translated. Turn vias are represented in DEF as a single rectangle (usually a piece of dangling metal).

- Paths

  Turn vias are created as path objects. The vias are no longer marked as turn vias. This option is selected by default.

- Cell Instances

  Turn vias are created as via cell instances. The vias remain marked as turn vias. This option is not selected by default.

Translate Inexactly Matched Vias To

Default: Contact Arrays And Paths selected. Specifies how Milkyway translates inexactly matched vias. Inexactly matched vias are vias that match a Milkyway technology contact code's cut dimensions. However, for inexactly matched vias, the enclosure dimensions are larger than the contact code's enclosure dimensions.

- Cell Instances

  If this option is selected, inexactly matched vias are created as via cell instances. This option is not selected by default.

- Contact Arrays And Paths

  If this option is selected, inexactly matched vias are created as contact arrays and extra wires on the enclosure layers. The extra wires are necessary to meet the via's enclosure requirements. This option is selected by default.

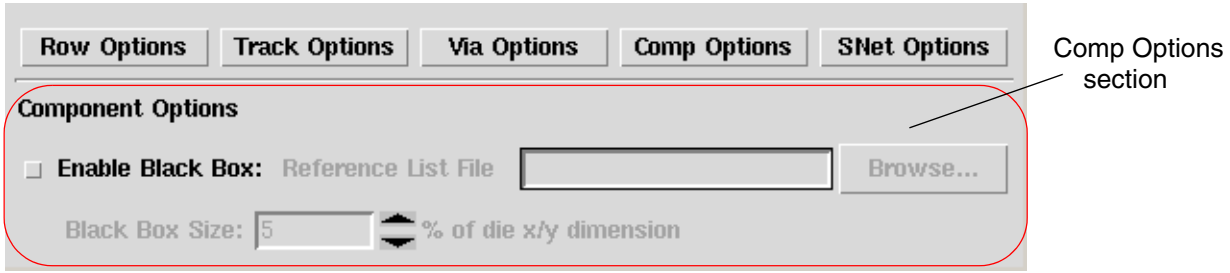
- Import Incremental LEF for Rotated Vias and NonDefaultRule

  If this option is selected, input the LEF file name containing the rotated vias and design specific nondefault rule vias. This option is not selected by default.

Comp Options

Clicking the Comp Options button opens the Component Options section, shown in Figure 4-20.

Figure 4-20 read_def Component Options



The Component Options section contains the following options:

Enable Black Box

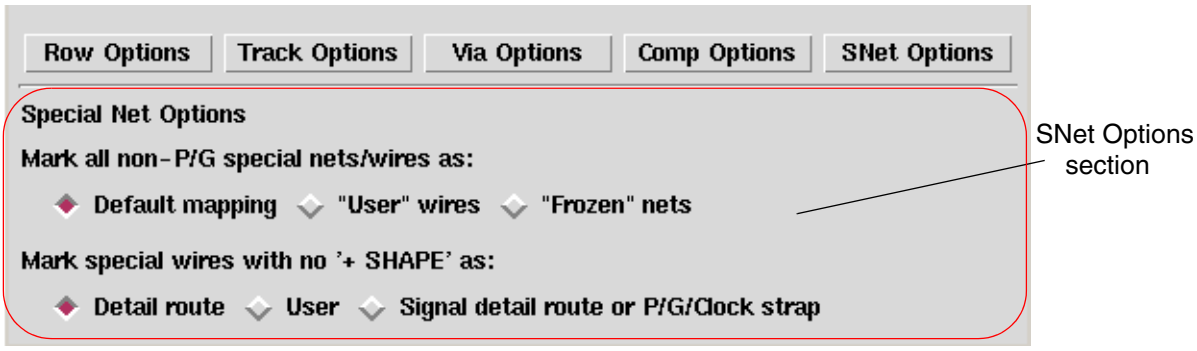
If you select the Enable Black Box option, you must enter the name of the reference list file (which contains the cell instance master names) in the Reference List File text box. Alternatively, you can browse to the reference list file by using the Browse button. This option is not selected by default.

You can also specify the percentage of die area dimension in the Black Box Size text box. The black box size is applied to each instance master. The default is 5.

SNet Options

Clicking the SNet Options button opens the Special Net Options section, shown in Figure 4-21.

Figure 4-21 read_def Special Net Options



The Special Net Options section contains the following options:

Mark all non-P/G special nets/wires as:

Specifies the type of marking to be done for special wires that are not power and ground wires.

- Default mapping

  If selected, special wires that are not power and ground will be mapped to default Milkyway route types. This option is selected by default.

- “User” wires

  If selected, all special wires that are not power and ground will be marked as user-entered route type. This is the default route type for shapes that are created by an editor. These route types are treated as “frozen” by the detail router.

- “Frozen” nets

  If selected, all special nets that are not power and ground nets will be marked as frozen nets.

Mark special wires with no “+ SHAPE” as

Specifies how special wires with no “+ SHAPE” attribute will be annotated in the database.

- Detail route

  If selected, all special wires with no “+ SHAPE” attribute will be marked as detail route type. This option is selected by default.

- User

  If selected, all special wires with no “+ SHAPE” attribute will be marked as user-entered route type.

- Signal detail route or P/G/Clock strap

  If selected, all power, ground, and clock special wires with no “+ SHAPE” attribute will be marked as “P/G/Clock strap.” Other special wires with no “+ SHAPE” attribute will be marked as signal detail route.

Note:

When “‘User’ wires” is selected for special wires that are not power and ground, if the marking of such special wires conflicts with that of special wires with no “+ SHAPE” attribute, the “‘User’ wires” option will overwrite the latter.

DEF Reader Semantic Checks

The `read_def` command provides semantic checks as per the *DEF Language Reference Manual*. As the DEF file parses, syntax and semantic issues are reported with information messages, warnings, and error messages, depending on the severity of the DEF statement violation. The DEF reader then processes the DEF file until it either finishes processing the input file or cannot proceed further due to a lack of required information. The DEF reader also provides more user control regarding the treatment of a logical netlist, a physical netlist, and cell versions. For more information, see Physical Implementation Online Help.

Exporting DEF From Milkyway Using write_def

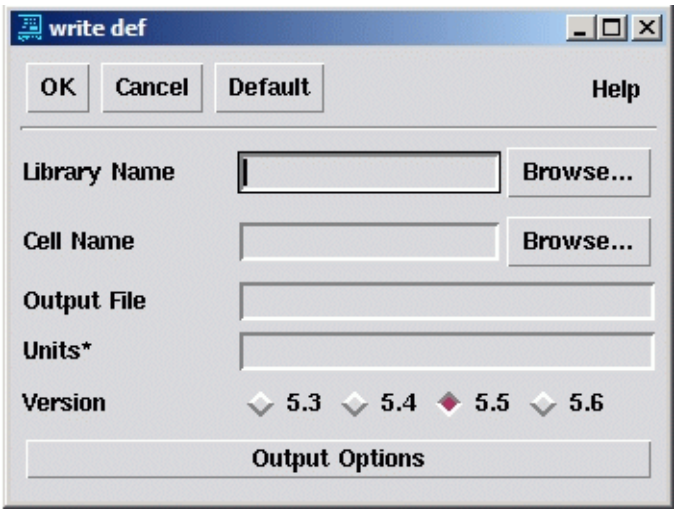
This section discusses the “write def” dialog box and options used to export DEF from the Milkyway environment. (For information about the “read def” dialog box and options used to import DEF into the Milkyway environment, see “Importing DEF Data Into Milkyway Using read_def” on page 4-30.)

To open the “write def” dialog box, enter the following on the command line:

```
Milkyway > write_def
```

The “write def” dialog box opens, as shown in Figure 4-22.

Figure 4-22 write def Dialog Box



The “write def” dialog box contains the following options:

Library Name (Required)

Specifies the Milkyway library that contains the design cell. The box can include the path of the library. Otherwise, the library path is resolved from the current working directory.

Cell Name (Required)

Specifies the name of the design cell.

Output File (Required)

Specifies the name of the DEF file to which the design cell is written. If the file already exists, Milkyway overwrites it. If the file does not exist, Milkyway creates it. The box can include the path of the output file. Otherwise, the file is written to the current working directory.

Units* (Optional)

Specifies the distance unit for the output DEF data. Valid values: 100, 200, 1000, 2000, 10000, and 20000.

For any other values, `write_def` exits with an error message and writes the default value. The default value is the length precision value of the technology file.

If no value is specified, the default value is written in the output DEF file.

Version

Specifies the DEF version to be exported. The DEF interface supports versions 5.3, 5.4, 5.5, and 5.6. Version 5.5 is selected by default.

Version Limitations

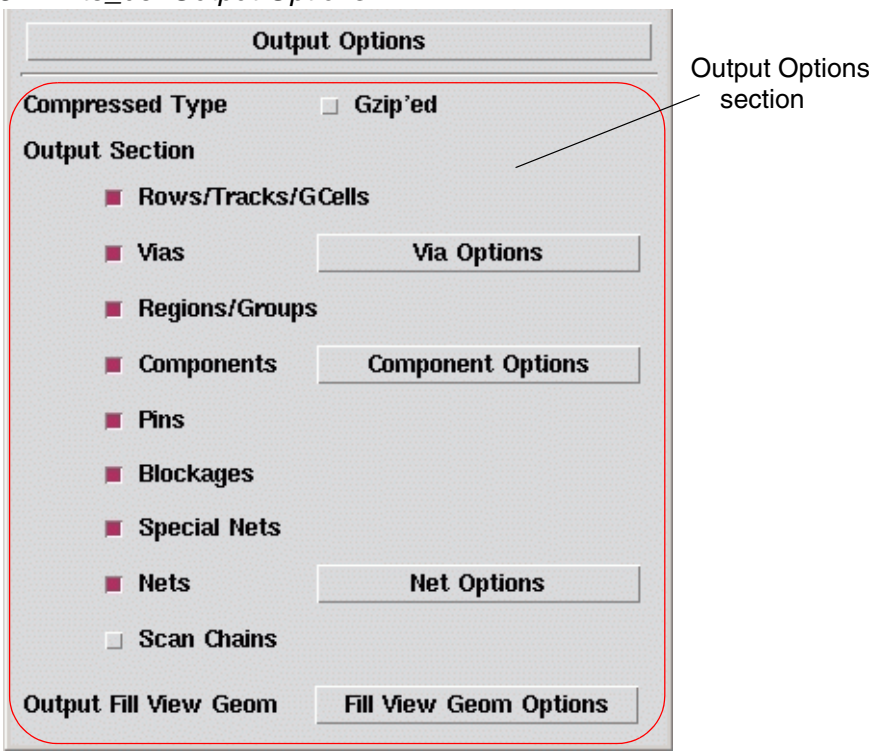
If you read a DEF file version 5.6 into your design and write out a DEF file with a previous version (5.3, 5.4, or 5.5),

- The output file might contain incomplete design data due to version incompatibilities.
- The generated vias change to fixed vias. When this happens, `read_def` issues a warning message.

Output Options

Clicking the Output Options button displays the Output Options section, as shown in Figure 4-23.

Figure 4-23 write_def Output Options



The Output Options section contains the following options:

Compressed Type

Default: not selected (compression off). If this option is selected, it specifies that the output format of the DEF is in gzip format.

If the output file name does not have a .gz extension, `write_def` appends .gz to the specified name. For example, if the Output File box is specified as mytest.def and the gzipped option is selected, `write_def` writes DEF in compressed format with the name mytest.def.gz.

Output Section

The following options are available:

- Rows/Tracks/GCells

  Default: selected. If this option is selected, the row, track, and global cell (GCell) grid sections are written. If it is not selected, the three sections are not written.

- Vias

  Default: selected. If this option is selected, the via section is written. If it is not selected, the via section is not written. The `write_def` command does not write all rotated vias and nondefault rule vias to the via section. They are optionally outputted to an incremental LEF file.

- Regions/Groups

  Default: selected. If this option is selected, the region and group sections are written. If it is not selected, the region and group sections are not written.

- Components

  Default: selected. If this option is selected, the component section is written. If it is not selected, the design components are not written; instead an empty component section with zero components is written. The component section is required in the DEF syntax.

- Pins

  Default: selected. If this option is selected, the pin section is written. If it is not selected, the pin section is not written.

- Blockages

  Default: selected. If this option is selected, the blockage section is written. If it is not selected, the blockage section is not written.

- Special Nets

  Default: selected. If this option is selected, the special net section is written. If it is not selected, the special net section is not written.

  The `write_def` command supports regular expressions * *pinName* for power and ground net connections in the Special Nets section of the DEF file. The corresponding net connections are omitted in the Nets section of the DEF file. This results in a compact DEF output file.

- Nets

Default: selected. If this option is selected, the regular net is written. If it is not selected, the design regular nets are not written; instead, an empty regular net section with zero nets is written. The net section is required in the DEF syntax.

- Scan Chains

  Default: not selected. If this option is selected, the scan chain section is written. If it is not selected, the scan chain section is not written.

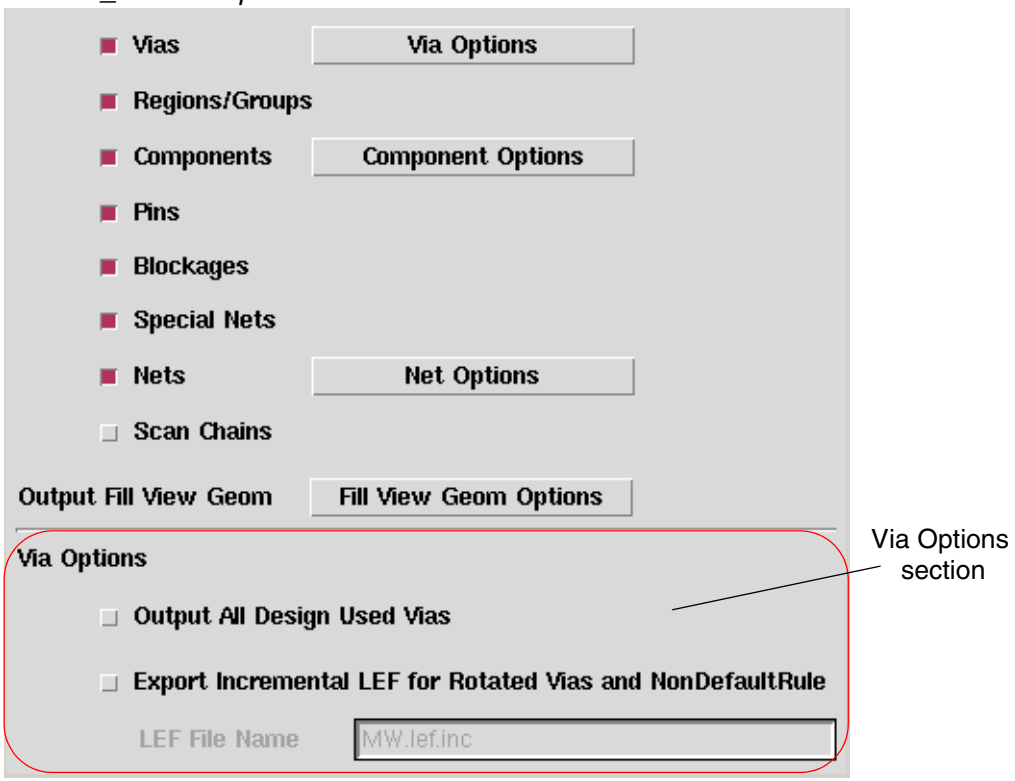
  Note:

  If you turn off all output options in the write def dialog box (Floorplan, Blockages, vias, Components, Pins, Nets, Special Nets, and Scan Chains), Milkyway outputs the constructs Header, Track, GCellGrid, FILL blockage, and Property Definition when you use DEF version 5.5.

Via Options

Clicking the Via Options button displays the Via Options section, as shown in Figure 4-24.

Figure 4-24 write_def Via Options



The Via Options section has the following options:

Output All Design Used Vias

Default: not selected. If this option is selected, Milkyway library contacts are written to the DEF via section. If the technology file and original DEF file contained different via definitions with the same name, the design uses the technology file definition and eliminates the duplicate output from the original DEF file definition. If it is not selected, Milkyway library contacts are not written to the DEF via section.

In general, library contacts should not be written to the DEF file, because DEF file should contain only design-specific information. Library contacts are not design-specified. They are defined in the library technology file and are available to any design in the library. Library contact information should be written only to the LEF file.

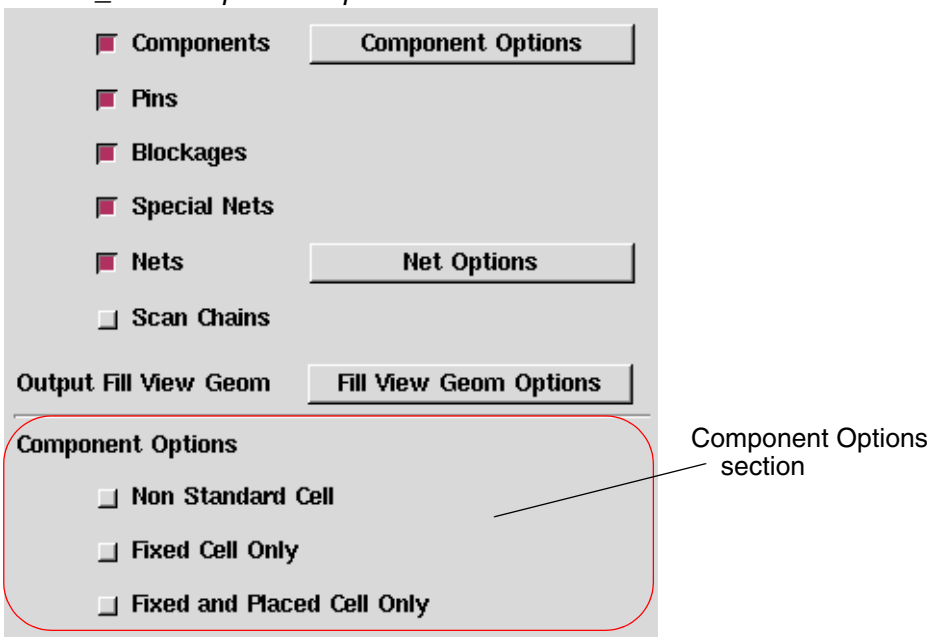
Export Incremental LEF for Rotated Vias and NonDefaultRule

Default: not selected. Default LEF file name: MW.lef.inc. If it is selected, write rotated vias and design-specified nondefault rule to a LEF file with the specified LEF file name.

Component Options

Clicking the Component Options button opens the Component Options section, as shown in Figure 4-25.

Figure 4-25 write_def Component Options



The Component Options section contains the following options:

Non Standard Cell

Default: not selected. If this option is selected, nonstandard cell information is written.

Fixed Cell Only

Default: not selected. If this option is selected, fixed cell information is written.

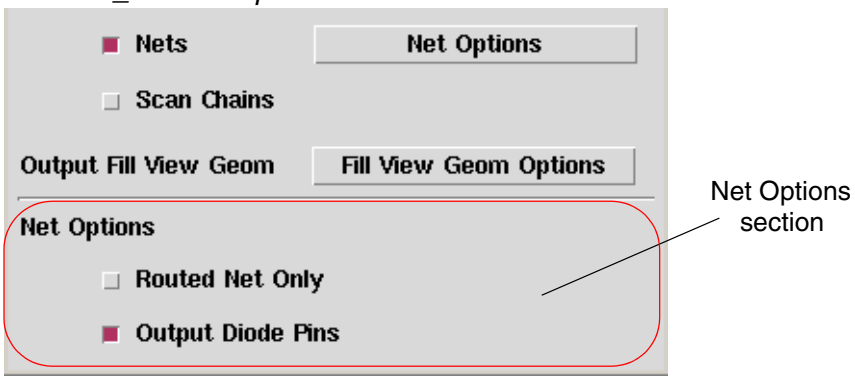
Fixed and Placed Cell Only

Default: not selected. If this option is selected, fixed and placed cell information is written.

Net Options

Clicking the Net Options button opens the Net Options section, as shown in Figure 4-26.

Figure 4-26 write_def Net Options



The Net Options section contains the following options:

Routed Net Only

Default: not selected. If this option is selected, only routed net wires are written to the regular net section. If it is not selected, all regular net wires are written to the regular net section.

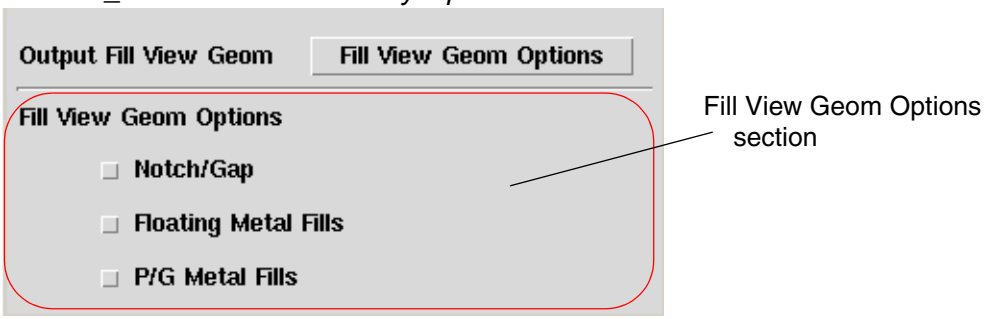
Output Diode Pins

Default: selected. If this option is selected, diode (extra) pins are written to the regular net section. If a special net has diode pins, its diode pins are written to the regular net section but its special pins and wiring remain in the special net section. If it is not selected, diode pins are not written.

Fill View Geom Options

Clicking the Fill View Geom Options button opens the Fill View Geom Options section, as shown in Figure 4-27.

Figure 4-27 write_def Fill View Geometry Options



The Fill View Geom Options section contains the following options:

Notch/Gap

Default: not selected. If this option is selected, notch and gap fills in the Fill View are written to the special net section with the shape `DRCFILL` (DEF 5.5) or shape `BLOCKAGEWIRE` (DEF 5.3 and DEF 5.4).

Floating Metal Fills

Default: not selected. If this option is selected, floating metal fills in Fill View are written to the Fills section.

P/G Metal Fills

Default: not selected. If this option is selected, power and ground metal fills in Fill View are written to the special net section with shape `FILLWIRE`.

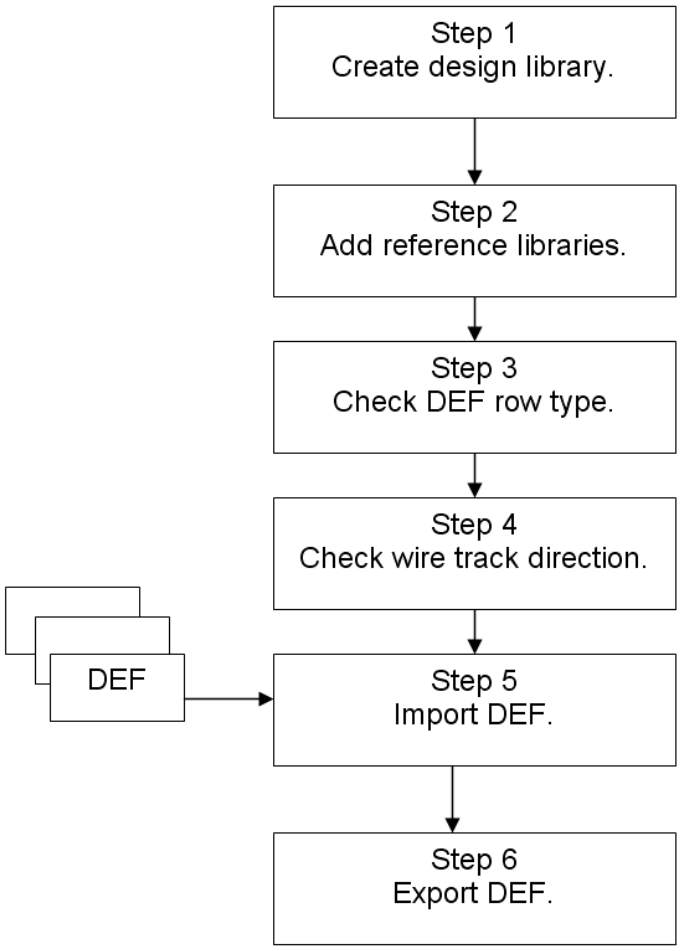
Recommended DEF Flows

The Milkyway DEF flow supports incremental DEF input, using Verilog. It supports Synopsys physical implementation tools as well as third-party place and route tools.

DEF Input and Output Flow

This section describes the flows used to successfully import a DEF file (or files) to a Milkyway design, using `read_def`, and to export the data to a DEF file, using `write_def`. Figure 4-28 is an overview of the input and output flow.

Figure 4-28 Flow for Importing and Exporting a DEF File



1. Create a design library from the technology file.
2. Add the required reference libraries.
3. Check the input DEF file for the row type.

   The name of the site in the design or reference library is “unit.” The row type should match this name. If the row type does not match the site name of the design or reference library (“unit”), Milkyway does not create rows.

4. Check for the wire track direction.

   By default, the wire track direction of all metal layers is horizontal. Therefore, you must tell the tool about the wire track direction of all the layers.

During DEF In, when the layer preferred wire track direction is not initialized, DEF reader attempts to set it according to the wire direction (WireDirection) object unitTile in the main and reference libraries. If it fails to set, DEF reader generates a warning message.

Next, select Set Layer Preferred Direction in the Track Options dialog box, to set the track direction for metal layers M1, M2, M3, and M4. All the metal layers higher than M4 are set to alternating directions with respect to M4.

5. Import the DEF file (or files) to the design library, using the `read_def` command.
6. Export the Design information to a DEF file, using the `write_def` command.

There are two types of vias in DEF. One type is a set of vias defined in the Milkyway technology file. The second type is a set of vias generated by the Astro tool, which are generated during design preparation. As such, they have no definition in the technology file.

For each via referenced in the net section wiring, if the match to any of the Milkyway Technology contact code is not found, the via is translated as a Milkyway via cell instance. The vias in the DEF vias section define all generated vias in the design. Therefore, a via cell is created to store all physical information before an effort to find the best match to any of the contact code.

DEF-Only Flow

In a DEF-only flow, both the netlist and physical information come from the DEF file. When you use a DEF-only flow, you must set the new `PCXGDEF` variable to 0 to update the netlist completely. To set the variable, enter the following on the Milkyway command line:

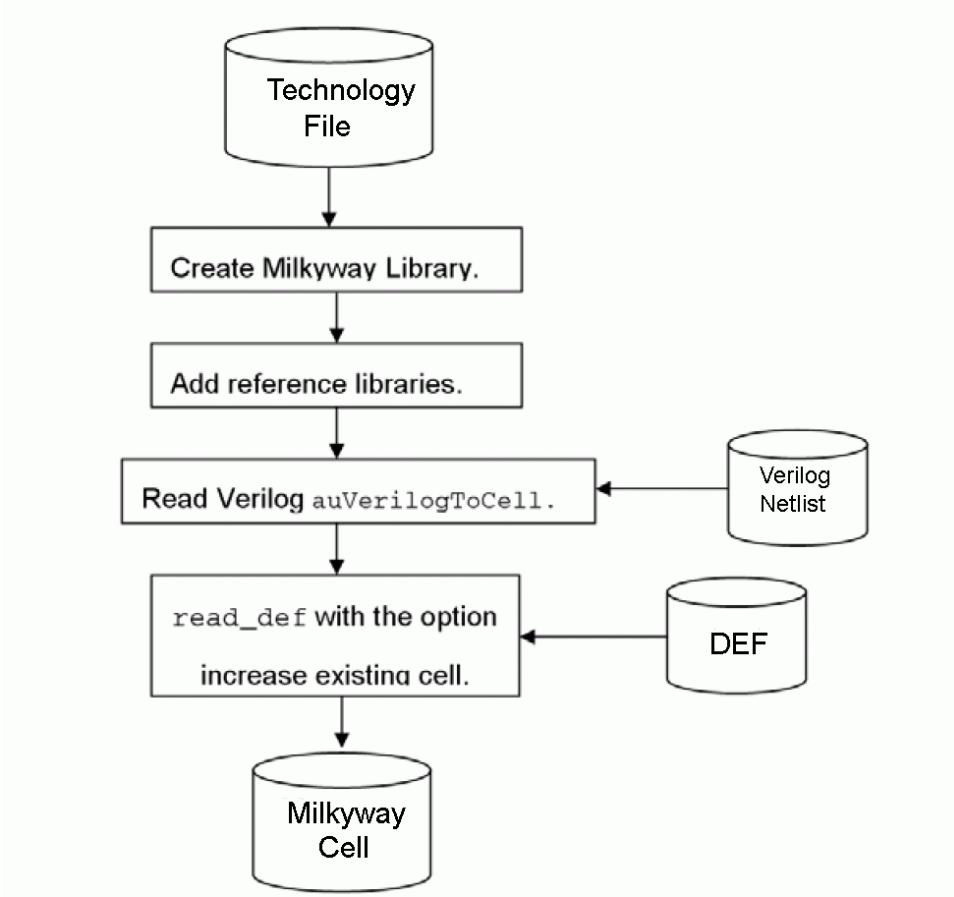
```
define PCXGDEF 0
```

Verilog Incremental DEF Flow

DEF can be read in incrementally, which means that `read_def` can add more information from the DEF file to the existing cell in the database. For example, if you have a placed cell and want to add clock routing on top of this cell, using a DEF file, you must select the Increase Existing Cell option in the `read_def` dialog box.

Figure 4-29 shows the incremental DEF flow of reading Verilog followed by DEF.

Figure 4-29 Verilog Incremental DEF Flow



1. Create a new Milkyway library, using appropriate technology files.
2. Add reference libraries to the design library.
3. Read the Verilog netlist and create a Milkyway design by using the `auVerilogToCell` command.
4. Read the DEF file (or files) in incremental mode.
   - Use the `read_def` command to read the DEF file.
   - The cell name should be the same as that created during Verilog In.
   - Select Increase Existing Cell in the “read def” dialog box.

Physical Compiler and Milkyway DEF Flow

This section describes how DEF design information can be transferred between Milkyway and the Physical Compiler tool.

The Milkyway DEF interface provides consistency with Physical Compiler. DEF data that is generated from Milkyway can be read without any errors into Physical Compiler. Similarly, DEF data from Physical Compiler is readable in Milkyway.

Importing DEF From Milkyway to Physical Compiler

Physical Compiler can read DEF and Verilog files that are written out by Milkyway. After you import DEF data into Physical Compiler, you can check for any errors in the design by using the `physopt -mpc -check_only` option in Physical Compiler.

To transfer DEF design data from Milkyway to Physical Compiler, do the following in Milkyway:

1. Use the `write_def` command to write out the DEF file from Milkyway.
2. Use the `auHierVerilogOut` command to write out the Verilog netlist for hierarchical designs, or use the `auVerilogOut` command to write out the Verilog netlist for flat designs.

For hierarchical designs, deselect the “Output Bus As Individual Bits” option in the `auHierVerilogOut` dialog box. When you do this, be sure the NETL view to be present in the Milkyway library for the design cell. Otherwise, when the Verilog is read in Physical Compiler, Physical Compiler gives LINK-1 errors for macro pins, as shown in the following error message:

```
Error: Can't find port 'A[0]' on reference to 'rf1sh416x8'
in 'merlot_core_after_dr'. (LINK-1)
If the hierarchy is preserved, dump the hierarchical Verilog
netlist using "auHierVerilogOut".
```

To read the DEF data in Physical Compiler, do the following in the psyn_shell interface:

1. Source the Physical Compiler setup file.
2. Read the Verilog netlist by using the `read_verilog` command.
3. Link the design by using the `link` and `link_physical` commands.
4. Read the DEF file by using the `read_def` command.
5. Run the `physopt -mpc -check_only` command to check for any errors in the design.
6. Run the `check_design` command to verify the design consistency.
7. Write out the DEF data by using the `write_def` command.

Importing DEF From Physical Compiler to Milkyway

To read a DEF file that was written out from Physical Compiler back into Milkyway,

1. Create a Milkyway library.
2. Add reference libraries.
3. Read the Verilog data that was previously written from Milkyway, and create a design cell by using `auVerilogToCell`.

   Note:

   You must specify the port and net connections in the Verilog In dialog box. Click the Global Net Options button in the `auVerilogToCell` dialog box to open the Verilog To Cell (Global Net Options) dialog box. For information about `auVerilogToCell`, See Physical Implementation Online Help.
4. Read the DEF data from Physical Compiler, by using `write_def` in incremental mode.

   Select Increase Existing Cell in the “read def” dialog box.

Note:

When DEF is being read from Physical Compiler to Milkyway, information messages will appear. You can safely ignore them. The following is a sample information message:

```
Port instance S in _RISC_CORE_I_PRGRM_CNT_TOP\/
I_PRGRM_CNT_Physical Compilerint_reg_0_ is connected
already.
```

Supported DEF 5.6 Syntax

Table 4-2 contains the DEF syntax (version 5.6) that Milkyway supports. The table headings have the following meanings:

- Parsed – Specifies that the syntax is recognized by Milkyway.
- Annotated – Specifies whether the constructs are stored in the Milkyway database.
- Written Out – Specifies whether a file is written out. The file could be written out in DEF format or as an attached file.

Note:

Due to version incompatibilities, you cannot read in a newer version of DEF and write out all corresponding information to an older version of DEF. This is not possible because the older version might be missing constructs that are supported in the newer version.

Table 4-2 DEF 5.6 Syntax Supported in Milkyway

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
1) VERSION `VERSION versionNumber ;`	Yes	No	Yes	The parser is a superset of 5.3, 5.4, 5.5, and 5.6.
2) DIVIDERCHAR `DIVIDERCHAR "character" ;`	Yes	Yes	Yes	This section is optional in DEF 5.6. Always assumed "/".
3) BUSBITCHARS `BUSBITCHARS "delimiterPair" ;`	Yes	No	Yes	This section is optional in DEF 5.6.
4) DESIGN `DESIGN designName ;`	Yes	No	Yes	The `read_def` command ignores the `designName`. The `designName` in `write_def` is the top module name.
5) TECHNOLOGY `TECHNOLOGY technologyName ;`	Yes	No	Yes	If the technology name is not `NULL`, it is set by `technologyName`; if it is `NULL`, you can ignore `technologyName`.

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
6) UNITS `UNITS DISTANCE MICRONS DEFconvertFactor ;`	Yes	No	Yes	Milkyway determines the design `UNITs` by the units in the technology file and not by DEF. DEF maps the `UNIT` value to the corresponding Milkyway design library. The values 10,000 and 20,000 are supported as legal DEF `UNITS` in DEF 5.6. The `read_def` command uses the units to determine the conversion scale factor, but it does not change the units in the technology file.
7) HISTORY `HISTORY anyText ;`	Yes	No	No	
8) PROPERTY DEFINITIONS `PROPERTYDEFINITIONS`	Yes	No	Yes	
`ObjectType propName propType [RANGE # #] [value | stringValue ; ?`	Yes	Yes	Yes	
`END PROPERTYDEFINITIONS`	Yes	No	Yes	
`ObjectType = { DESIGN | COMPONENT | NET | SPECIALNET | GROUP | ROW | COMPONENTPIN | REGION }`	Yes	Yes	Yes	
`PropType = { INTEGER | REAL | STRING }`	Yes	Yes	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
9) DIEAREA `DIEAREA pt pt [pt] ... ;`	Yes	Yes	Yes	Defines the boundary of the design. With DEF 5.6, geometric shapes (such as blockages, pins, and special net routing) can be outside the die area to allow proper modeling of pushed-down routing from top-level designs into sub blocks. However, routing tracks should still be inside the die area.
10) ROW				
`ROW`	Yes	Yes	Yes	
`RowName`	Yes	Yes	Yes	
`rowType`	Yes	Yes	Yes	
`origX origY`	Yes	Yes	Yes	
`orient`	Yes	Yes	Yes	
`[ DO numX BY 1`	Yes	Yes	Yes	The `DO` syntax is optional in DEF version 5.6, in which case a single site is created.
`STEP spaceX 0`	Yes	Yes	Yes	This statement is optional in DEF 5.6. If not specified, the value is determined by the size of the `SITE` in LEF.
`| DO 1 BY numY`	Yes	Yes	Yes	The `DO` syntax is optional in DEF 5.6, in which case a single site is created.
`STEP 0 spaceY ]`	Yes	Yes	Yes	This statement is optional in DEF 5.6. If not specified, the value is determined by the size of the `SITE` in LEF.
`[ + PROPERTY { propName propVal }?]...;`	Yes	No	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
11) TRACKS				
`TRACKS`	Yes	Yes	Yes	
`{ X | Y }`	Yes	Yes	Yes	
`start`	Yes	Yes	Yes	
`DO numTracks`	Yes	Yes	Yes	
`STEP space`	Yes	Yes	Yes	
`LAYER layerName ;`	Yes	Yes	Yes	
12) GCELLGRID `GCELLGRID`	Yes	Yes	Yes	When the input DEF file includes `GCELLGRID` information, `write_def` writes it out from the input DEF file. If the information is not included in the input DEF file, `write_def` uses the information from the global route cell.
`X start DO numColumns+1 STEP space`	Yes	Yes	Yes	
`Y start DO numRows+1 STEP space ;`	Yes	Yes	Yes	
13) VIAS				
`VIAS numVias`	Yes	No	Yes	
`[- viaName`	Yes	Yes	Yes	`ViaNameMapFile` is used to record how the via name is mapped to the Milkyway database. In general, keep the contact name unchanged unless the special character needs to be changed to '_'.

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
				As a rule, full contact array names must be changed as follows: If the contact array is central symmetrical, the name should be *viaName-XSize-YSize-ALL-xTimes-yTimes*; otherwise, the name should be *viaName|left|bottom|right|top|ALL|xTimes|yTimes*.
`[ + VIARULE viaRuleName`	Yes	Yes	Yes	When you specify `DEFAULT` as the reserved via rule name, the via uses the previously defined `VIARULE GENERATE` rule with the `DEFAULT` keyword that exists for this routing-cutrouting-routing layer combination. Currently, the technology file cannot differentiate between a via rule generate contact code and a default via contact code. Thus, router- generated vias are written out as fixed vias. The `read_def` command checks the following information. It quits with an error if the following occurs: • the `viaRuleName` contact code is not found in the design library. • the contact code in the reference library comes from a `VIARULE GENERATE` statement. • the cut size or layers are not equal to the definition in the contact code or the cut spacing or enclosure value is smaller than the definition in contact code. If this occurs, no via instance object will be created.
`+ CUTSIZE xSize ySize`	Yes	Yes	Yes	If the cut size is not equal to the definition in the contact code, `read_def` quits with an error.

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`+ LAYERS botmetalLayer cutLayer topMetalLayer`	Yes	Yes	Yes	If the layers are not equal to the definition in the contact code, `read_def` quits with an error.
`+ CUTSPACING xCutSpacing yCutSpacing`	Yes	Yes	Yes	If the cut spacing is smaller than the definition in the contact code, `read_def` quits with an error.
`+ ENCLOSURE xBotEnc yBotEnc xTopEnc yTopEnc`	Yes	Yes	Yes	If the enclosure value is smaller than the definition in the contact code, `read_def` quits with an error.
`[+ ROWCOL numCutRows NumCutCols]`	Yes	Yes	Yes	
`[+ ORIGIN xOffset yOffset]`	Yes	Yes	Yes	
`[+ OFFSET xBotOffset yBotOffset xTopOffset yTopOffset]`	Yes	Yes	Yes	
`[+ PATTERN cutPattern] ]`	Yes	Yes	Yes	
`| [ + RECT layerName pt pt`	Yes	Yes	Yes	
`| + POLYGON layerName pt pt pt] ...];`	Yes	No	No	
`END VIAS`	Yes	No	Yes	
14) Styles				This section is currently not supported.
`[Styles`	Yes	No	No	
`numStyles ;`	Yes	No	No	
`{- STYLE styleNum pt pt pt ... ;} ...`	Yes	No	No	
`END STYLES]`	Yes	No	No	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
15) NONDEFAULTRULES				This section is currently not supported.
`NONDEFAULTRULES numRules ;`	Yes	No	No	
`{- ruleName`	Yes	No	No	
`[+ HARDSPACING]`	Yes	No	No	
`{+ LAYER layerName`	Yes	No	No	
`WIDTH minWidth`	Yes	No	No	
`[DIAGWIDTH diagWidth]`	Yes	No	No	
`[SPACING minSpacing]`	Yes	No	No	
`[WIREEXT wireExt]} ...`	Yes	No	No	
`[+ VIA viaName] ...`	Yes	No	No	
`[+ VIARULE viaRuleName] ...`	Yes	No	No	
`[+ MINCUTS cutLayerName numCuts] ...`	Yes	No	No	
`[+ PROPERTY {propName propVal} ...] ...`	Yes	No	No	
`END NONDEFAULTRULES`	Yes	No	No	
16) REGIONS				
`REGIONS numRegions ;`	Yes	No	Yes	
`[ - regionName pt pt`	Yes	Yes	Yes	
`[ pt pt ]?`	Yes	Yes	Yes	The `read_def` and `write_def` commands support multiple rectangles.

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`+ TYPE { FENCE | GUIDE }`	Yes	Yes	Yes	`FENCE`: exclusive and rigidity = 10. `GUIDE`: nonexclusive and rigidity = 5. By default, `region` is nonexclusive and rigidity = 10.
`[ + PROPERTY { propName propVal }... ]... ; ]?`	Yes	Yes	Yes	
`END REGIONS`	Yes	No	Yes	
17) COMPONENTS				
`COMPONENTS numComps ;`	Yes	No	Yes	
`[ - compName modelName`	Yes	Yes	Yes	
`[ + EEQMASTER macroname ]`	Yes	Yes	Yes	
`[ + SOURCE NETLIST | DIST | USER | TIMING ] ]`	Yes	Yes	Yes	
`[+ {FIXED pt orient | COVER pt orient | PLACED pt orient | UNPLACED} ]`	Yes	Yes	Yes	
`[+ HALO left bottom right top]`	Yes	No	No	
`[ + WEIGHT weight ]`	Yes	Yes	Yes	
`[ + REGION regionName ]`	Yes	Yes	Yes	
`[ + PROPERTY { propName propVal }... ]... ; ]?`	Yes	Yes	Yes	
`END COMPONENTS`	Yes	No	Yes	
18) PINS				
`PINS numPins ;`	Yes	Yes	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`[ [ - pinName + NET netName ]`	Yes	Yes	Yes	
`[ + SPECIAL ]`	Yes	Yes	Yes	
`[ + DIRECTION { INPUT | OUTPUT | INOUT | FEEDTHRU } ]`	Yes	No	Yes	Milkyway does not have a `FEEDTHRU` type. Therefore, `FEEDTHRU` is translated into `INOUT`.
`[+ NETEXPR "netExprPropName defaultNetName"]` `[+ SUPPLYSENSITIVITY powerPinName]` `[+ GROUNDSENSITIVITY groundPinName]`	Yes	No	No	
`[ + USE [SIGNAL | POWER | GROUND | CLOCK | TIEOFF | ANALOG | SCAN | RESET ] ]`	Yes	Yes	Yes	Milkyway does not support `TIEOFF, ANALOG, SCAN, RESET` and regards them as unknown.
`<antenna rules>`	Yes	Yes	Yes	
`[ + LAYER layerName pt pt | + POLYGON layerName pt pt pt ...] ...` `[ + { FIXED | PLACED | COVER } pt orient ]`	Yes	Yes	Yes	`pinName` is used to set `nameId`. Polygon-shaped pins are supported in DEF 5.6.
`[SPACING minSpacing | DESIGNRULEWIDTH effectiveWidth`	Yes	No	No	
`END PINS`	Yes	No	Yes	
19) PINPROPERTIES				
`PINPROPERTIES num ;`	Yes	No	Yes	
`[- {compName pinName | PIN pinName}`	Yes	Yes	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`[ + PROPERTY { propName propVal }... ]... ; ]…`	Yes	Yes	Yes	
`END PINPROPERTIES`	Yes	No	Yes	
20) BLOCKAGES				
`Blockage numblockages ;`	Yes	No	Yes	
`- LAYER LayerName`	Yes	Yes	Yes	
`+ COMPONENT CompName | + SLOT | + FILLS`	Yes	No	No	
`+ PUSHDOWN`	Yes	Yes	No	
`[+ SPACING minSpacing | + DESIGNRULEWIDTH effectiveWidth]`	Yes	No	No	
`{ Rect pt pt`	Yes	Yes	Yes	
`| POLYGON pt pt pt ...} ...`	Yes	No	No	
`- PLACEMENT`	Yes	Yes	Yes	
`+ COMPONENT CompName |`	Yes	No	No	
+ PUSHDOWN	Yes	Yes	No	Currently, `write_def` handles `PUSHDOWN` placement blockage the same as other placement blockages, and the `PUSHDOWN` keyword cannot be written out.
`Rect pt pt`	Yes	Yes	Yes	
`END BLOCKAGES`	Yes	No	Yes	
21) SLOTS				This section is currently not supported.
`SLOTS NumSlots ;`	Yes	No	No	
`- LAYER layer name`	Yes	No	No	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`{ Rect pt pt`	Yes	No	No	
`| POLYGON pt pt pt ... }` `...`	Yes	No	No	
`END SLOTS ;`	Yes	No	No	
22) FILLS				Floating metal fills are stored in the FILL view.
`FILLS Num Fills;`	Yes	No	Yes	
`- LAYER layer name`	Yes	Yes	Yes	
`{ Rect pt pt`	Yes	Yes	Yes	
`| POLYGON pt pt pt ...}` `...`	Yes	No	No	
`END FILLS`	Yes	No	Yes	
23) SPECIALNETS				Connectivity is read in only for power nets.
`SPECIALNETS numNets ;`	Yes	No	Yes	
`-netName`	Yes	Yes	Yes	The name string is referenced by `nameId`.
`( compNameRegExpr pinName )`	Yes	Yes	Yes	
`+ SYNTHESIZED`	Yes	No	No	Connectivity that is read in for `SYNTHESIZED` is ignored.
`+ VOLTAGE volts`	Yes	Yes	Yes	
`+ SOURCE { NETLIST | DIST | USER | TIMING }`	Yes	Yes	Yes	
`+ FIXEDBUMP`	Yes	Yes	Yes	
`+ ORIGINAL netName`	Yes	Yes	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`+ USE {SIGNAL | POWER | GROUND | CLOCK | TIEOFF | ANALOG | SCAN | RESET}`	Yes	Yes	Yes	
`+ PATTERN { STEINER | BALANCED | WIREDLOGIC | TRUNK }`	Yes	Yes	Yes	
`+ ESTCAP wireCapacitance`	Yes	Yes	Yes	
`+ WEIGHT weight`	Yes	Yes	Yes	
`+ PROPERTY: { propName propVal }... ]` `SPECIALWIRING`	Yes	Yes	Yes	
`+ POLYGON layerName pt pt pt ...`	Yes	No	No	Milkyway does not support `POLYGON`. In DEF 5.6, 45-degree routing is described using `routingPoints`. Note: DEF 5.6 allows you to import and export 45-degree preroutes and polygon-shaped pins. However, DEF supports only nonextension and half-width extension 45-degree routing.
`+ RECT layerName pt pt`	Yes	No	No	
`{+ COVER | + FIXED | +ROUTED}`	Yes	Yes	Yes	
`+ SHIELD shieldNetName`	Yes	Yes	Yes	
`{ layerName routhWidth | NEW layerName routeWidth }`	Yes	Yes	Yes	
`routingPoints x y [extValue]`	Yes	Yes	Yes	DEF 5.6 supports 45-degree routing in point to point style.

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`viaName [ DO numX BY numY STEP stepX stepY ]`	Yes	Yes	Yes	
`+ SHAPE { RING | PADRING | BLOCKRING | STRIPE | FOLLOWPIN | IOWIRE | COREWIRE | BLOCKWIRE | BLOCKAGEWIRE | FILLWIRE | DRCFILL }`	Yes	Yes	Yes	`routeType` is determined by `SHAPE` and `USE`. `DRCFILL` maps to notch/gap in the FILL view.
`+ STYLE styleNum`	Yes	No	No	
24) NETS				
`NET numNets`	Yes	No	Yes	
`- netName`	Yes	Yes	Yes	The `Name` string is referenced by `nameId`.
`{ compName pinName | PIN pinName [ + SYNTHESIZED ] }`	Yes	Yes	Yes	
`MUSTJOIN ( compName pinName )`	Yes	Yes	Yes	
`+ SHIELDNET shieldNetName`	Yes	Yes	Yes	
`+ VPIN vpinName [ LAYER layerName ] pt pt [ PLACED pt orient | FIXED pt orient | COVER pt orient ]`	Yes	No	No	
`+ SUBNET subnetName [ ( { compName pinName | PIN pinName | VPIN vpinName } ) ] [NONDEFAULTRULE ruleName ] [ regularWiring ] …]`	Yes	No	No	
`+ XTALK class`	Yes	Yes	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`+ NONDEFAULTRULE ruleName`	Yes	Yes	Yes	
`+ SOURCE { DIST | NETLIST | TEST | TIMING | USER }`	Yes	Yes	Yes	
`+ FIXEDBUMP`	Yes	Yes	Yes	
`+ FREQUENCY frequency`	Yes	Yes	Yes	
`+ ORIGINAL netName`	Yes	Yes	Yes	
`+ USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL | TIEOFF }`	Yes	Yes	Yes	
`+ PATTERN {BALANCE | STEINER | TRUNK | WIREDLOGIC }`	Yes	Yes	Yes	
`+ ESTCAP wireCapacitance`	Yes	Yes	Yes	
`+ WEIGHT weight`	Yes	Yes	Yes	
`[ + PROPERTY: { propName propVal }... ]... ; ]?`	Yes	Yes	Yes	
`REGULARWIRING`				
`{+ COVER | + FIXED | + ROUTED | + NOSHIELD}`	Yes	Yes	Yes	The `FIXED`/`COVER` wiring route type is user-defined.
`layerName | NEW layerName`	Yes	Yes	Yes	
`[ TAPER`	Yes	Yes	Yes	`TAPER` sets the width to the minimum width of the layer.
`| TAPERRULE ruleName ]`	Yes	Yes	No	`read_def` uses the rules specified in `taperrule`.
`+ STYLE styleNum`	Yes	No	No	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`( x y [extValue] )`	Yes	Yes	Yes	Milkyway supports 45-degree routing in DEF 5.6.
`viaName [ orient ]`	Yes	Yes	Yes	
25) SCANCHAINS				
`SCANCHAINS numScanChains ;`	Yes	No	Yes	
`[ - chainName`	Yes	Yes	Yes	
`+ PARTITION <name> [MAXBITS mbits]`	Yes	Yes	Yes	
`[ + COMMONSCANPINS [ IN pin ] [ OUT pin ] ]`	Yes	Yes	No	The scan in and out port information is written out to the `FLOATING/ORDERED` statement.
`[ + START { fixedInConp | PIN } [ outPin ] ]`	Yes	Yes	Yes	
`[ + FLOATING {floatingComp [ ( IN pin ) ] [ ( OUT pin ) ]`	Yes	Yes	Yes	
`[ ( BITS numBits ) ]} ...]`	Yes	Yes	Yes	
`[+ ORDERED {fixedComp [ ( IN pin ) ] [ ( OUT pin ) ]`	Yes	Yes	Yes	
`[ ( BITS numBits ) ]} ...]`	Yes	Yes	Yes	
`[ + STOP { fixedOutComp | PIN } [ inPin ] ] ; ]…`	Yes	Yes	Yes	
`END SCANCHAINS`	Yes	No	Yes	
26) GROUPS				
`GROUPS NumOfGroups ;`	Yes	No	Yes	

Table 4-2 DEF 5.6 Syntax Supported in Milkyway (Continued)

DEF 5.6 Syntax	Parsed	Annotated	Written out	Comments
`[- groupName compNameRegExpr ...`	Yes	Yes	Yes	
`[+ REGION regionName ]`	Yes	Yes	Yes	
`[ + PROPERTY { propName propVal }... ]... ; ]?`	Yes	Yes	Yes	
`END GROUPS ;`	Yes	No	Yes	
27) END DESIGN				
`END DESIGN`	Yes	No	Yes	

5

Importing Physical Library Data

To prepare your data, you must first import the physical cells into one or more libraries. The steps for importing the physical cells are covered in this chapter, which contains the following sections:

- Library Preparation Using PLIB and PDB Data
- Creating a Milkyway Library by Using read_plib
- Creating a Milkyway Library by Using read_lib

Library Preparation Using PLIB and PDB Data

PLIB (the text format for specifying physical information) defines the elements of an IC process technology and associated library of cell models and contains library information for a class of designs. This library data includes layer, via, placement site type, and macro cell definitions.

A physical library specifies the information required for floor planning, RC estimation and extraction, placement, and routing. PDB is the compiled binary format of PLIB. Milkyway calls the Library Compiler tool to compile the PLIB data and convert it to FRAM and CEL. If you start with a PDB file instead of a PLIB file, the compilation step is already done and the PDB file is converted to FRAM and CEL.

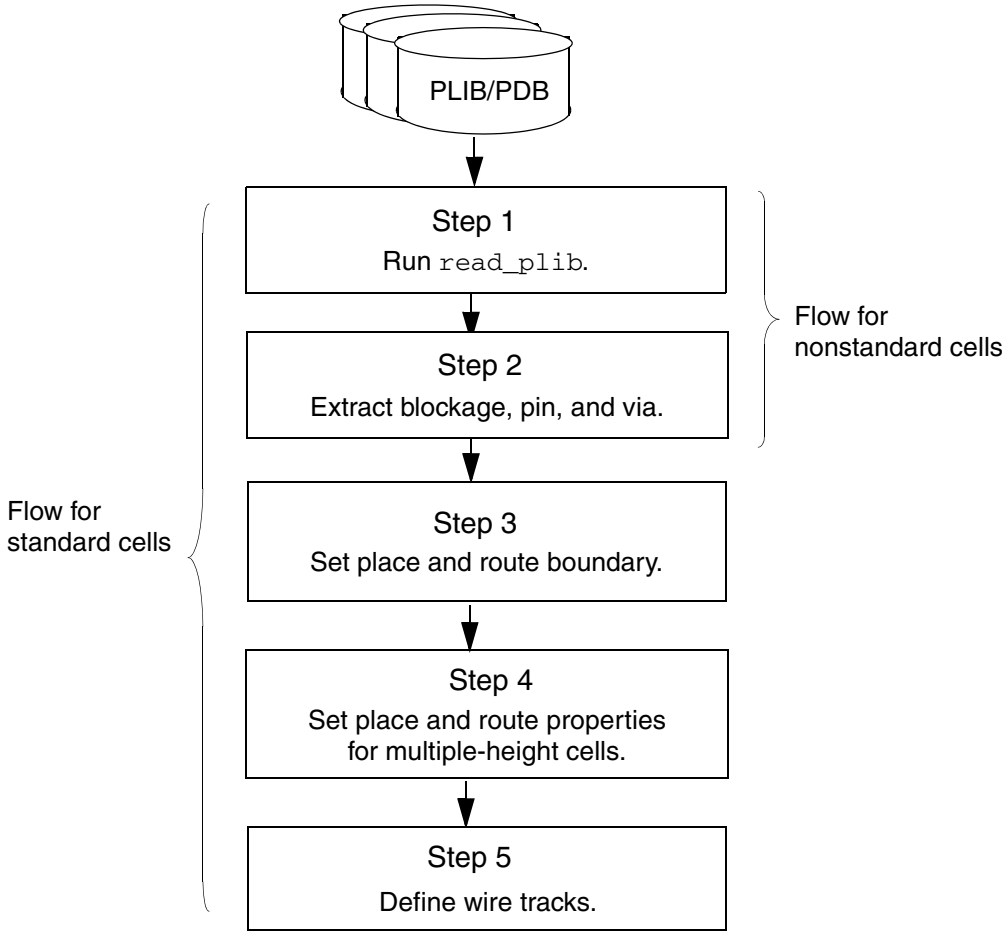
The commands for importing physical library data are `read_plib` (automated flow) and `read_lib` (advanced flow) for standard and nonstandard cells. Both commands provide complete translation for PLIB and PDB syntax.

The process of importing PLIB/PDB data to create a Milkyway reference library is described in the following sections:

- Creating a Milkyway Library by Using read_plib
- Creating a Milkyway Library by Using read_lib

The five-step flow for creating a Milkyway library for standard cells is shown in Figure 5-1.

Figure 5-1 Flow for Creating a Milkyway Library From PLIB/PDB Data



Overview of read_plib

PLIB is used as the ASCII format for representing physical information in the Synopsys flow. There are also translators available to generate PLIB data from Milkyway or LEF. Complete translation is available for all PLIB syntax, using the `read_plib` command.

The `read_plib` command automates PLIB importing for standard cells in a five-step flow and nonstandard cells in a two-step flow. (See Figure 5-1 on page 5-3.) Both flows create a place-and-route-ready Milkyway reference library annotated with physical library data.

Note:

All the options selected during PLIB/PDB library preparation are saved when you write out a PLIB file. You can make additional changes to your design by reloading the PLIB file; you do not need to go through the steps in the GUI again.

For information about PLIB syntax, see the *Library Compiler Reference Manual: Physical Libraries*.

Figure 5-2 shows the automated `read_plib` flow for creating a Milkyway library for standard cells. Note that the `read_lib` advanced flow follows the same five steps; however, you must manually perform each step.

Figure 5-2 read_plib Flow for Standard Cells

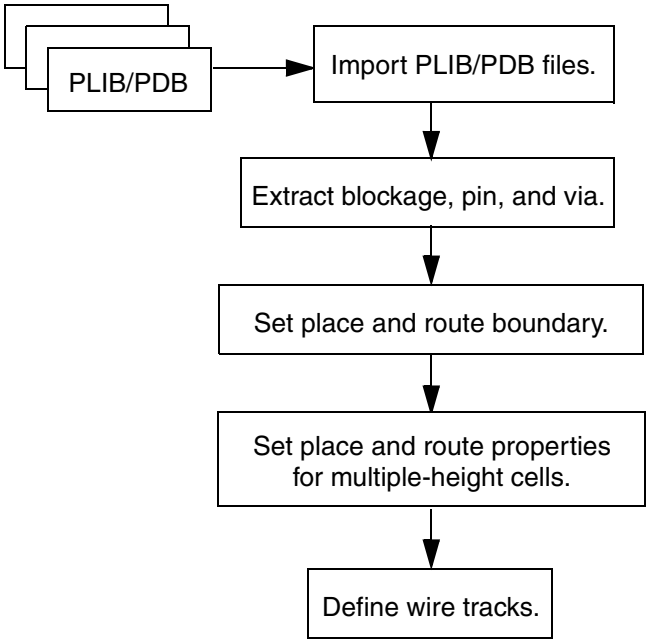
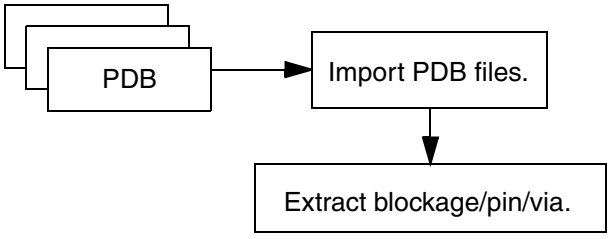


Figure 5-3 shows the `read_plib` flow for nonstandard cells, using a two-step automated process. This `read_plib` flow runs through all the steps shown in Figure 5-2 for the standard cell flow. During the automated `read_plib` flow for nonstandard cells, you can ignore any error messages that might be reported in the last three steps.

The `read_lib` advanced flow uses the same two steps for nonstandard cells as shown in Figure 5-3. However, you must manually perform the steps.

Figure 5-3 read_plib Flow for Nonstandard Cells



Overview of read_lib

The `read_plib` command automates the five-step flow required to create a place-and-route-ready Milkyway library. However, in situations where the automated flow does not work for your library, you can manually run the five-step flow by using the `read_lib` command. Use the `read_lib` manual flow if you want to change the default settings used in `read_plib`.

You use the Read Library dialog box to perform the manual flow. To open the dialog box, enter `read_lib` on the Milkyway command line or choose Cell Library > Library Preparation in the menu. The options in the Read Library dialog box guide you through the steps required to create a place-and-route-ready Milkyway library. For more information about the `read_lib` flow, see “Creating a Milkyway Library by Using read_lib” on page 5-11.

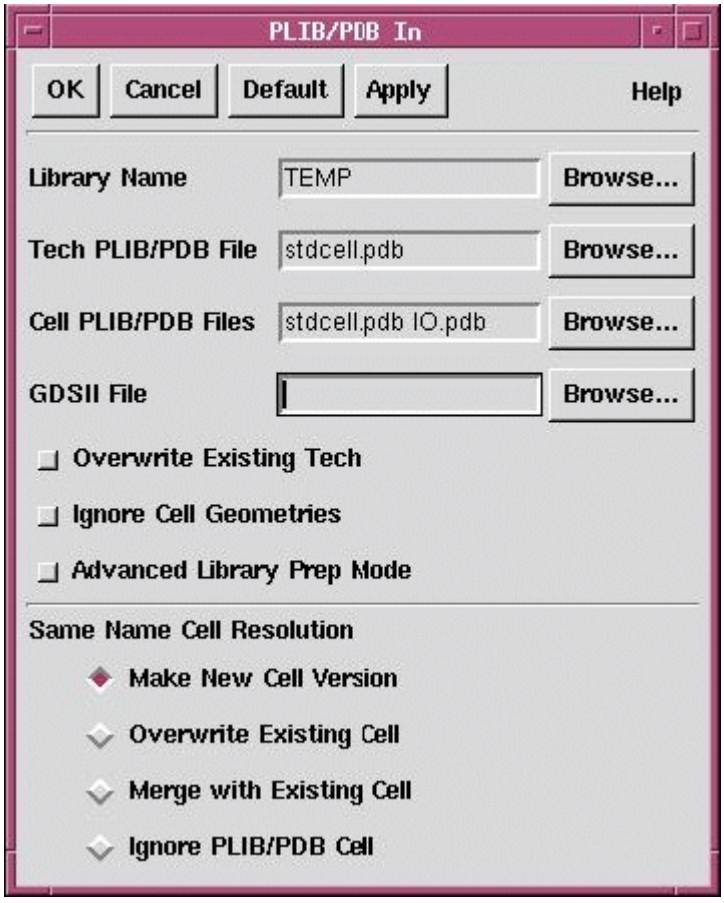
Creating a Milkyway Library by Using read_plib

The `read_plib` command automates the Milkyway library preparation flow and creates a place-and-route-ready Milkyway library by automatically running the steps shown in Figure 5-2 on page 5-4. You must perform step 1. After you complete step 1, the tool automatically runs step 2 through step 5.

Step 1: Importing PLIB and PDB Files, Using read_plib

The steps for importing PLIB and PDB files, using the `read_plib` command, are as follows:

1. Run the `read_plib` command (or choose Cell Library > Import PLIB in the menu) to open the PLIB/PDB In dialog box.
2. Specify the library name. The following figure shows the PLIB/PDB In dialog box with a library named TEMP.



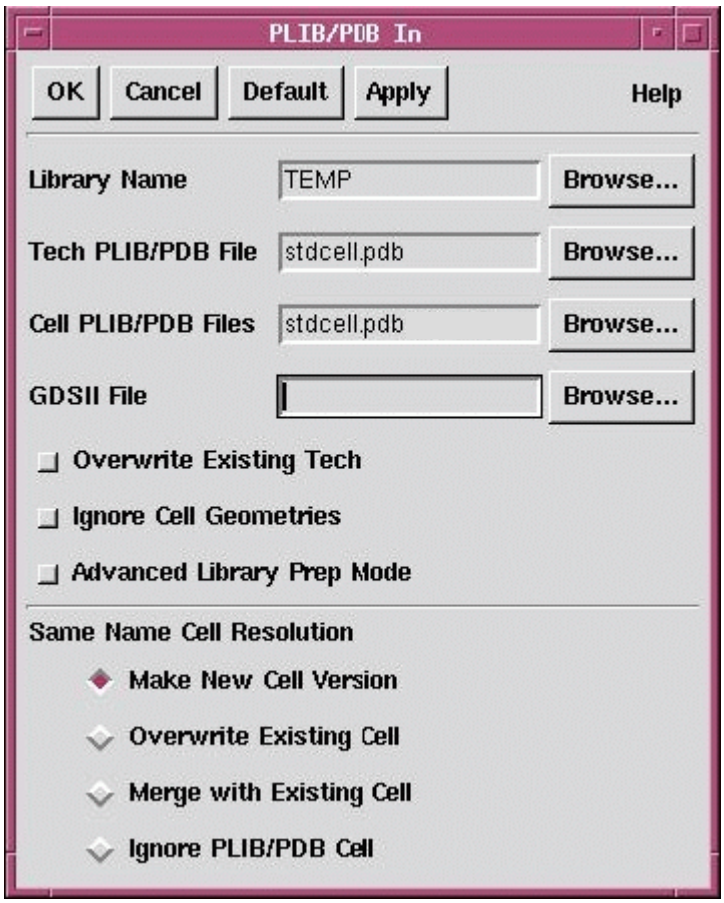
You do not need to have an existing library to run `read_plib`. If you do not have an existing library, the `read_plib` command creates a new reference library based on the PLIB/PDB information. If the Milkyway library already exists, you can incrementally import the PDB file.

3. (Optional) Specify the PLIB/PDB file in the Tech PLIB/PDB File and Cell PLIB/PDB Files boxes. There can be multiple methods for reading in PLIB and PDB files. For example, you can

   - Import a single PLIB or PDB file with the technology and cell information
   - Import multiple PLIB and PDB files for technology and cell information

   If you import a single PDB file with all the technology and cell information, you should specify the same PDB file in the Tech PLIB/PDB File and Cell PLIB/PDB Files boxes in the PLIB/PDB In dialog box. For example, if the PDB file is called stdcell.pdb in the Tech PLIB/PDB File box, specify stdcell.pdb in the Cell PLIB/PDB Files box as well, as shown in Figure 5-4.

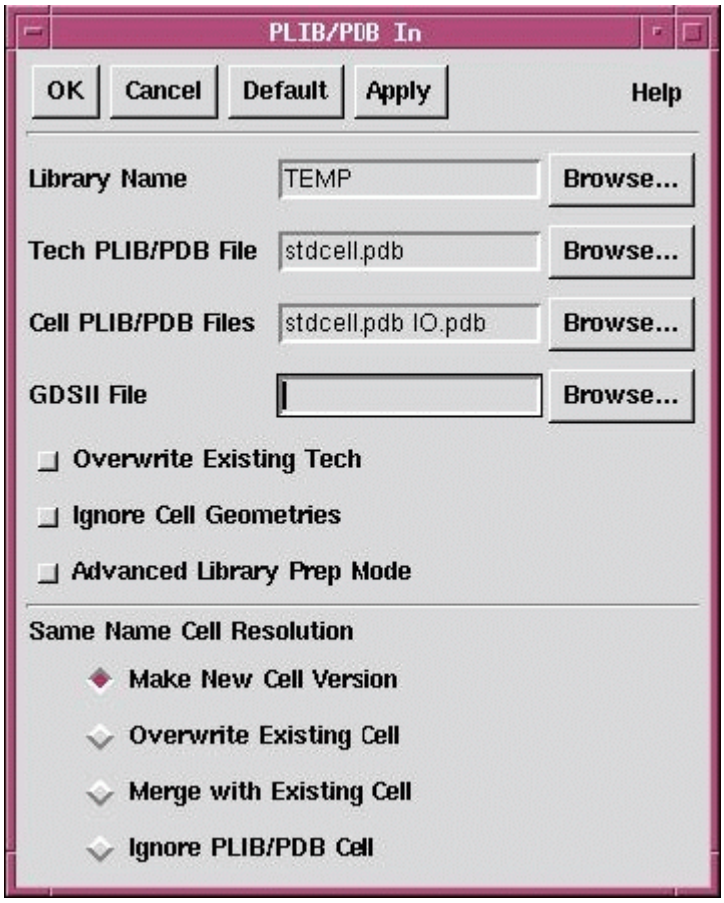
Figure 5-4 Importing a Single PDB File



If you have multiple PDB files for technology information and multiple PDB files for cell information, you must specify one file in the Tech PLIB/PDB File section (choose the most complete file) and multiple files in the Cell PLIB/PDB Files section.

For example, in the Tech PLIB/PDB File box, you can specify stdcell.pdb, which has all the technology information, and in the Cell PLIB/PDB Files box, specify stdcell.pdb and IO.pdb as shown in Figure 5-5. Use a blank space or comma as a separator when you specify multiple files.

Figure 5-5 Importing Multiple PDB Files



Note that only the technology section from the files specified in the Tech PLIB/PDB File box is used. The cell-level information is not used. Similarly, only the cell-level information from the files specified in the Cell PLIB/PDB Files box is used. The technology information from the files specified in the Cell PLIB/PDB Files box is not used.

4. (Optional) Specify the GDSII input file. If you have a GDSII file and want to create a new Milkyway library by using geometry data from the GDSII file instead of from the PLIB file, you must perform the following steps:

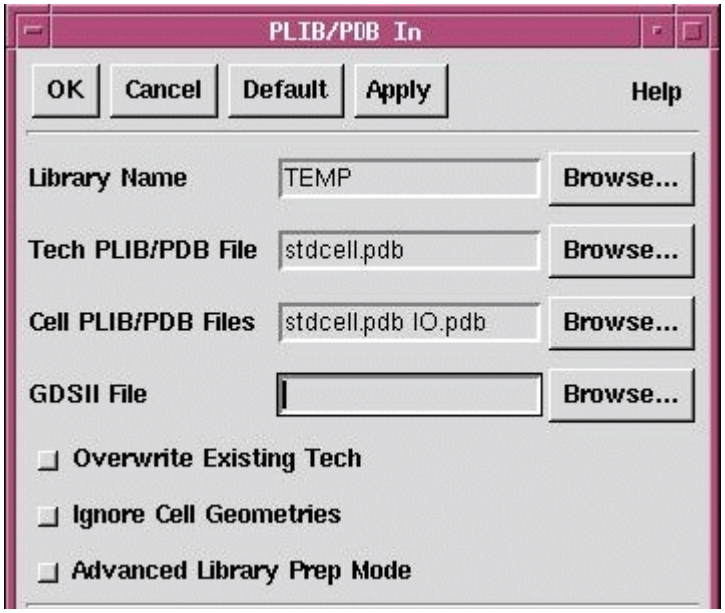
   a. Specify the GDSII file in the GDSII File box.
   b. Specify a technology PDB input file (to determine the technology information) in the Tech PLIB/PDB File box.
   c. Specify a cell PLIB/PDB file (to determine the cell-related information, such as pin direction and antenna property) in the Cell PLIB/PDB Files box.
   d. Click the Ignore Cell Geometries option so that Milkyway uses the geometries from the GDSII file.

If you are using the incremental flow, where you have an existing Milkyway library and want to add cells, you do not need to specify a technology PDB input file. However, you must specify a cell PLIB/PDB file to determine the cell-related information, such as pin direction and antenna property, and you must specify the GDSII file for the cells that need to be added.

5. Click OK.

read_plib Options. The PLIB/PDB In dialog box is divided into two sections: the main options and the Same Name Cell Resolution options. Figure 5-6 shows the main options section, followed by a description of each option.

Figure 5-6 PLIB/PDB In Dialog Box: Main Options



Library Name (Required)

The Milkyway library in which the library information and reference cells will be created. The string can include the path to the library.

If no path is specified, the library is resolved from the current working directory. If you do not have an existing library, the `read_plib` command creates a new reference library based on the PLIB/PDB information.

Note:

The `read_plib` command can accept library names with up to 1,024 characters.

Tech PLIB/PDB File (Optional)

The technology PDB input file. Specify the PLIB/PDB file that has technology information in this box. If multiple PLIB/PDB files have technology information, specify the file that is most complete. Even if you specify a complete standard cell library with technology information, only the technology section from the file specified in this option will be used. The cell-level data will be ignored.

Cell PLIB/PDB Files (Optional)

The cell PLIB/PDB input files. You can specify multiple PLIB/PDB files if, for example, you have separate PLIB/PDB files for standard cells, macros, and I/Os. Use a blank space or comma as a separator when you specify multiple files.

GDSII File (Optional)

The GDSII input file. If you have a GDSII file and want to create a new Milkyway library by using geometry data from the GDSII file instead of from the PLIB file, specify the GDSII file in this box. You must also specify a technology PDB input file to determine the cell-related information (such as pin direction and antenna property) and specify a cell PLIB/PDB file, and you must click the Ignore Cell Geometries option so that Milkyway uses the geometries from the GDSII file.

If you are using the incremental flow, you do not need to specify a technology PDB input file. However, you must specify a cell PLIB/PDB file and the GDSII file for the cells that need to be added.

Overwrite Existing Tech (Optional)

This option is not selected by default. If you select this option, all the existing technology information from the library is removed, and the new technology information from the Tech PLIB/PDB files is used instead.

Ignore Cell Geometries (Optional)

This option is not selected by default. If you select this option, the pin, rectangle, polygon, and via objects are not created. After PLIB/PDB input, you must input the GDSII data to create the physical information. This option is typically used for the Tech PLIB/PDB and physical cell GDS flow.

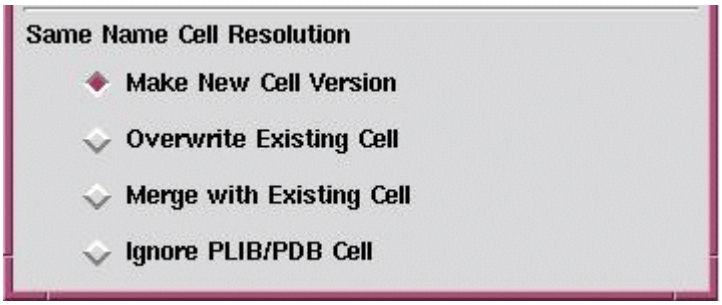
Advanced Library Prep Mode (Optional)

This option is not selected by default. In the default mode, `read_plib` automatically completes all library preparation steps and creates a Milkyway library ready for place and route.

If you select this option, you must manually run the steps to create a Milkyway library (see Figure 5-1 on page 5-3). Use this option for special cases or for advanced library preparation.

Figure 5-7 shows the Same Name Cell Resolution section of the PLIB/PDB In dialog box, followed by a description of each option.

Figure 5-7 PLIB/PDB In Dialog Box: Same Name Cell Resolution Options



Make New Cell Version

This option is selected by default. In the default mode, a new cell version is created from the cells defined in the PLIB/PDB file. The old version is maintained.

Overwrite Existing Cell

This option is not selected by default. If you select this option, the latest cell versions in the Milkyway library are overwritten by the PLIB/PDB cells.

Merge with Existing Cell

This option is not selected by default. If you select this option, the latest cell version includes the information defined in the PLIB/PDB file and the existing Milkyway library information. Cell-related information such as macro pin and macro obstructions is appended to the existing information.

Ignore PLIB/PDB Cell

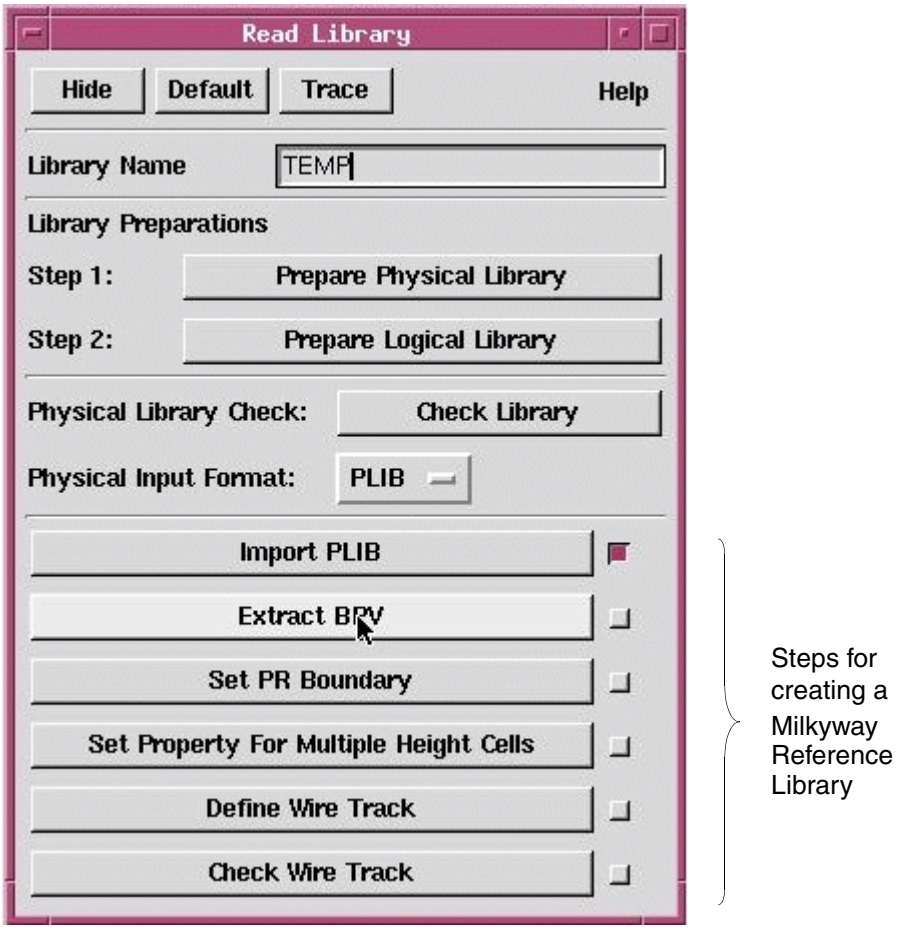
This option is not selected by default. If you select this option and have an existing Milkyway library, cells defined in the PLIB/PDB file with the same name are ignored when you import the PLIB/PDB file.

Creating a Milkyway Library by Using read_lib

Although running the `read_plib` command automates the five steps required to create a Milkyway library that is ready for place and route, in situations where the automated flow does not work for your library you can manually run the five-step flow with the `read_lib` command. When you invoke the `read_lib` command or choose Cell Library > Library Preparation in the menu, the Read Library dialog box opens and guides you through the steps required to create a place-and-route-ready Milkyway library.

Figure 5-8 shows the library preparation option settings in the Read Library dialog box that are automatically used during `read_plib`. The last step, Check Wire Track, is not part of the automatic flow. An optional step, you can run it manually at the end of the `read_lib` flow.

Figure 5-8 Read Library Dialog Box



Step 1: Importing PLIB and PDB Files, Using read_lib

Step 1 for `read_lib` (import PLIB/PDB) has the same default switches as described in step 1 for `read_plib` (see "read_plib Options" on page 5-9), except that the Advanced Library Prep Mode option is selected in the Read PLIB/PDB In dialog box for running the manual flow.

To begin the `read_lib` advanced flow, open the Read Library dialog box by typing `read_lib` on the Milkyway command line or choose Cell Library > Library Preparation in the menu.

Complete the following steps in the Read Library dialog box:

1. Specify the library name.
2. Click Prepare Physical Library.

3. Click the Physical Input Format menu, and choose PLIB.
4. Click Import PLIB.

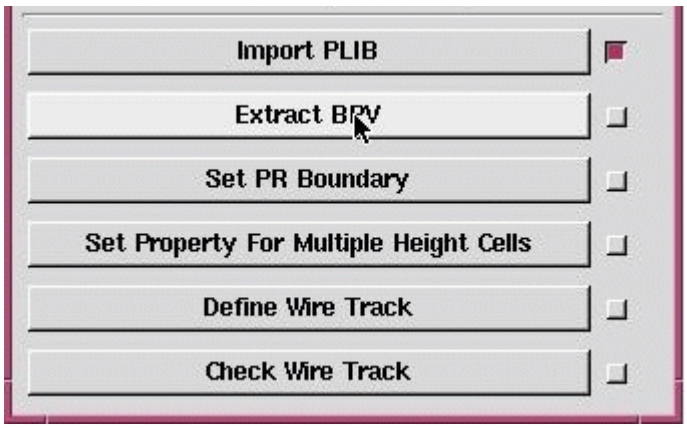
   When you click Import PLIB, the PLIB/PDB In dialog box opens and the Advanced Library Prep Mode option is automatically selected.

   Complete the remaining steps in the PLIB/PDB In dialog box.
5. (Optional) Specify the PLIB/PDB file in the Tech PLIB/PDB File and Cell PLIB/PDB Files boxes. There can be multiple methods for reading in PLIB and PDB files. For more information about the methods for reading in PLIB and PDB files, see “Step 1: Importing PLIB and PDB Files, Using read_plib” on page 5-5.
6. Click OK.

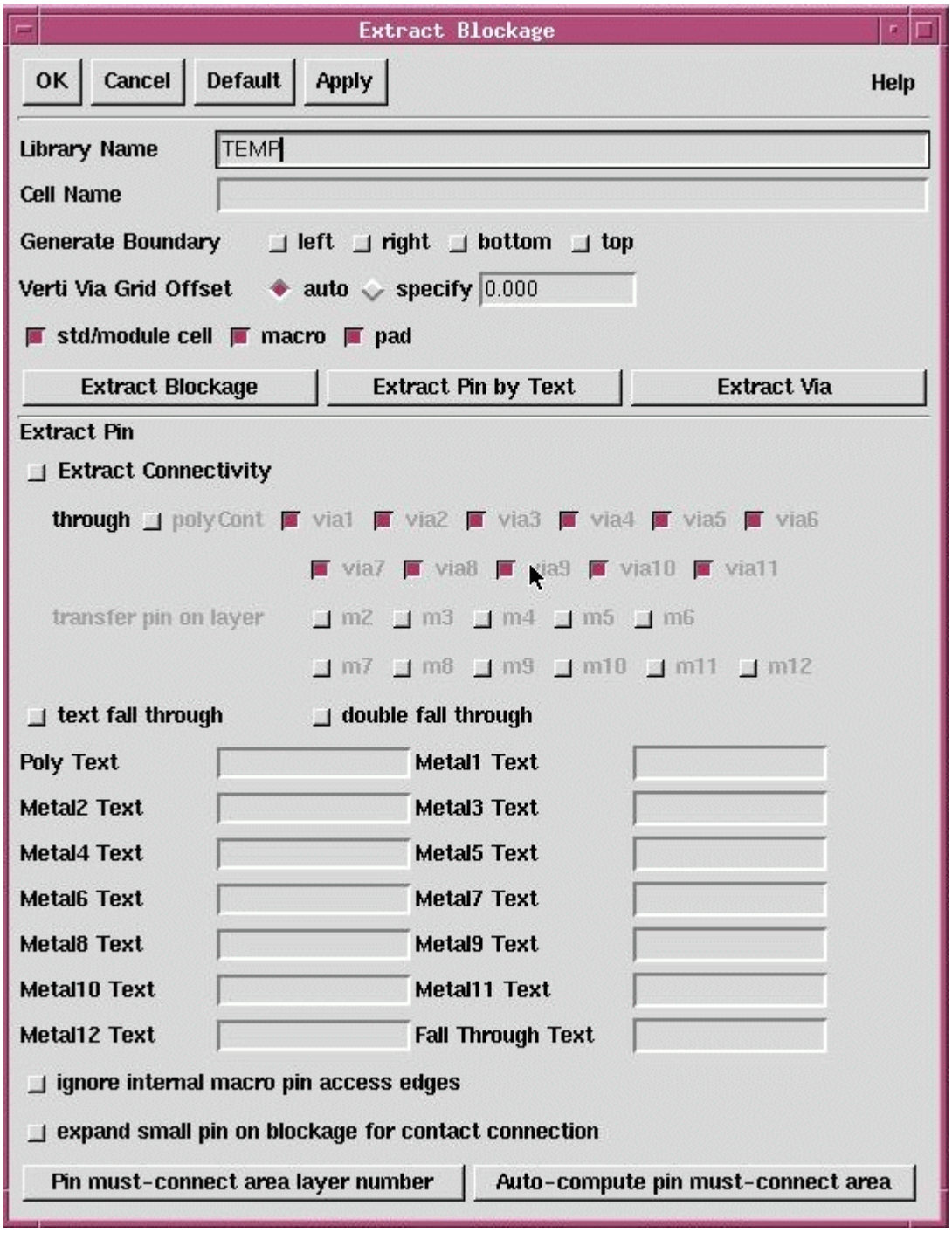
You can now manually import the PLIB file using the advanced flow. (See step 2.)

Step 2: Extracting Blockage, Pin, and Via and Generating FRAM View

1. Run the `auExtractBlockagePinVia` command or select Extract BPV in the Read Library dialog box to open the Extract Blockage dialog box.



   Complete the following steps in the Extract Blockage dialog box to generate a FRAM view on the current Milkyway reference library and to extract blockage, pin, and via information.



2. Specify the library name (if it is not provided).

3. Select the options you want.

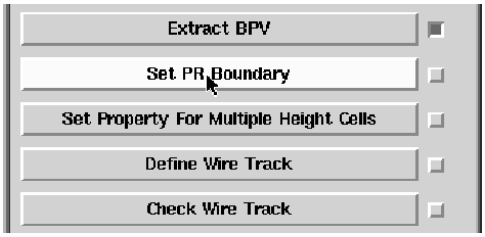
   For information about `auExtractBlockagePinVia` options, see Physical Implementation Online Help.

4. Click OK.

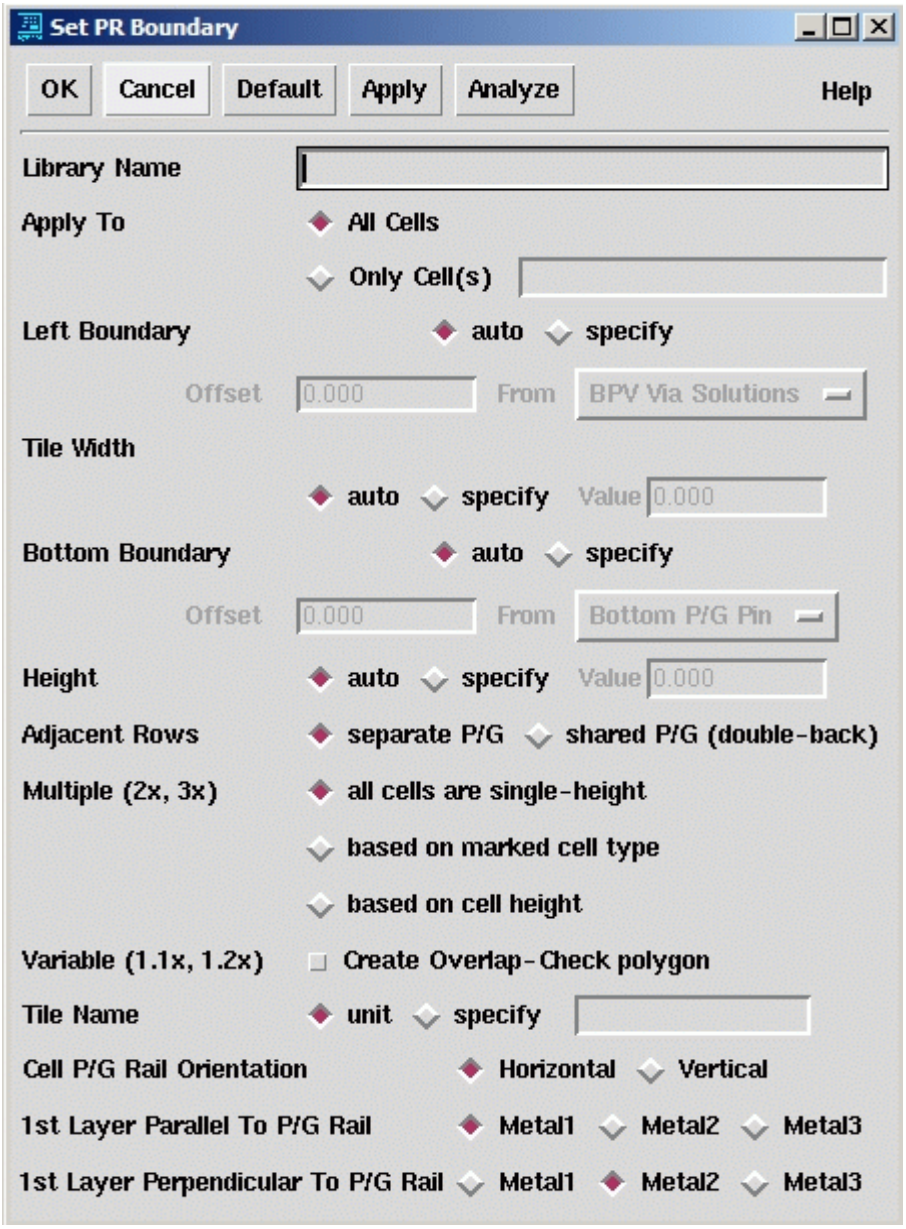
Electrical equivalence (EEQ) information for pins is extracted, based on the EEQ information provided in the PLIB/PDB file.

Step 3: Setting the Place and Route Boundary

1. Run the `auSetPRBdry` command or select Set PR Boundary in the Read Library dialog box to open the Set PR Boundary dialog box.



   Complete the following steps in the Set PR Boundary dialog box to determine the place and route boundary in the current Milkyway reference library.



2. Specify the library name. If your library was created with PLIB/PDB data, Milkyway uses the “based on marked cell type” option in the Multiple (2x, 3x) section of the dialog box. The PLIB/PDB format provides enough information to determine the place and route boundary.

You must specify the library name for the marked cell type to be effective. After you specify the library name, the “based on marked cell type” option is automatically selected in the dialog box.

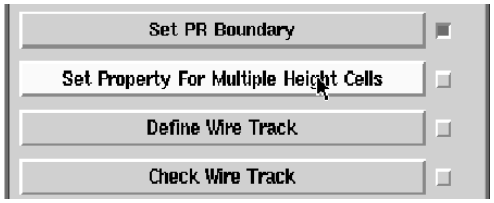
3. Select the options you want.

   For information about `auSetPRBdry` options, see Physical Implementation Online Help.

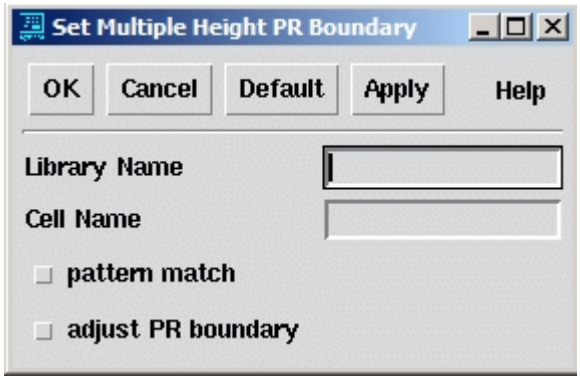
4. Click OK.

Step 4: Specifying the Multiple-Height Place and Route Boundary

1. Run the `cmSetMultiHeightProperty` command or select Set Property For Multiple Height Cells in the Read Library dialog box to open the Set Multiple Height PR Boundary dialog box.



   Complete the following steps in the Set Multiple Height PR Boundary dialog box to set the place and route properties for multiple-height cells in the current Milkyway reference library.



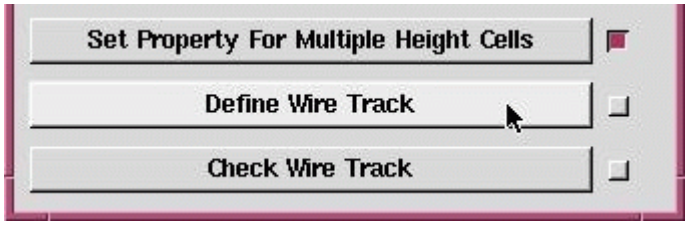
2. Specify the library name (if it is not provided).
3. Select the options you want.

   For information about `cmSetMultiHeightProperty` options, see Physical Implementation Online Help.

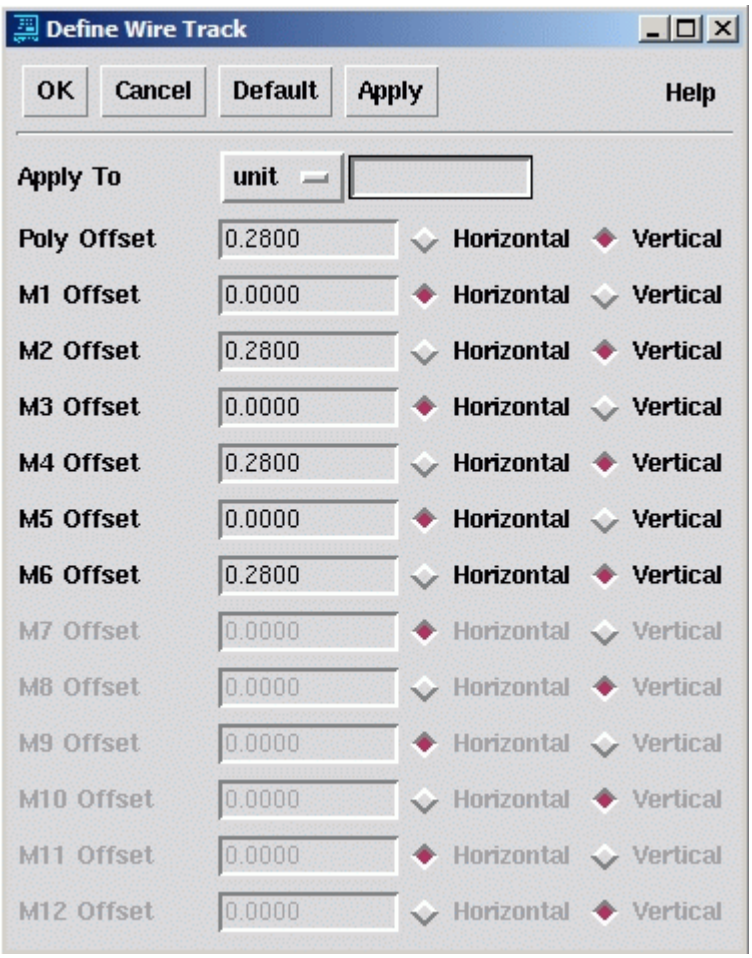
4. Click OK.

Step 5: Defining Wire Tracks

1. Run the `axgDefineWireTracks` command or select Define Wire Track in the Read Library dialog box to open the Define Wire Track dialog box.



In the Wire Track dialog box, the offsets and metal directions are already specified. They are derived from the Input PLIB/PDB file.



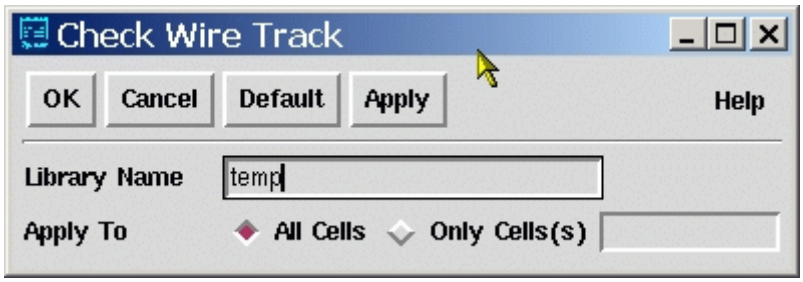
2. Click OK.

Step 6: (Optional) Checking Wire Tracks

The `read_plib` command in default mode does not run this step. Therefore if you need to check your wire tracks, you must manually run this step.

1. Run the `axgCheckWireTrack` command or select Check Wire Track in the Read Library dialog box to open the Check Wire Track dialog box.

Complete the following steps in the Check Wire Track dialog box to check wire tracks in the current Milkyway reference library.



2. Specify the library name (if it is not provided).
3. Click OK.

   If you see pins without a desirable access point on the grid, redo the Define Wire Track step (modify the outing layer's offset). Note that it is acceptable to have a small percentage of pins that are not accessible. (A higher percentage of inaccessible pins can result in the router's taking longer to complete the design.)

6

Data Preparation Using PDEF

The steps for translating Physical Design Exchange Format (PDEF) data are covered in this chapter, which contains the following sections:

The PDEF Interface

The Physical Design Exchange Format provides a standard medium for passing physical design information between high-level design and physical design applications. PDEF can also be used as a back-annotation medium for passing physical design information from back-end applications (floorplanning and layout) to those on the front end. This format defines a standard set of attributes describing particular types of physical information.

The Milkyway PDEF interface discussed in this section uses a common PDEF reader and writer to provide consistency between Synopsys tools.

Note:

The `aprPDEFIn` command has been replaced by `read_pdef` and the `aprPDEFOut` command has been replaced by `write_pdef`. The `aprPDEFIn` and `aprPDEFOut` commands are no longer supported.

This section includes the following subsections:

PDEF Import and Export Commands

The following commands read and write in the PDEF interface:

- `read_pdef`

  Imports (reads) a PDEF file to a Milkyway design cell.

- `write_pdef`

  Exports (writes) a Milkyway design cell to a PDEF file.

The `read_pdef` and `write_pdef` commands are available in both Scheme and Tcl modes. The details of these commands are described later in this section.

For more information about the Scheme and Tcl modes, see the Milkyway Environment Extension Language Reference Manual.

Importing PDEF Into Milkyway Using read_pdef

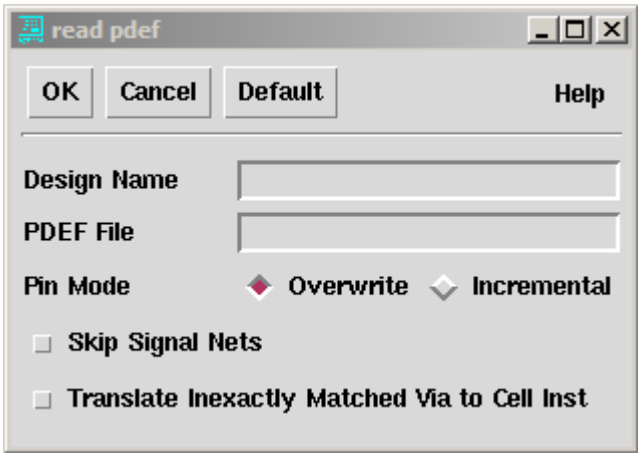
This section discusses the read pdef dialog box and options used to import PDEF to the Milkyway database. For information about the write pdef dialog box and options used to export PDEF from the Milkyway environment, see “Exporting PDEF From Milkyway Using write_pdef” on page 6-4.

To open the read pdef dialog box, enter the following on the command line:

```
Milkyway > read_pdef
```

The read pdef dialog box opens, as shown in Figure 6-1.

Figure 6-1 read pdef Dialog Box



read_pdef Options

The read pdef dialog box contains the following options:

Design Name (Required)

Specifies the name of the design cell into which the PDEF file is read.

PDEF File (Required)

Specifies the name of the PDEF input file.

Pin Mode

Has two options:

- Overwrite (Default)

  Specifies that the existing pins of the port to be read in are deleted.

- Incremental

  Specifies that the existing pins of the port to be read in are not deleted.

If the Overwrite mode is specified and there are already two pins in the design that belong to port_1, such as pin 1 and pin 2, and there are two pins in the PDEF file to be read in, such as pin 3 and pin 4, then pin 1 and pin 2 are deleted and pin 3 and pin 4 are read in.

Skip Signal Nets

Specifies that all signal nets are skipped when you want to ignore preroutes.

Translate Inexactly Matched Via to Cell Inst

Specifies that inexactly matched vias are stored as cell instances. If this option is not specified, inexactly matched vias are stored as contact arrays and wires.

Inexactly matched vias are vias that match a Milkyway technology contact code's cut dimensions. However, for inexactly matched vias, the enclosure dimensions are larger than the contact code's enclosure dimensions.

read_pdef Example (Tcl Mode)

In the following example, the PDEF file name is "in.pdef" and the design name is "top". The core area will not be updated, all signal nets will be skipped, and the inexactly matched via will be stored as a cell instance.

```
Milkyway> read_pdef -pdef_file_name "in.pdef" \
-design "top" -dont_update_core_area \
-skip_signal_nets -trans_inexact_match_via_to_cell_inst
```

Exporting PDEF From Milkyway Using write_pdef

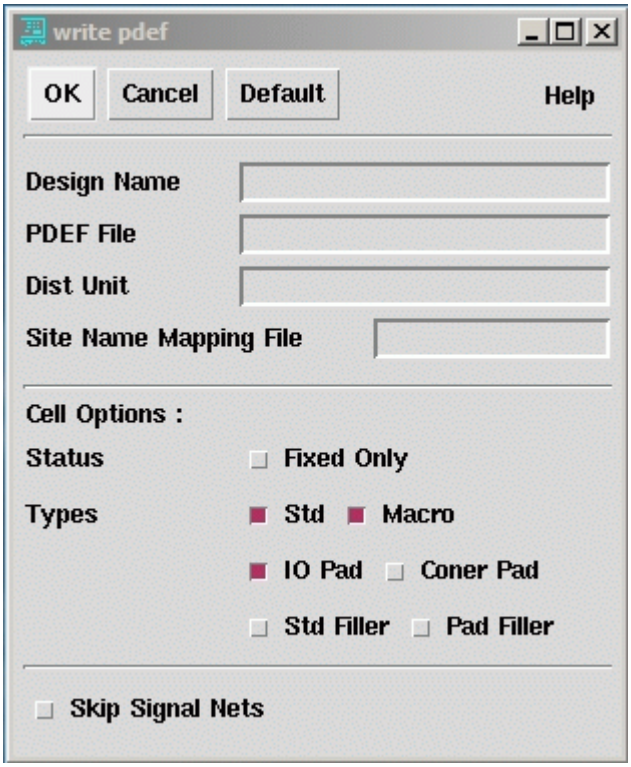
This section discusses the write pdef dialog box and options used to export PDEF from the Milkyway database. (For information about the read pdef dialog box and options used to import PDEF into the Milkyway environment, see "Importing PDEF Into Milkyway Using read_pdef" on page 6-3.)

To open the write pdef dialog box, enter the following on the command line:

```
Milkyway > write_pdef
```

The write pdef dialog box opens, as shown in Figure 6-2.

Figure 6-2 write pdef Dialog Box



write_pdef Options

The write pdef dialog box contains the following options:

Design Name (Required)

Specifies the name of the design cell that will be output. This is the Milkyway library that contains the design cell. The box can include the path of the library. Otherwise, the library path is resolved from the current working directory.

PDEF File (Required)

Specifies the name of the PDEF file to which the design cell is written. If the file already exists, Milkyway overwrites it. If the file does not exist, Milkyway creates it. The box can include the path of the output file. Otherwise, the file is written to the current working directory.

Distance Unit (Optional)

Specifies the distance unit used in the output PDEF file. By default, it is the distance unit defined in the Milkyway library.
Valid values: 100, 200, 1000, and 2000.

For any other values, `write_pdef` exits with a warning message and writes the default value. The default value is the length precision value of the technology file. If no value is specified, the default value is written in the output PDEF file.

Site Name Mapping File

Specifies the site name mapping file name. The site name mapping file is used to change the site name in the output PDEF file. The format is as follows:

```
original_site_name renamed_site_name
```

Cell Options

Status (Fixed Only)

Specifies that only the fixed cell should be output.

Types (Std, Macro, IO Pad, and so on)

Specifies that only the types of cells selected should be output. If all types are specified, all types of cells will be output.

Skip Signal Nets

Specifies that signal nets will not be output.

write_pdef Example (Tcl Mode)

In the following example, the output PDEF file name is “out.pdef” and the design cell to be output is “top”. The distance unit is the distance unit that is defined in the Milkyway library, all signal nets will be skipped, and the wire load cell name will be output in the PDEF file.

```
Milkyway> write_pdef -pdef_file_name "out.pdef" -design "top" -skip_signal_nets \
-output_wire_load_name
```

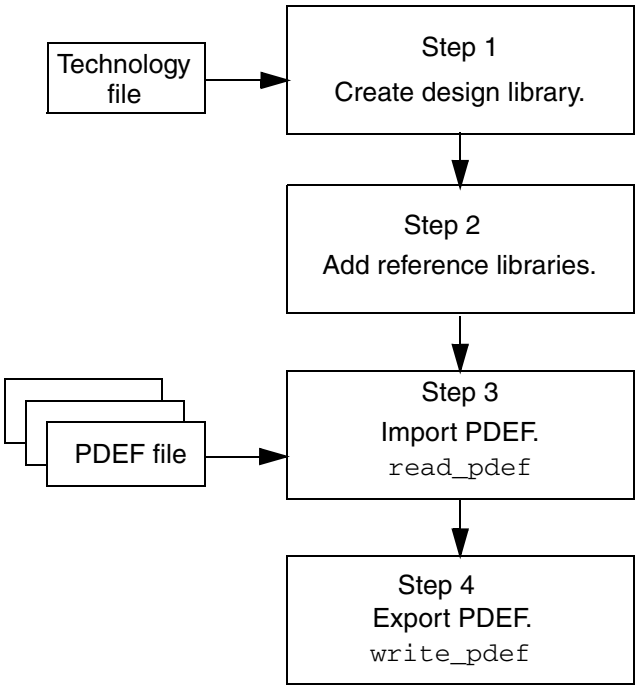
The PDEF Flow

The Milkyway PDEF flow supports Synopsys physical implementation tools as well as third-party place and route tools. The flow is discussed in the following section.

PDEF Input and Output Flow

This section describes the forms and options used to successfully import a PDEF file to a Milkyway cell with `read_pdef` and to export the data to a PDEF file with `write_pdef`. Figure 6-3 is an overview of the input and output flow.

Figure 6-3 Flow for Importing and Exporting a PDEF File



1. Create a design library from a technology file.
2. Add the required reference libraries.
3. Import the PDEF file to the design library, using the `read_pdef` command.
4. Export the design information to a PDEF file, using the `write_pdef` command.

IEEE PDEF Syntax

IEEE PDEF is a recognized standard supported by the IEEE organization. The PDEF v. IEEE 1481-1998 (IEEE PDEF) syntax used with Synopsys tools is described in these sections:

- Limitation of IEEE PDEF
- Third-Party Tool and Vendor Requirements
- Guidelines for Using IEEE PDEF
- IEEE PDEF Design Definition Syntax

Synopsys provides some IEEE PDEF user attributes that are specific to Synopsys. These user attributes have special meaning to Synopsys for floorplanning. See the *Physical Compiler User Guide, Volume 1*.

Limitation of IEEE PDEF

In IEEE PDEF, placement obstructions are not part of the syntax.

Third-Party Tool and Vendor Requirements

Third-party tools used in the design flow must read and write IEEE PDEF.

Guidelines for Using IEEE PDEF

When you use IEEE PDEF, observe the following guidelines:

- Object names must be the full path name. Do not build names by concatenating local, short names with CLUSTER names to create full path names.
- Buses must be bit-blasted.

  Describe the bus as the individual nets A[0], A[1], and A[2], not as A[0:2]. A\[1\] is treated as being equivalent to A[1].
- Do not use wildcard characters.
- The combination of RESTRICTION attributes at different points in the physical hierarchy allows the tool to be very specific about the groups of cells and nets that are reoptimized. For example, CLUSTERs can be effectively marked as `dont_touch` by applying the `fixed_logic`, `no_new_cells`, and `no_new_buffers` attributes.

IEEE PDEF Design Definition Syntax

The IEEE PDEF design file contains chip-level I/O port location information in addition to cluster and cell information. The chip-level I/O ports are described as pins on the topmost cluster of the design. If your physical hierarchy matches your logical hierarchy, pins on subclusters in the physical hierarchy can correspond to ports on submodules in your logic hierarchy.

Cell orientation information is necessary and used by Physical Compiler.

Cell location information in the design IEEE PDEF file gives the location of the origin of each cell in the layout.

(CLUSTERFILE

(PDEFVERSION "IEEE 1481-1998")

(DESIGN qstring)

The read_pdef command uses this value to choose which design in memory to annotate the PDEF file to. The read_pdef -design command overrides this value. The write_pdef command writes out the actual top-level design name from the in-memory design.

(DATE qstring)

Whatever date specified by qstring is overwritten by write_pdef, which writes the date when the command is issued.

(VENDOR qstring)

Whatever vendor name specified by qstring is overwritten by write_pdef, which writes (VENDOR "Synopsys, Inc.").

(PROGRAM qstring)

Whatever program name specified by qstring is overwritten by write_pdef, which writes (PROGRAM "Design Compiler").

(VERSION qstring)- Whatever version name specified by qstring is overwritten by write_pdef, which writes the software version used. For example (VERSION "2003.06").

(DIVIDER . | / | : | |) -This construct is used by read_pdef to annotate information to the design.db. The write_pdef command always prints out (DIVIDER /).

(PIN_DELIMITER . | / | : | |) - This construct is used by read_pdef to annotate information to the design.db. The write_pdef command always prints out (PIN_DELIMITER /).

(BUS_DELIMITER [| { | (| < | : | . [ ] | } |) | >]) - The read_pdef command requires that the bus delimiter characters are [ ]. The write_pdef command always prints out (BUS_DELIMITER [ ]).

(NETLIST_TYPE VERILOG | VHDL87 | VHDL93 | EDIF) - This construct is ignored by read_pdef. The write_pdef command always prints out (NETLIST_TYPE VERILOG).

(DISTANCE_UNIT float) - Although the default value is defined as 1.0, it is recommended to explicitly specify the value. The write_pdef command always prints out (DISTANCE_UNIT 0.001).

(DESIGN_FLOW cell_locations | cluster_locations |pin_locations | datapath_constraints | datapath_placement |gates | logical_clusters | obstruction | power routes | complete_routes)

This construct is ignored by read_pdef. The write_pdef command always prints out (DESIGN_FLOW " ").

(NAMEPREFIX integer path)

For the IBM/ChipBench flow, this construct causes problems. Set the variable pdefout_full_path_name to true before writing out PDEF. This construct (if present) is used by read_pdef to annotate information to the design.db. The write_pdef command by default uses this construct to reduce the PDEF file size.

(CORE_AREA x y x y)

Construct is specific to Synopsys. If not specified, read_pdef derives the core area value from the bounding box that bounds all the SITE definitions for the floorplan. Used by coarse placement for cell placement. This construct (if present) is used by read_pdef to annotate information to the design.db. This construct is written out by write_pdef.

(DEF_TRACK_0 "Y" "value1" "value2" "value3" "layer_names")

Construct is specific to Synopsys, needed to ensure correct legalization of cells. Converted from the DEF construct "TRACKS Y value1 DO value2 STEP value3 LAYER layer_names ;". "Y" indicates the tracks step in the Y direction (horizontally). The "value1" is the starting point for the first track, "value2" is the number of tracks to insert, "value3' is the track pitch or step distance, "layer_names" is the list of routing layers. Note the double quotation marks around each value. This construct (if present) is used by read_pdef to annotate information to the design.db. This construct is written out by write_pdef.

(DEF_TRACK_1 "X" "value1" "value2" "value3" "layer_names")

Construct is specific to Synopsys, needed to ensure correct legalization of cells. Converted from the DEF construct "TRACKS X value1 DO value2 STEP value3 LAYER layer_names ;". "X" indicates the tracks step in the X direction (vertically). The "value1" is the starting point for the first track, "value2" is the number of tracks to insert, "value3 is the track pitch or step distance, "layer_names" is the list of routing layers. Note the double quotation marks around each value. This construct (if present) is used by read_pdef to annotate information to the design.db. This construct is written out by write_pdef.

(SITE row_name "site_name" "x y" "H" "orientation" "spacing" "number")

Construct is specific to Synopsys, needed to ensure correct legalization of cells. Converted from the DEF construct "ROW row_name site_name x y orientation DO number BY 1 STEP spacing site_width ;". Note the double quotation marks around each value. Note that

site_width is dropped, being derived from the physical .pdb file. This construct (if present) is used by read_pdef to annotate information to the design.db. This construct is written out by write_pdef.

(SITE row_name "site_name" "x y" "V" "orientation" "spacing" "number")

Construct is specific to Synopsys, needed to ensure correct legalization of cells. Converted from the DEF construct "ROW row_name site_name x y orientation DO 1 BY number STEP spacing site_width ;". Note the double quotation marks around each value. Note that site_width is dropped, being derived from the physical .pdb file. This construct (if present) is used by read_pdef to annotate information to the design.db. This construct is written out by write_pdef.

(LAYER_DEF

(LAYER qstring layer_ID)

The qstring name is the string that describes the routing layer, for example "Metal1". This corresponds to the physical library name for the routing layer. References to the LAYER attribute by other sections of the PDEF file are by layer_ID only.

)

Any reference to a layer_ID that is not declared in the LAYER_DEF section is treated by read_pdef as a placement obstruction. For example, if LAYER_DEF declares layer IDs 10 through 20, referring to layer ID 1 is treated as a placement obstruction.

(VIA_DEF

(VIA name via_ID

The name is the string that describes the via, for example "Via1". This name corresponds to the physical library name for the VIA. References to the VIA attribute by other sections of the PDEF file are by via_ID only.

(VIA_PATTERN qstring)

Construct is specific to Synopsys, needed to describe vias containing multiple cut definitions and corresponds to the DEF PATTERNAME construct.

(VIA_RECT layer_name "x1 y1 x2 y2")

Construct is specific to Synopsys for declaring the various layers and rectangles used to describe a via. Multiple VIA_RECT statements are allowed. Note the double quotation marks around the x- and y-coordinates.

)

)

/* Comments */

Comments can span multiple lines. Comments are not stored in the design.db during the read_pdef process, and thus the output PDEF (from using write_pdef) does not have your comments.

(CLUSTER qstring

The qstring name is the string that describes the cluster.

(user attributes)
Any user-specified attributes are stored in the design.db by read_pdef and printed out by write_pdef. Physical Compiler takes no action on these attributes.

(RECT x1 y1 x2 y2)

Optional construct. Spedifies the specific XY coordinates for the CLUSTER location and the outline of the design. Multiple RECT statements are allowed and will describe a rectilinear block outline.
(OBSTRUCTION layer_ID x1 y1 x2 y2)

Multiple OBSTRUCTION statements are allowed.

(CONTENT_LOCATIONS relative | absolute)

relative: All coordinates within the cluster are relative to the origin of the CLUSTER, which in turn is described relative to the CLUSTER's parent. The cluster origin is the bottom-left coordinate of the CLUSTER's RECT declarations.
absolute: All coordinates within the CLUSTER are assumed to be absolute, and the origin is the design origin of (0 0). The write_pdef command always produces PDEF with absolute coordinates.

(PIN name

The name is the logical PIN name, for example "top/data_out4", that corresponds to the port definition in the design netlist. Physical-only ports declared in the input PDEF file must use a unique name that does not conflict with an existing name.

(LOC x y)

Specifies the location of the PIN object. The coordinates refer to the location of the pin's origin.This construct is written out by write_pdef.

(RECT x1 y1 x2 y2)

Specifies the specific x- and y-coordinates for the PIN outline. Multiple RECT statements are allowed. This statement takes precedence over the LOC statement. This construct is written out by write_pdef.

(LAYER layer_ID)

Specifies the routing layer that the PIN's RECTangle statement applies to. Multiple LAYER statements are allowed, but there must be multiple RECT statements (one for each LAYER statement) and the association is implied by the order of the RECT and LAYER statements.This construct is written out by write_pdef.

(TYPE control | data | tri | gnd [qstring] | pwr [qstring] | clock | set | reset | test_en [qstring] | scan_in [qstring] | scan_out [qstring] | reserved)

This attribute is stored in the design.db by read_pdef. Physical Compiler takes no action on this attribute, but it is written out by write_pdef.

(PIN_LAYER "layer (x1 y1) (x2 y2) ")

Construct is specific to Synopsys, needed to explicitly associate a specific layer and the rectangle of the PIN. This construct is written out by write_pdef.

)

(CELL name

The name is the logical CELL instance name, for example "top/module/cell1". This construct describes (logical) netlist leaf cells that connect to other cells in the netlist to implement the user's desired functionality. Physical-only cells declared in the input PDEF file must use a unique name that does not conflict with an existing name.

(GATE_NAME name)

The name is the cell's reference from the library, for example, "nand2x".

(X_BOUNDS x_lower x_upper)

(Y_BOUNDS y_lower y_upper)

Use of X_BOUNDS and/or Y_BOUNDS would describe a bounding box that the cell must be placed inside. This optional construct is used only if the cell has been grouped in a bounding box (by using create_bounds).

(LOC x y)

Describes a specific location that the cell must be placed at. If both X_BOUNDS/ Y_BOUNDS and LOC constructs are specified, the LOC statement is used because it is more specific. The x- and y-coordinates for a CELL are always assumed to be the cell's lower-left corner.

(ORIENT "0" | "90" | "180" | "270" | "0-mirror" | "90-mirror" | "180-mirror" | "270-mirror")

The orientation of the cell. If this construct is missing, the cell’s orientation is assumed to be "0" (or N in DEF syntax).

(RESTRICTION FIXED_PLACEMENT)

Specifies that placement of the cell must not be changed in any way. This also creates a dont_touch attribute to prevent logic optimizations.

(TYPE qstring)

Any string can be inserted, and this user-defined attribute is stored by read_pdef and written back out by write_pdef.

(TYPE spare | used)

This attribute is flagged as a user attribute by read_pdef but stored in the design.db. Physical Compiler takes no action on this attribute, but it is written out by write_pdef.

)

(NET name

The name is the logical name of the net. If the net name does not exist in the design netlist, the NET is treated as a physical net. That is, it is converted by read_pdef to a PNET, and the result of write_pdef is that this net is described by PNET, not NET, statements. Note that the TYPE statements (being NET attributes) must be declared before the ROUTE statements.

(TYPE control | data | tri | gnd [qstring] | pwr [qstring] | clock | set | reset | test_en [qstring] | analog | low_noise | hi_perf | reserved)

This attribute is stored in the design.db by read_pdef. Physical Compiler takes no action on this attribute but it is written out by write_pdef.

(TYPE qstring)

Any string can be inserted, and this user-defined attribute is stored by read_pdef and written back out by write_pdef.

(ROUTE

Each ROUTE statement describes a set of connected net segments. If the NET is formed from several disjoint sets of segments, there is several ROUTE statements, one for each set.

(RESTRICTION EDIT)

This attribute is stored in the design.db by read_pdef. Physical Compiler takes no action on this attribute, but it is written out by write_pdef.

(LAYER_WIDTH layer_ID width)

Describes the routing segment's layer ID and the width of metal used for the following segment description. This construct is written out by write_pdef.

(layer_ID point point) //segment description

Describes the routing topology of the net, each point describing an x- and y-coordinate.The specified coordinates must be orthogonal in nature; do not change both X and Y values at the same time. The use of "*" to indicate that the previous value of X or Y is used is highly recommended.

(VIA via_ID point)

Describes the location of VIA objects, using the via ID from the via DEF description and specifying the location of the via (using the via's origin).

)

(PNET name

The name is the (physical) name of the net. These objects typically do not exist in the design netlist. Note that the TYPE statements (being PNET attributes) must be declared before the ROUTE statements.

(TYPE data | gnd [qstring] | pwr [qstring] | reserved)

This attribute is stored in the design.db by read_pdef. Physical Compiler relies on (TYPE pwr) and (TYPE gnd) to identify power net statements.

(TYPE qstring)

Any string can be inserted, and this user-defined attribute is stored by read_pdef and written back out by write_pdef.

(ROUTE

Each ROUTE statement describes a set of connected net segments. If the PNET is formed from several disjoint sets of segments, there is several ROUTE statements, one for each set.

(RESTRICTION EDIT)

This attribute is stored in the design.db by read_pdef. Physical Compiler takes no action on this attribute but it is written out by write_pdef.

(LAYER_WIDTH layer_ID width)

Describes the routing segment's layer ID and the width of metal used for the following segment description. This construct is written out by write_pdef.

(layer_ID point point) //segment description

Describes the routing topology of the net, each point describing an XY coordinate.The specified coordinates must be orthogonal in nature, do not change both X and Y values at the same time. The use of "*" to indicate that the previous value of X or Y is used is highly recommended.

(VIA via_ID point)

Describes the location of via objects, using the via ID from the via DEF description and specifying the location of the via (using the via's origin).

)

)

)

7

Processing the Cells

After you translate the physical design data to Synopsys layout cells, you need to process the cells by taking the steps in the following sections:

- Identifying Power and Ground Ports
- Smashing Hierarchical Cells
- Extracting Pins and Blockages
- Specifying Port Information for Advanced Operations
- Using Multiple-Height Cells
- Using Variable-Height Cells
- Using Multiple Cell Classes
- Setting the Place and Route Boundary
- Defining Wire Tracks

Identifying Power and Ground Ports

After you import the cells to your library, you must identify the power and ground ports in your cells, by using the `dbSetCellPortTypes` functions. (For more information, see the *Milkyway Environment Extension Language Reference Manual*.) If you change power or ground port information after pin and blockage extraction, you must rerun pin and blockage extraction for the cells you changed.

Note:

At this point in the design flow, you use `dbSetCellPortTypes` functions only to identify power and ground ports. After pin and blockage extraction, you create a port information file containing `dbSetCellPortTypes` functions to specify the port information required for advanced operations. For more information, including a full syntax description, see "Specifying Port Information for Advanced Operations" on page 7-11.

Syntax

```
dbSetCellPortTypes libName cellName '(
                (powerPortName "Power")
                (groundPortName "Ground")
                )
```

where the arguments are as follows:

libName	Name of the library for which you are identifying power and ground ports Valid values: Name of any library in the current directory
cellName	Name of the cell for which you are identifying power and ground ports Valid values: Name of any cell in the specified library, or an asterisk (*) to indicate all cells

Note:

The asterisk does not include cells for which you apply a `dbSetCellPortTypes` function that uses a specific *cellName*.

powerPortName	Name of the power port in the cell(s)
groundPortName	Name of the ground port in the cell(s)

Example

```
      dbSetCellPortTypes "demo" "*" '(
            ("VDD" "Power" )

("GND" "Ground" )
            ) #f
```

This function identifies power and ground ports for all cells in the library `demo` (except any cells for which you apply a `dbSetCellPortTypes` function that uses a specific cell name).

Smashing Hierarchical Cells

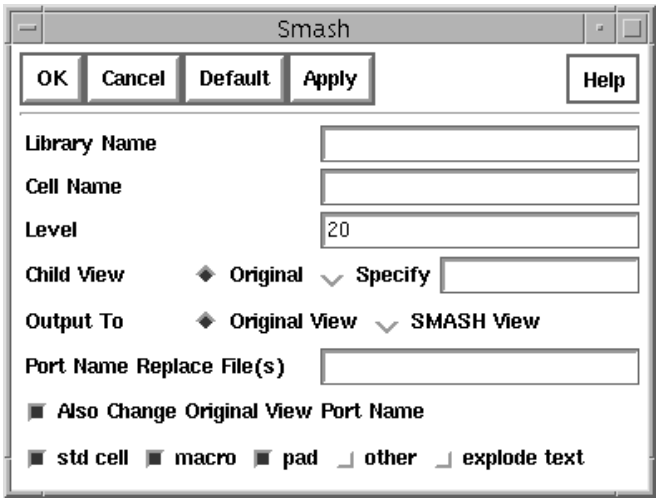
To perform processing that involves geometries inside the hierarchy of a layout cell, you must first smash the layout cell to a level sufficient to provide the access required to perform the processing. For example, to remove unnecessary metal2 from metal1 pins inside a cell instance that is inside a layout cell, you must first smash the layout cell to a level that makes the metal1 pins accessible.

During the creation of pin/blockage cells, Milkyway might need access to geometries inside the hierarchy of layout cells to extract the pins and create the blockage and via regions. In some cases, however, you might want to limit the access. For example, you might want to limit the hierarchical access in blocks to avoid unnecessary processing and to achieve the blockage areas you want.

To smash one or more layout cells,

1. Enter `cmSmash` or choose Cell Library > Smash.

   The Smash dialog box appears.



2. Fill in the Smash dialog box. For detailed descriptions of command options, see Physical Implementation Online Help.
3. Click OK.

Extracting Pins and Blockages

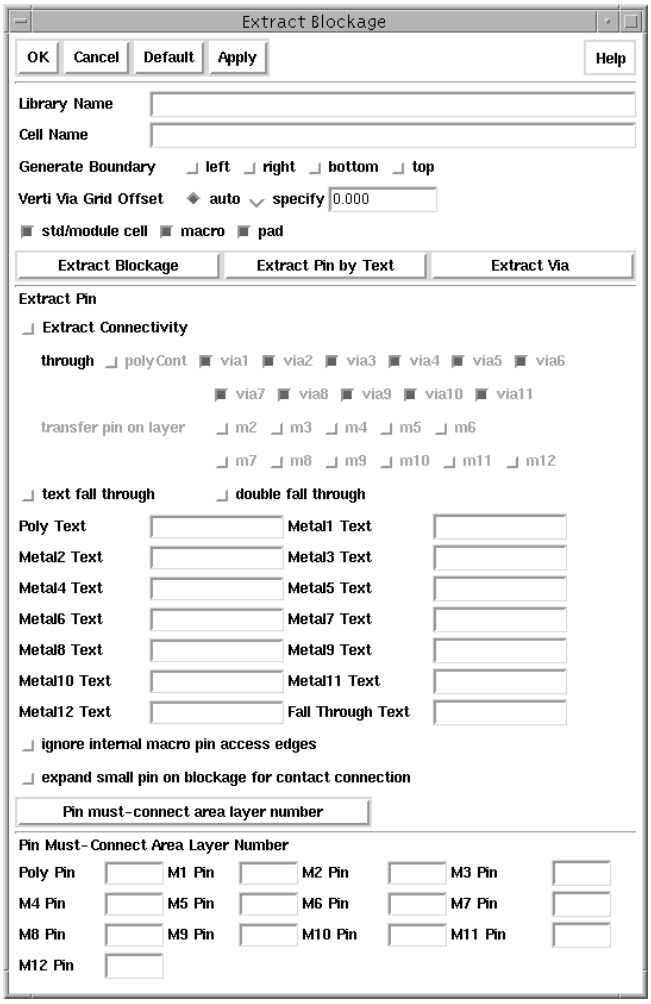
After you create and process the layout cells, you need to create pin/blockage cells for each layout cell in your library. The pin/blockage cells contain the following:

- Pins and blockage areas (areas where no routing on a particular layer can occur)
- Via regions (areas where vias can be placed during routing)
- Pin solutions (solutions for routing to pins)

If your library contains large macro cells with many small blockage areas, you might want to use the `auExtractMacroPins` command to extract pin geometries and delete blockage geometries in these cells before you create the pin/blockage cells.

For cells created in the Milkyway database, you can perform pin and blockage extraction by using the `geNewMakeMacro` command. Use `geNewMakeMacro` to create pin/blockage (.fram) cells for a macro cell created with a tool based on Milkyway and to place the cell in a Synopsys design.

You create pin/blockage cells by using the `auExtractBlockagePinVia` command, which displays the Extract Blockage dialog box.



For detailed descriptions of command options for the Extract Blockage dialog box, see `auExtractBlockagePinVia` in Physical Implementation Online Help.

After entering values and selections for the options in this dialog box, click OK to perform the extraction, which creates the pin/blockage cells.

You can run this process with different options for standard/module cells, macro cells, and pads. To do so, select the types of cells you want to process with the std/module cell, macro, and pad options. After you perform extraction, you can use Library > Check (`cmCheckLibrary`) to check your library. For detailed information about using the `cmCheckLibrary` command, see Physical Implementation Online Help.

Creating Via Regions and Pin Solutions

During pin and blockage extraction, Milkyway creates via regions and pin solutions for standard cells. Via regions appear as rectangular outlines and identify areas in which the router can place vias during detail routing. Pin solutions consist of one or more “virtual” objects, which the router can use when making a connection to the pin.

Identifying Pins

Milkyway identifies pins by reading text labels associated with geometries in the cells. Determining which text label refers to which geometry requires information regarding which layers are used for the text labels. You provide this information in the Extract Blockage dialog box.

Layers for text labels can follow any of the following conventions:

- Text is on the same layer as the geometry it identifies.
- Text is on a different layer than the geometry it identifies.
- All text is on the same layer.

In the case of multiple layer pins with text on only one layer, you select Extract Connectivity to instruct Milkyway to trace the connectivity of the pin and extract the entire pin.

When Text Is on the Same Layer as Its Geometry

When the text layers match the layers for the geometries they identify, you only need to make sure that “text fall through” is not selected in the dialog box. You do not need to specify the text layers associated with the interconnection metals. When text layers are not specified (and “text fall through” is not selected), Milkyway assumes they match the geometry layers.

When Text Is on a Different Layer Than Its Geometry

When each geometry layer has one or more text layers associated with it, you need to supply the following text-to-metal information in the Extract Blockage dialog box:

- Poly Text is the layers used for the text identifying poly geometries.
- Metal# Text is the layers used for the text identifying metal# geometries.

Type the names or numbers of the layers used for the text that annotates geometries on each interconnection metal (or leave the box blank if the text layer matches the geometry layer). To specify multiple layers, separate the names or numbers with commas.

Note:
Milkyway always extracts any text that is on the same layer as the geometry beneath it.

When All Text Is on the Same Layer

When all text labels are on the same layer, you need to fill out the Extract Blockage dialog box as follows:

- Selecting “text fall through” instructs Milkyway to look at the layers beneath the origin of each text label on the Fall Through Text layer to identify a geometry annotated by the text. If more than one geometry exists beneath the origin of the text label, Milkyway will select the geometry to associate with the label according to the following priorities, from highest to lowest:
  - metal3
  - metal2
  - metal1
  - poly

  For example, if one of the geometries is metal3, Milkyway selects this geometry. If not, Milkyway looks for metal2, then metal1, then poly.
- Selecting “double fall through” instructs Milkyway to look at the layers beneath the origin of each text label on a metal2 geometry to identify a geometry (in addition to the metal2 geometry) annotated by the text. If more than one geometry (besides the metal2 geometry) exists beneath the origin of the text label, Milkyway will select the second geometry to associate with the label, according to the following priorities, from highest to lowest:
  - metal1
  - poly

  For example, if one of the geometries is metal1, Milkyway selects this geometry. If not, Milkyway looks for poly.

Type the name or the number of the Fall Through Text layer, the layer used for text labels. Milkyway reads each text label on the Fall Through Text layer to identify the pins represented by the geometry (or two geometries) beneath that text label.

Creating Blockage Areas

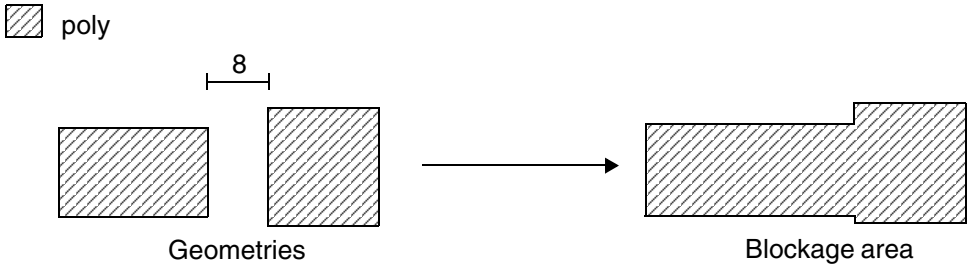
Milkyway creates blockage areas by merging any geometries on the same layer that are within “merging distance” of each other (except pins). By default, the merging distance equals

$$(2 \times minSpacing) + minWidth$$

where *minSpacing* and *minWidth* are the minimum spacing and minimum width, respectively, specified in the technology file for the layer.

For example, if the technology file specifies a poly *minSpacing* of 2 and a poly *minWidth* of 4, the merging distance equals 8. (The technology file also defines unit values. For more information, see Appendix A, "Technology File.") Therefore, Milkyway merges the geometries on the left to create the blockage area on the right, as shown in Figure 7-1.

Figure 7-1 Merging Same-Layer Geometries



Using the Threshold options in the Extract Blockage dialog box, you can increase or decrease merging for geometries on the following layers:

- poly
- metal1
- metal2
- metal3
- metal4
- metal5
- metal6
- polyCont
- via1

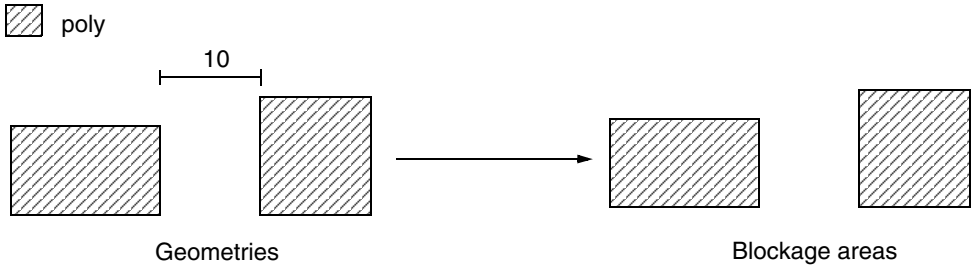
Typing a threshold value for one of these layers changes the merging distance to the following:

$$(2 \times minSpacing) + threshold$$

where *minSpacing* is the minimum spacing specified in the technology file for the layer and *threshold* is the threshold you specify in the Extract Blockage dialog box.

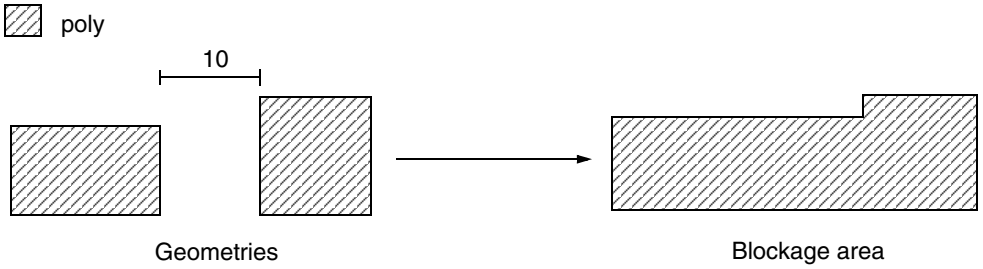
For example, if the technology file specifies a poly *minSpacing* of 2 and a poly *minWidth* of 4 and the poly threshold is 0, the merging distance equals 4. Therefore, no merging occurs, as shown in Figure 7-2.

Figure 7-2 No Merging Occurs



However, if you specify a poly threshold of 6, the merging distance increases to 10. Therefore, Milkyway merges the geometries on the left to create the blockage area on the right, as shown in Figure 7-3.

Figure 7-3 Using a Threshold Value to Vary the Merging Distance



Note:

You can speed up processing of macro cells that contain many small blockage areas by deleting blockage geometry prior to creating the pin/blockage cells.

Creating Access to Metal1 Pins Inside Blockage Areas

If cells have metal1 pins inside metal1 blockage, you need to select the “pin on blockage” in the Extract Blockage dialog box.

- If you do not select this option, the router cannot access the metal1 pins.
- If you select this option, Milkyway will cut through the metal1 blockage where there are metal1 pins to allow the router to access the metal1 pins. In this case, the router can route in metal2 over the metal1 blockage and drop vias to access the metal1 pins.

Creating Cell Boundaries

If the cells do not have boundaries defined, Milkyway will create boundaries for them. If the cells have boundaries defined and you want to use these existing boundaries, you need to deselect the Generate Boundary options in the Extract Blockage dialog box.

Horizontal Via Placement

To specify how you want to place standard cells to achieve optimum horizontal via placement, use the Verti Via Grid Offset option.

Verti Via Grid Offset

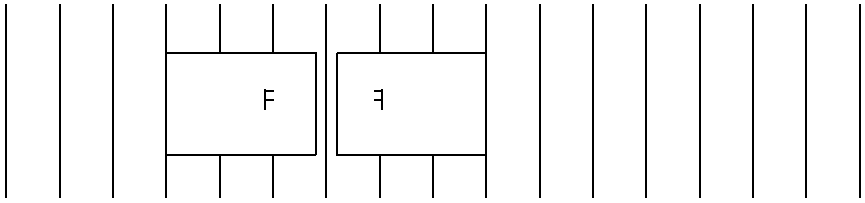
Indicate how you want Astro to place standard cells to achieve optimum horizontal via placement.

Option	Instructs Astro to place cells so that
`auto`	The maximum number of vias are on a vertical wire track
`specify`	The left side of an unflipped cell (or the right side of a flipped cell) is at a distance equal to the value you type from a vertical wire track

The specify option works as follows:

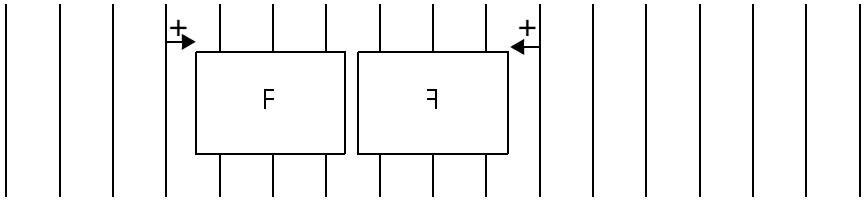
To place an unflipped cell so that its left side is on a vertical wire track (or a flipped cell so that its right side is on a vertical wire track), as shown in Figure 7-4, select specify and type a zero (0).

Figure 7-4 Specifying Placement of Cell on Vertical Wire Track



To place an unflipped cell so that its left side is a specified distance from the right of a vertical wire track (or a flipped cell so that its right side is a specified distance from the left of a vertical wire track), as shown in Figure 7-5, select Specify and type a positive number (less than the pitch of the vertical routing layer) to indicate the distance.

Figure 7-5 Specifying Placement of Cell at Specified Distance From Vertical Wire Track



Pin Transfer

Milkyway can convert a pin to its adjacent lower metal layer. This ability provides more flexibility in routing to the pins.

The `auExtractBlockagePinVia` command can transfer the maximum number of pins according to the specified layer(s) without changing the internal connectivity of the cell—for example, metal2 to metal1, metal3 to metal2, and so on.

After you have performed any required hierarchical layout cell expansion, you can use the transfer pin on layer option of the `auExtractBlockagePinVia` command to process the pins. (For more information about `auExtractBlockagePinVia`, see Physical Implementation Online Help.)

Note:

The transfer pin on layer option works only when the `auExtractBlockagePinVia` Extract Connectivity option is selected.

Specifying Port Information for Advanced Operations

Before you created the pin/blockage cells, you identified power and ground ports by using `dbSetCellPortTypes` functions. (For more information about `dbSetCellPortTypes`, see the *Milkyway Environment Extension Language Reference Manual*.) If you plan to perform advanced operations, you need to specify additional port information after you create the pin/blockage cells. This information includes the following:

- Whether a port is input, output, inout, or three-state
- Whether a port is a clock net port
- Whether any unconnected ports should be tied to power or ground

Note:

If you change the tie-up/tie-down information after initial floorplanning (which binds the netlist and the layout), you must update connections in the cells, using Design Setup: Setup > P/G Connect, as described in "Specifying Global Net Connections" on page 9-2.

- Whether a port that is asynchronous with respect to clock is active high or active low
- Whether a port is the input to or output from a clocked cell
- Whether a port is the address input of a memory cell (RAM or ROM)
- Whether a port is the enable for the scan clock
- Whether a port is a scan input, scan output, or scan clock input for a scannable cell
- Whether a port is a control line for a three-state cell

Advanced operations requiring additional port information include the following:

- Automatic tie-up/tie-down (For more information, see "Specifying Global Net Connections" on page 9-2.)
- Timing-driven layout
- Clock tree synthesis
- Scan chain optimization
- ECO

This section provides a cross-reference table to show the additional information required for each of these operations. It also explains how to

- Create a port information file
- Load a port information file
- Write current port information to a file

Requirements for Advanced Operations

Table 7-1 cross-references the advanced operations with the additional information required.

Table 7-1 Information Required for Advanced Operations

Advanced information	Auto tie-up/tie-down	Timing-driven layout	Clock tree synthesis	Clock distribution	Scan chain optimization	ECO
Whether a port is input, output, inout, or three-state		Y	Y	Y	Y	Y
Whether a port is a clock net port		Y	Y			
Whether any unconnected ports should be tied to power or ground	Y					Y
Whether a port is asynchronous with respect to clock is active high or active low		Y				
Whether a port is the input to or output from a clocked cell		Y				
Whether a port is the address input of a memory cell (RAM or ROM)		Y				
Whether a port is the enable for the scan clock		Y				
Whether a port is a scan input, scan output, or scan clock input for a scannable cell		Y			Scan trace only	

Table 7-1 Information Required for Advanced Operations (Continued)

Advanced information	Auto tie-up/tie-down	Timing-driven layout	Clock tree synthesis	Clock distribution	Scan chain optimization	ECO
Whether a port is a control line for a three-state cell		Y				

Creating a Port Information File Manually

Although you can type `dbSetCellPortTypes` functions in the input area of the Milkyway window, it is probably easier to do the following:

- Create a port information file containing a `dbSetCellPortTypes` function for each cell in the library. You can create the file with any UNIX text editor.
- Load the file by using the load function, as described later.

(For more information, see the *Milkyway Environment Extension Language Reference Manual*.)

Alternatively, you can include `dbSetCellPortTypes` functions in the CLF file, which you load immediately after you create the library. (For more information, see "Creating and Loading a CLF File" on page C-3.)

For syntax information, see the *Milkyway Environment Extension Language Reference Manual*.

Note:

If you are performing timing-driven layout with data from Synopsys, you can create a CLF file containing the information required for timing-driven layout, by translating the Synopsys timing file, as described in the *Static Timing Analyzer User Guide*.

Loading Port Information

To load a port information file, open the library and type the following in the input area of the Synopsys application window:

```
load fileName
```

where *`fileName`* is the name of the port information file.

For example,

```
load "port.types"
```

loads the information specified in the port.types file.

Writing Port Information to a File

To write current port information to a file, type the following in the input area of the Milkyway window:

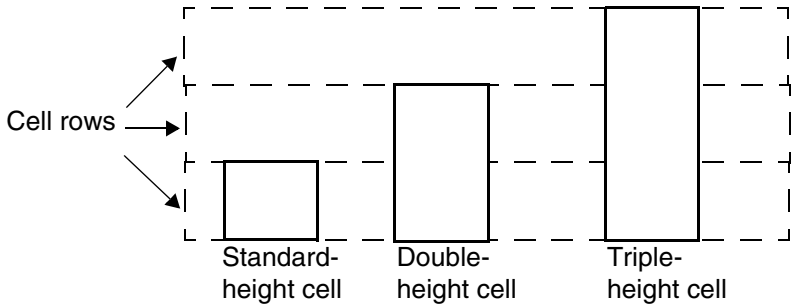
```
dbDumpGPortTable libName fileName
```

where *libName* is the name of the library for which you want port information and *fileName* is the name of the file to which you want to write the information.

Using Multiple-Height Cells

Astro lets you use double- and triple-height cells with the standard cells in your design, as shown in Figure 7-6. These multiple-height cells use the same unit tile as the standard cells and are placed over multiple cell rows.

Figure 7-6 Multiple-Height Cell Examples



If you are using multiple-height cells, you use the same design flow as with standard cells, except that you do the following steps in Milkyway:

1. Specify whether a cell is double- or multiple-height by using `cmMarkCellType`.

   Note:

   You do not have to specify the cell type option at this point. Astro can figure out whether a cell is double- or triple-height when generating the place and route boundary.

2. Generate the place and route boundary for the cells, using `auSetPRBdry` with the Multiple (2x, 3x) option. For information about setting place and route boundaries, see "Setting the Place and Route Boundary" on page 7-17.

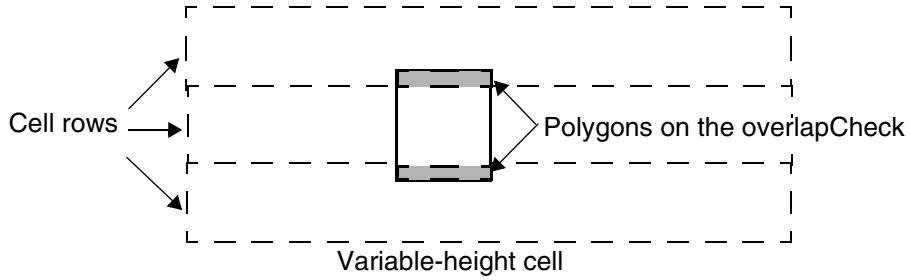
3. Use `cmSetMultiHeightProperty` to adjust the place and route boundary.

   For more information about the `cmMarkCellType` and `cmSetMultiHeightProperty` commands, see Physical Implementation Online Help.

Using Variable-Height Cells

Astro lets you use variable-height cells that are 1.1 and 1.2 times as large as the standard cells in your design, as shown in Figure 7-7. These variable-height cells use the same unit tile as the standard cells and overlap the cell rows.

Figure 7-7 Variable-Height Cells Example



If you are using variable-height cells, you will use the same design flow as with standard cells, except that in Milkyway you select the Create Overlap-Check polygon option when generating the place and route boundaries for the cells with `auSetPRBdry`. For each cell with a cell boundary that is bigger than the place and route boundary, Milkyway creates a polygon on the overlapCheck layer (layer 192). Astro uses these polygons on the overlapCheck layer to place cells so that they do not overlap. For information about setting the place and route boundaries, see "Setting the Place and Route Boundary," which follows.

Unlike with double- or triple-height cells, you do not specify the cell type by using `cmMarkCellType` with variable-height cells.

Note:

If you have variable-height cells in your design, do not use the Double Back option when setting up the design with `axgPlanner`. (For more information, see Physical Implementation Online Help.)

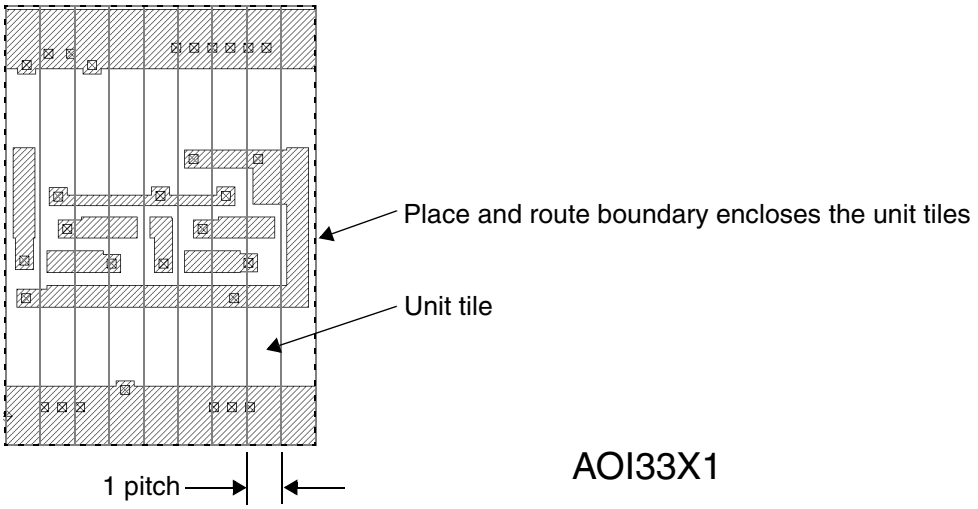
Using Multiple Cell Classes

Cell classes enable you to place different types of cells in particular areas of the design. Astro lets you separate cells into different classes by creating multiple tiles and assigning each cell to a particular tile.

Setting the Place and Route Boundary

Astro uses the place and route boundary to align wire tracks and power and ground rails during cell placement. The place and route boundary is the bounding box of the unit tiles used by a standard cell, as shown in Figure 7-8.

Figure 7-8 Place and Route Boundary Example



Note:

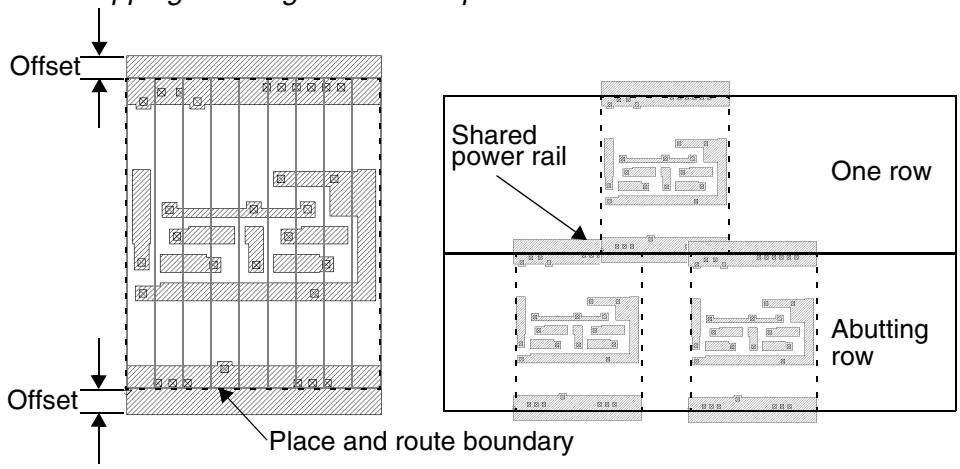
Jupiter uses the place and route boundaries to estimate the area required for a group of cells.

You generate place and route boundaries for cells and create the unit tile in your library by using `auSetPRBdry`. Astro observes the following specifications when creating the place and route boundaries and unit tile for each library:

- The width of the unit tile is equal to the metal2 pitch specified in the technology file.
- The place and route boundary width is an integral multiple of the unit tile width.
- For single-height cells, all standard-cell place and route boundaries have the same height, which is the height of the unit tile.
- For multiple-height cells, the place and route boundaries for standard cells have various heights, depending on the Multiple Height option you select.

If the cells are designed to overlap, make the place and route boundary smaller. For example, to make the power and ground pins of abutting rows overlap in a back-to-back floorplan, decrease the height of the place and route boundary so the cells abut along the center of the power and ground pins, as shown in Figure 7-9.

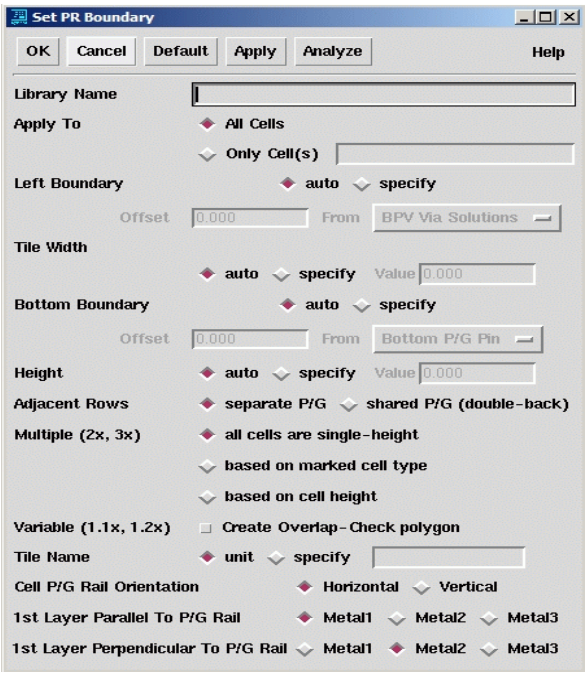
Figure 7-9 Overlapping Abutting Rows Example



To set the place and route boundary,

1. Enter `auSetPRBdry` or choose Cell Library > Set PR Boundary.

   The Set PR Boundary dialog box appears.



2. Enter the required information in the Set PR Boundary dialog box. For more information about the dialog box options, see Physical Implementation Online Help.

   Note:

   You can select Analyze in the Set PR Boundary dialog box to check whether a cell is double- or triple-height.

3. Click OK.

The `auSetPRBdry` command creates the place and route boundaries for the specified cells and creates a unit tile.

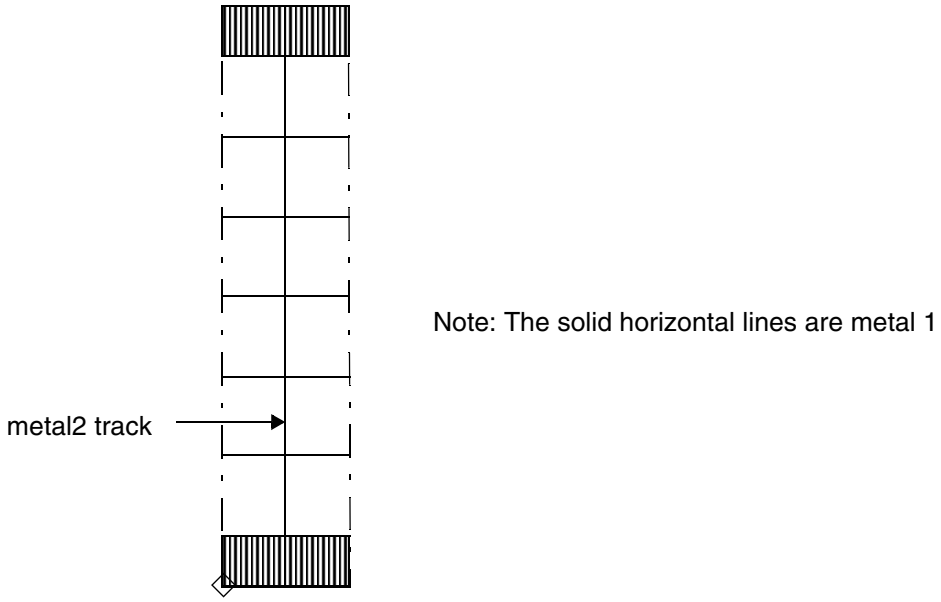
If you are using multiple-height cells in your design, reset the place and route boundary for these cells by using the `cmSetMultiHeightProperty` command. For more information about using multiple-height cells, see "Using Multiple-Height Cells" on page 7-15.

Defining Wire Tracks

After you adjust the cell boundary and create the unit tiles by using `auSetPRBdry`, you need to create wire tracks in the unit tile. The unit tile cell must contain wire tracks, which assist the router in placing wires for each of the routing layers.

Each layer has its own set of wire tracks, which run in the preferred direction for that layer. For example, wire tracks on metal1 run along the horizontal axis. Figure 7-10 shows an example of a unit tile with the wire tracks defined.

Figure 7-10 Unit Tile With Wire Tracks Defined

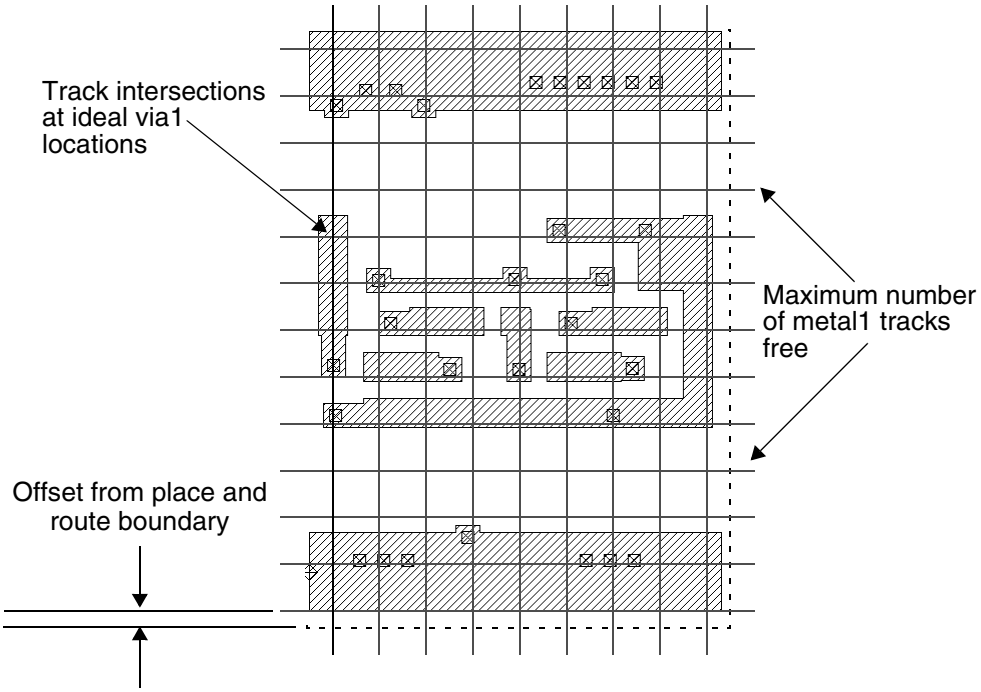


Use the following guidelines to ensure that the wire tracks are created in a way that maximizes routing:

- The width of the unit tile should be the metal2 pitch.
- The distance from the bottom of the unit tile to the first wire track must be 0 or one half the metal1 pitch. Otherwise, when you use the Double Back option in `axgPlanner`, Astro leaves space between the cell rows.

  This space depends on the size of the space between the top track and the top of the unit tile or the bottom track and the bottom of the unit tile and is subject to the row spacing rule (using `rowSpacing`) in the technology file.
- If there are metal1 routing areas inside the cell, try to maximize the number of metal1 tracks available for routing.
- Via1 locations should intersect as many tracks as possible.
- Generally, the wire track on metal2 should be centered vertically in the unit tile cell, although there are libraries in which the metal2 wire track is aligned with the left boundary (see Figure 7-11).

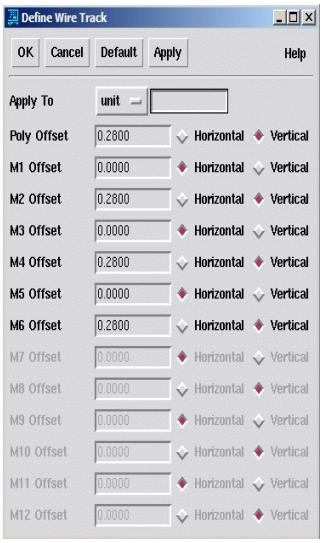
Figure 7-11 Standard Cell Placed on Array of Unit Tiles



To define wire tracks,

1. Enter `axgDefineWireTracks` or choose Wire Tracks > Define Unit Tile Wire Tracks.

   The Define Wire Track dialog box appears.



2. Enter the required information in the Define Wire Track dialog box. For more information about the dialog box options, see `axgDefineWireTracks` in Physical Implementation Online Help.
3. Click OK.

8

Translating Logical Design Data

If you have completed translation and processing of the physical design data, you have one or more libraries containing the following:

- Layout cells (CEL views)
- Pin/blockage cells (FRAM views used by Astro)

Now you are ready to translate the logical design data. This data can be in EDIF or Verilog HDL.

When you import a netlist in EDIF or HDL format, the result is a netlist cell, which represents the logical connectivity of a design at the top level. This connectivity information is then bound to the top-level layout cell during floorplanning.

Translating EDIF, HDL, or VHDL Data

This section explains how to import data in EDIF, HDL, or VHDL format. The basic steps are as follows:

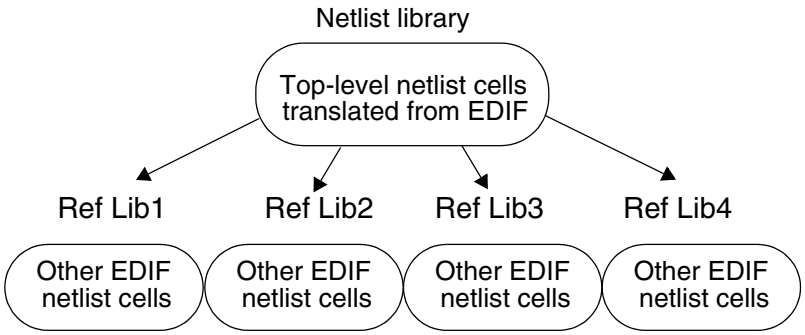
- Specify the library in which you want to place the netlist cell (EDIF, Verilog, or VHDL).
- Specify any cell name mapping you want to use.
- Translate the data, creating one or more netlist cells.
- Expand the netlist cell(s), if necessary.

Specifying the Library for EDIF Files

The translation process creates libraries containing the netlist cells. Generally, the library structure is as shown in Figure 8-1. The arrows indicate that the netlist library references the libraries containing other netlist cells.

Note:
Reference libraries are specified by “external” in the netlist.

Figure 8-1 Netlist Library Structure



The EDIF file specifies the names of these libraries. You can change the name of a netlist library as specified in the file by using any UNIX text editor. Find either of the following lines in the EDIF file, and change it to reflect the library name you want:

```
(library libraryName (EDIFLevel 0) ...
(library (rename "libraryName" "DESIGNS")(EDIFLevel 0) ...
```

where *libraryName* is the name of a library and DESIGNS is a library name example. Replace this library name with the name you want for the netlist library.

Note:

If you want to place the netlist cell in a separate library, you do not need to edit the EDIF file or create the netlist library unless you want to specify a name different from that specified in the netlist.

The translation process creates the library specified by *libraryName* if that library does not exist. However, you need to be sure that your layout library references the netlist library before you perform floorplanning. For more information, see "Referencing Cells in Another Library" on page 2-6.

Specifying Cell Name Mapping

By default, Milkyway translates a netlist by using the cell names given in the EDIF or HDL file. If the cell names in your netlist do not match the cell names in your layout (GDSII Stream file), you can specify mapping for those cells in an EDIF-to-GDSII or HDL-to-GDSII map file. Each line in the file specifies a particular EDIF (or HDL) cell name that you want translated to a particular GDSII Stream cell name.

Syntax

The syntax for a line in the EDIF (or HDL)-to-GDSII map file is

edifOrHDLCellName gdsIICellName

where the variables are as follows:

edifOrHDLCellName	Name of the cell in the netlist Valid values: Name of any cell referenced in the netlist file
gdsIICellName	Name of the cell in the GDSII Stream layout Valid values: Name of any cell referenced in the GDSII Stream file

Note:

If you omit mapping information for any cells, Milkyway will translate those cells with the cell names given in the netlist.

Example

```
ANDedif ANDgds
```

This line tells Milkyway to translate any cell named `ANDedif` to a cell named `ANDgds`.

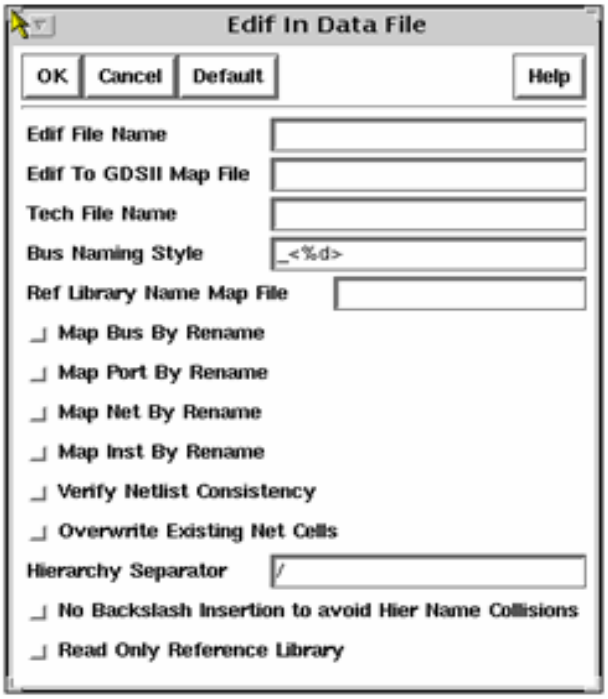
Translating EDIF Files

After you specify your library in the EDIF file, the following procedure creates a Synopsys netlist cell in the library you specified. (If the library does not exist, the procedure creates it.)

To translate the EDIF file,

1. Enter `auEdifIn` or choose Netlist In>Edif In.

   The Edif In Data File dialog box appears.



2. Fill in the Edif In Data File dialog box. For detailed descriptions of the command options, see Physical Implementation Online Help.
3. Click OK.

   The `auEdifIn` command translates the EDIF file and places the resulting netlist cell in the library specified in the EDIF file. If this library does not exist, Milkyway creates it.

   To find the names of the newly created netlist library and netlist cell, use Milkyway. You can use the Qualify feature to display only the netlist and expanded netlist cells sorted by date. For instructions, see the *Physical Implementation Getting Started*.

Translating Verilog (HDL) Files

If you want to translate multiple Verilog files, you need to create a Verilog list file that contains the names of the files you want to translate. If the files are not in the directory where you started Milkyway, you need to include the full path. The format of the file is as follows:

fileName_1 fileName_2 ... fileName_n

where *fileName_1*, *fileName_2*, and *fileName_n* are the names (including the paths, if necessary) of the files.

The following procedure creates a Synopsys netlist cell in the specified library. (If the library does not exist, the procedure creates it.)

To translate the Verilog file,

1. Enter `auVerilogToCell` or choose Netlist In > Verilog To Cell.

   The Verilog In Data File dialog box appears.

2. Complete the fields in the dialog box. For detailed descriptions of command options, see Physical Implementation Online Help.

3. Click OK.

Translating VHDL Files

If you want to translate multiple VHDL files, create a VHDL list file that contains the names of the files you want to translate. If the files are not in the directory where you started Milkyway, include the full path. The format of the file is

fileName_1 fileName_2 ... fileName_n

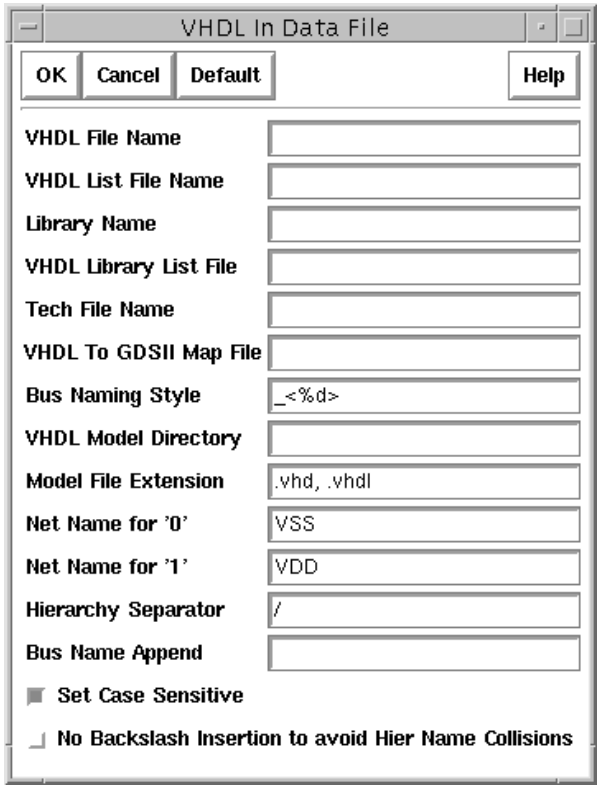
where *fileName_1*, *fileName_2*, and *fileName_n* are the names (including the paths, if necessary) of the files.

If you want to specify a scheme for mapping VHDL cell names to GDSII cell names, create a VHDL To GDSII map file. For information about creating a VHDL To GDSII map file, see the *Physical Implementation Getting Started*.

To translate the VHDL file,

1. Enter `auVhdlIn` or choose Netlist > InVHDL In.

   The VHDL In Data File dialog box appears.



2. Fill in the Verilog In Data File dialog box. For detailed descriptions of command options, see Physical Implementation Online Help.
3. Click OK.

Specifying Bus and Scalar Name Mapping for Mentor Data

During the EDIF translation, Milkyway translates buses and scalars to individual nets and ports. You can specify one of two methods for naming the elements of a bus or scalar when the EDIF file defines buses and scalars in the format generated by the Mentor system.

Buses

The Mentor system generates EDIF files that define buses in the format

```
port (array (rename edifName "schName") #InArray)
```

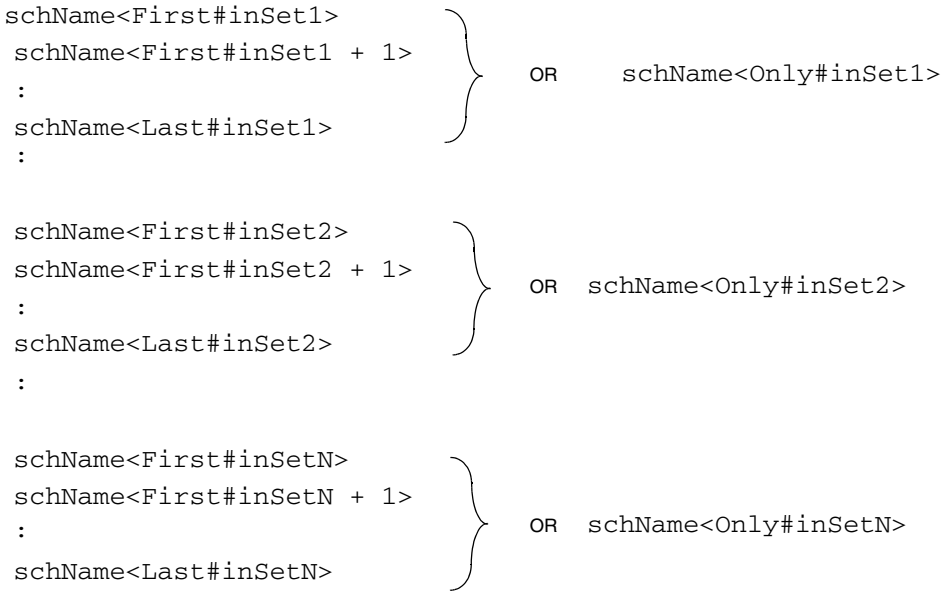
where the variables are as follows:

edifName	EDIF name for the bus.

schName	User-specified name for the bus (usually the name used by the schematic editor), in the format *busName*(*RangeOfNumbersInSet1*[,*RangeOfNumbersInSet2, ... RangeOfNumbersInSetN*]) where *busName* is the name of the bus and *RangeOfNumbersInSetX* specifies the range of element numbers in each set, in one of the following formats: • If the set contains multiple elements: `First#inSet:Last#inSet` • If the set contains one element: `Only#inSet` Note: The square brackets ([ ]) indicate that multiple sets are optional.
#InArray	Number of elements in the bus.

During the translation, you can specify one of the following naming methods:

- If you select Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the elements as follows:



- If you deselect Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the elements as follows:

```
editName<0>
edifName<1>
:
edifName<#InArray - 1>
```

For example, an EDIF file defines a bus as follows:

```
port (array (rename ABC_40_0_58_3_41_ "XYZ(3:6,9:10,12)") 7)
```

During the translation, you can specify one of the following methods for naming the elements of the bus:

- If you select Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the elements as follows:

```
XYZ<3>
XYZ<4>
XYZ<5>
XYZ<6>
XYZ<9>
XYZ<10>
XYZ<12>
```

- If you deselect Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the elements as follows:

```
ABC_40_0_58_3_41_<0>
ABC_40_0_58_3_41_<1>
ABC_40_0_58_3_41_<2>
ABC_40_0_58_3_41_<3>
ABC_40_0_58_3_41_<4>
ABC_40_0_58_3_41_<5>
ABC_40_0_58_3_41_<6>
```

Scalars

The Mentor system generates EDIF files that define scalars in the format

```
port (rename edifName "schName")
```

where the variables are as follows:

edifName	EDIF name for the scalar

schName	User-specified name for the scalar (usually the name used by the schematic editor), in the format *scalarName (Element#)* where *scalarName* is the name of the scalar and *Element#* is the number of the element

During the translation, you can specify one of the following naming methods:

- If you select Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the scalar element as follows:

  `schName<Element#>`

- If you deselect Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the scalar element as follows:

  `edifName`

For example, an EDIF file defines a scalar as follows:

```
port (rename ABC_40_0_58_3_41_ "XYZ(3)")
```

During the translation, you can specify one of the following methods for naming the scalar element:

- If you select Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the scalar element as follows:

  `XYZ<3>`

- If you deselect Map Bus By Rename in the Edif In Data File dialog box (and use the default Bus Naming Style), the translation process will name the scalar element as follows:

  `ABC_40_0_58_3_41_`

Specifying Bus Name Mapping for Synopsys Data

The Synopsys system generates EDIF files that define buses in the format

```
port (array (rename edifName "schName") #InArray)
```

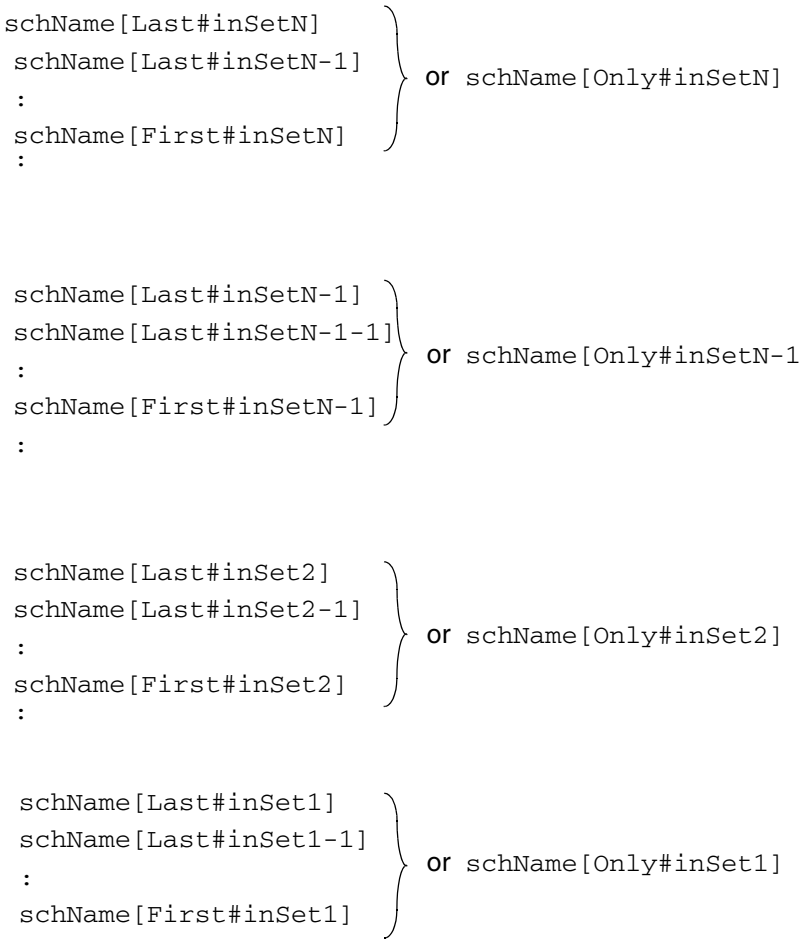
where the variables are as follows:

edifName	EDIF name for the bus

schName	User-specified name for the bus (usually the name used by the schematic editor), in the format *busName[RangeOfNumbersInSetN[,* *RangeOfNumbersInSetN-1, ...* *RangeOfNumbersInSet1]]* where *busName* is the name of the bus and *RangeOfNumbersInSetX* specifies the range of element numbers in each set, in one of the following formats: • If the set contains multiple elements: `Last#inSet:First#inSet` • If the set contains one element: `Only#inSet` Note: The inside set of square brackets ([ ]) indicates that multiple sets are optional. (The outside set of square brackets is part of the schematic name.)
`#InArray`	Number of elements in the bus

During the translation, you can specify one of the following naming methods:

- If you select Map Bus By Rename in the Edif In Data File dialog box (and specify a Bus Naming Style of _[%d]), the translation process will name the elements as follows:



- If you deselect Map Bus By Rename in the Edif In Data File dialog box (and specify a Bus Naming Style of _[%d]), the translation process will name the elements as follows:

```
edifName[#InArray - 1]
edifName[#InArray - 2]
:
edifName[1]
edifName[0]
```

For example, an EDIF file defines a bus as follows:

```
port (array (rename ABC_7_0_ "XYZ(12,10:9,6:3)") 7)
```

During the translation, you can specify one of the following methods for naming the elements of the bus:

- If you select Map Bus By Rename in the Edif In Data File dialog box (and specify a Bus Naming Style of _[%d]), the translation process will name the elements as follows:

```
XYZ[12]
XYZ[10]
XYZ[9]
XYZ[6]
XYZ[5]
XYZ[4]
XYZ[3]
```

- If you deselect Map Bus By Rename in the Edif In Data File dialog box (and specify a Bus Naming Style of _[%d]), the translation process will name the elements of the bus as follows:

```
ABC_3_0_[6]
ABC_3_0_[5]
ABC_3_0_[4]
ABC_3_0_[3]
ABC_3_0_[2]
ABC_3_0_[1]
ABC_3_0_[0]
```

Expanding a Netlist Cell

If you import a netlist from EDIF or HDL, you bind the netlist and the layout. To perform this operation, every object referenced in the netlist must exist in the layout. In some cases, objects in the netlist do not exist in the layout; however, components of these objects do. In such cases, you need to expand the netlist recursively until all objects in the netlist are broken down into component objects that exist in the layout.

During netlist expansion, you can also specify top-level global net connections to nets and ports in the subcells.

To perform the expansion operation, Milkyway must be able to compare the netlist with the layout.

For more information about specifying reference libraries, see “Referencing Cells in Another Library” on page 2-6.

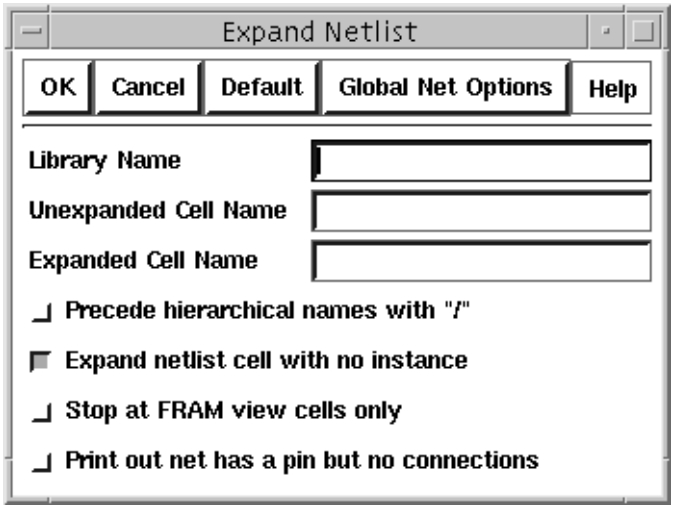
Note:

If you are going to be using Jupiter following data preparation, you might not want to perform an expansion. Rather, you might want to use the flatten and grouping functions of Jupiter.

To expand the netlist and specify global net connections in subcells,

1. Enter `cmCmdExpand` or choose Netlist > Expand.

   The Expand Netlist dialog box appears.



2. Complete the Expand Netlist dialog box and the Global Net Options Expand Netlist subform. For detailed descriptions of command options, see Physical Implementation Online Help.
3. Click OK in the Expand Netlist dialog box.

   The `cmCmdExpand` command expands the netlist cell, connects the ports and nets in the subcells to top-level global nets, and places the resulting expanded netlist cell in the netlist library.

9

Initializing the Top-Level Design Cell

Now that you have the data translated and processed, you are ready to prepare a top-level cell for floorplanning. Depending on the format of your input data, this process consists of one or more of the steps covered in the following sections:

- Binding the Layout and the Netlist
- Creating Cell Instances Not in the Netlist
- Specifying Global Net Connections

Before you begin, you need to be sure your libraries are set up correctly. For information about the recommended library structure, see “Organizing Libraries” on page 2-2.

Note:

You perform the operations in this chapter with your tool based on Milkyway. For information about starting the tool, see the *Physical Implementation Getting Started*.

Binding the Layout and the Netlist

If you translated your logical data from EDIF, you need to bind the layout and the netlist, using the Bind Netlist command. For information about the Bind Netlist command in Astro, see Physical Implementation Online Help.

In Jupiter, the `fphBindHierPlan` command binds the hierarchical netlist to a specified top cell.

Creating Cell Instances Not in the Netlist

If your design requires cell instances not referenced by the netlist, such as power and ground pads and corner pads, you can create them with `dbCreateCellEQClass`. For more information, see the *Milkyway Environment Extension Language Reference Manual*.

Specifying Global Net Connections

You specify port-to-net connections for power and ground nets in the open cell by using the appropriate command from the following table.

Application	Scheme command	Tool: menu command
Astro	`aprPGConnect`	Preroute>Connect Ports to P/G

You specify port-to-net connections for power and ground nets in the open cell by using `aprPGConnect`. For more information, see Physical Implementation Online Help.

If your netlist specifies power/ground connections, do not specify them with this command.

Note:

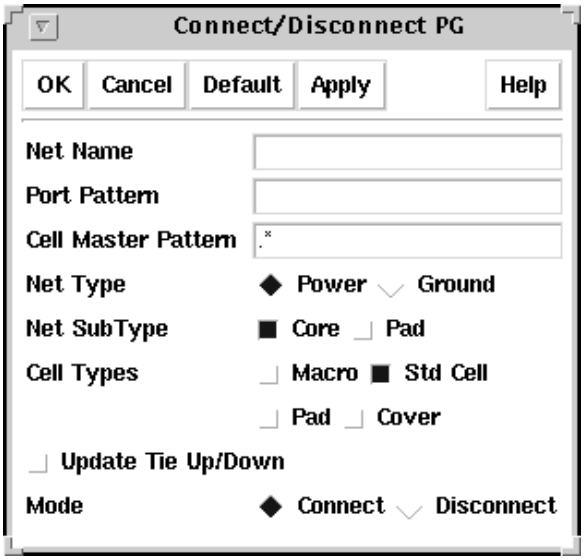
If you modify tie-up/tie-down specifications after binding the netlist and layout, you use Setup > P/G Connect to update connections in the cells.

In this case, leave the Net Name and Port Pattern blank and execute the command with the Update Tie Up/Down option selected. For information about modifying tie-up/tie-down specifications, see “Specifying Port Information for Advanced Operations” on page 7-11.

To specify connections for power and ground nets,

1. Enter `aprPGConnect` or choose Preroute > Connect Points to P/G.

The Connect/Disconnect PG dialog box appears.



Note:

Net SubType does not appear in the dialog box in Astro. An additional Cell Type, Cover, appears in Astro.

2. Fill in the Connect/Disconnect PG dialog box. For detailed descriptions of command options, see Physical Implementation Online Help.
3. Click OK.

10

Library Checking

Advanced IC design relies on high-quality libraries. Ensuring correct and consistent library data is an essential task before design creation. Milkyway provides checking capability for logical and physical libraries to facilitate successful design and verification.

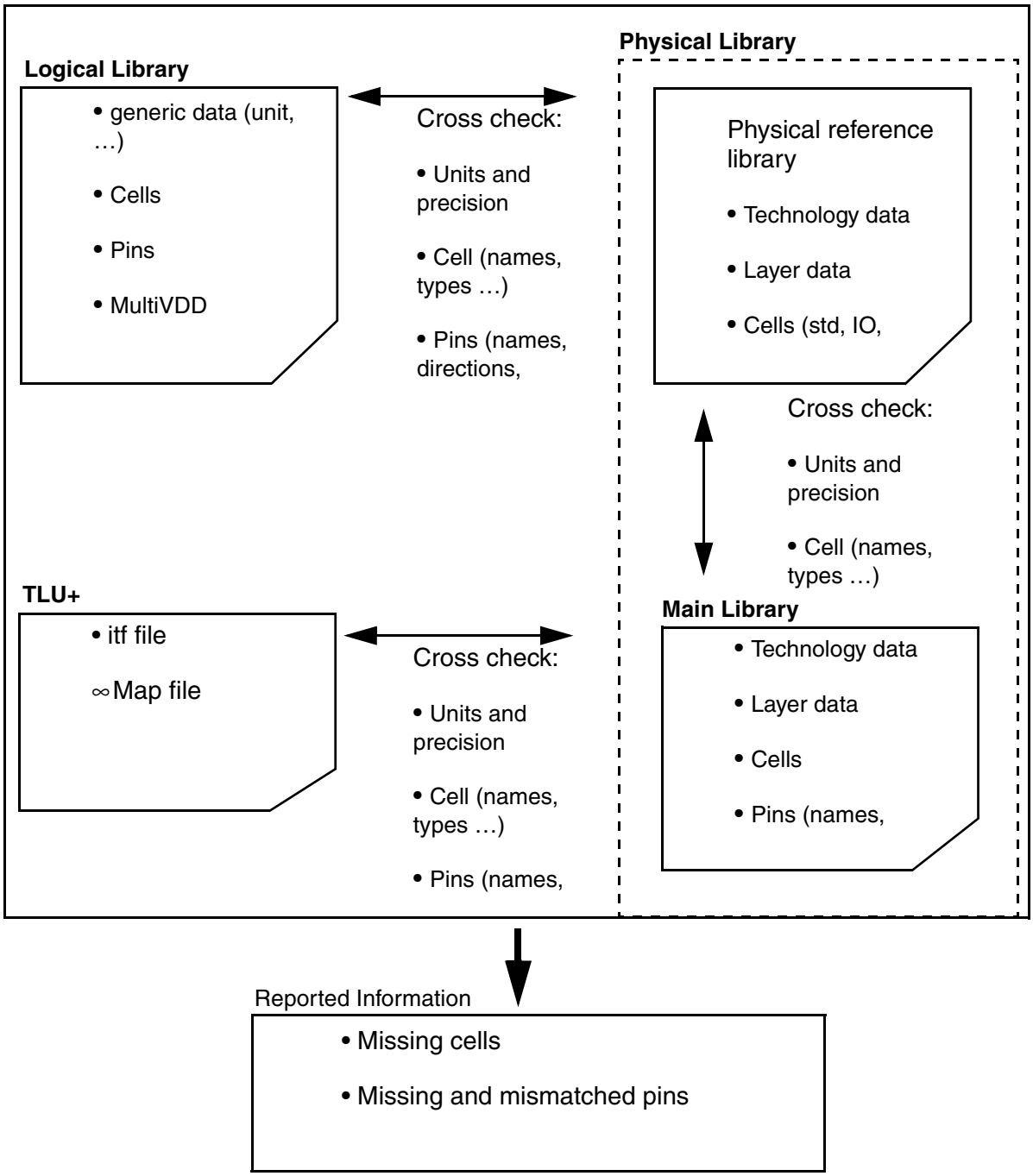
Note:
Although Milkyway provides consistency checks, you need to ensure consistency between your physical and timing libraries.

This chapter has the following sections:

Library Checking Overview

Milkyway is the common database used in the Synopsys Galaxy platform to store libraries and design data. It provides checks within a library and between libraries. Milkyway performs library checking during and after library creation. A diagram of some of the Milkyway library checks is shown in Figure 10-1.

Figure 10-1 Overview of Library Checks



You perform library checking with the `check_library` Tcl command. The setup and options of this command are described in the following section.

Preserving Power and Ground Pin Connections

If a pin is defined as a power pin in the Milkyway reference library, but not in the logic library (or vice versa), IC Compiler issues the MWDC-139 or MWDC-143 error message. It disconnects the power net from the pin and connects the pin to logic zero.

To check the pin type in the Milkyway reference library, output the port table from the Milkyway reference library by using the dbDumpGPortTable command.

To prevent IC Compiler from disconnecting power nets, you must ensure that the power pins are properly defined in both the Milkyway reference library and the logic library. To accomplish this, use one of the following methods:

- Remove the power pins that are wrongly defined as signal pins from the logic library and remove the connections to these pins from the design netlist.
- Update the pin definitions in the logic library by using the following syntax:

```
pg_pin (pg_pin_name) {
voltage_name : voltage_name;
pg_type : type_value
}
```

For more information about the pg_pin syntax, see *Chapter 2, "Advanced Low Power Modeling*," of the *Library Compiler User Guide: Modeling, Timing, Signal Integrity, and Power in Technology Libraries*.

Note:

Do not change the pin type in the Milkyway reference library from power or ground to signal to match the pin type in the logic library. Doing so can cause problems in power and ground routing and rail analysis.

check_library Command Setup and Options

This section covers the `check_library, set_check_library_physical_options`, and `report_check_library_options` Tcl commands.

Note that with the Z-2007.03 release, only physical library checking is available using the `check_library` command.

Before you run `check_library`, you need to set up the checking options by using the `set_check_library_physical_options` command as follows:

```
set_check_library_physical_options -option_name
```

If no options are specified with the command, `check_library` does not perform library checking.

You use the check_library command to check the data and quality for a specified library for, example,

```
check_library -mw_lib_name testing123.mw
```

You can review checking options by using the `report_check_library_options` command. Some of the options you can set are shown below. Consult the man pages for complete syntax and a current list of options. The following table provides an overview of available options.

The available library check options.

Check Item	Option
Routeability: physical pin access (pin on tracks)	`-routeability`
DRC checks for library cells (FRAM view) (from `cmCheckLibrary`)	`-drc`
Cell view vs. FRAM view in reference library with missing views and mismatched views (For example, earlier FRAM views) reported	`-view_cmp`
Missing antenna property for cells and antenna rules in the layers, and missing signal EM rule	`-antenna` `-signal_em`
Cells with identical names in different reference libraries with names of cells reported	`-same_name_cell`
Report boundaries for (macro) cells, rectilinear or rectangular, and coordinates	`-rectilinear_cell`
Internal property/attribute/flag (for example, preferred routing direction, tile pattern, place and route boundary, and wire track)	`-phys_property {place route}`
Report physical only cells (filler cells with and without metal, diode cells with antenna props, and corner cells)	`-physical_only_cell`
TF consistency check enhancement between main and reference libraries	`-tech_consistency`
Technology data quality for a single library (from `cmCheckLibrary`)	`-tech`

Library Check Options

`-routeability`

Checks for physical pin on-track accessibility and quality of defined wire tracks. It reports the total number of pins without on-track routability and lists pin names, directions, layers, and tracks for each cell of this issue.

In the track column, `H` denotes that the pin is not accessible on the Horizontal track, `V` denotes that the pin is not accessible on the Vertical track, and `H&V` denotes that the pin is not accessible on both `H` and `V`.

If a pin is reported as having poor accessibility, the pin might be routed by off-track wire during detail routing.

If a large number of pins is reported as having no on-track routability, you can adjust the offset values of the wire track (0 or half pitch preferred) and rerun `axgDefineWireTrack` or the `create_lib_track` command to reduce the number of pins with bad accessibility. For a report example, see "Report for -routeability Option" on page 10-10."

`-view_cmp`

Checks the CEL view against the FRAM view in the library and reports missing views and mismatched views. In the report table, columns CEL and FRAM list the cell name and cell version number. An `X` denotes that the cell is missing the view.

If a cell has a FRAM view that is earlier than its CEL view, it is marked as mismatched. No content in the cell (such as pins) is checked for mismatch. For a report example, see "Report for -view_cmp Option" on page 10-10.

`-antenna`

Checks for a missing antenna property for cells, and checks antenna rules in the layers. For an antenna property, list cell names and pin names. If an input pin is missing the gate size, it is counted as missing the property.

For an Output pin, if it is missing diode protection, it is counted as missing the property.

For a macro, if it is missing hierarchical antenna property, it is counted as missing the property. For antenna rules, mode, diode mode, default metal ratio, default cut ratio, and maximum ratio are reported. For a report example, see "Report for -antenna Option" on page 10-10.

`-signal_em`

Checks for signal electromigration rule (current model and model type) for each routing layer. For a report, example see "Report for -signal_em Option" on page 10-11.

`-same_name_cell`

Checks for cells with identical names among different reference libraries linked to the specified main (or design) library and the main library itself.

If there are cells with the same name in multiple reference libraries, the first one in the reference control file is used. For a report example see "Report for -same_name_cell Option."

`-rectilinear_cell`

Checks for cells with rectilinear boundaries including cell types and coordinates. For a report example see "Report for -rectilinear_cell Option."

`-phys_property {place route}`

Checks for placement and routing properties.

-phys_property {place} checks for placement properties for each standard cell, such as place and route boundary, cell height, unit tile, coordinate, and tile pattern, where cell height is relative to unit tile height, for example, 1xH for single height.

For macros, it reports cell boundary and height.

`-phys_property {route}` checks and reports for routing properties for each routing layer. For a report example, see “Report for -phys_property Option” on page 10-12.

`-physical_only_cell`

Checks for physical only cells (filler cells with and without metal, diode cells with antenna properties, and corner cells) with cell type and its property. For a report example see “Report for -physical_only_cell Option.”

`-tech_consistency`

Specify checks for technology data consistency between the main or design library specified in the check_library command and each reference library. Looks for problems such as missing layer data and mismatched technology data. For a report example, see “Report for -tech_consistency Option” on page 10-12.

`-tech`

Checks for the technology data for the specified library. This option differs from the `-tech_consistency` option, in that it checks the technology data for a single library. The checking messages are similar to those from library creation and technology data replacement.

This option requires write permission on the directory where the library resides.

`-drc`

Checks DRC violations for cells in the FRAM view for the specified library. Lists the cells with DRC violations in table format with cell name, cell type and error cell. For details of DRC violations, view the reported error cell information.

This option requires write permission on the directory where the library resides.

`-cells {list cell_1, cell_2, cell_3, ... cell_n}`

Specifies a list of cell names. If not specified, all cells in the library are checked. The `-cells` option can be specified with `-routeability`, `-antenna`, `-rectilinear_cell`, `-phys_property {place}`, and `view_cmp`. If it is specified with any other options, this option is ignored.

`-all`

Checks for all options described above.

`-reset`

Resets the options to default values.

The check_library Command Reporting Examples

After checking a library, Milkyway generates a report in table format. To facilitate reading, each option check report starts with a `#BEGIN_CHECK_ITEM` and concludes with an `#END_CHECK_ITEM` string. At the beginning and end of a `check_library` report, the library name and check date are reported as follows:

```
#BEGIN_CHECK_LIBRARY
   Main library name:/mylibs/my_best_cell
   Date and time:Thurs July 04 12:00:00 1776
#END_CHECK_LIBRARY
```

You can review checking options by using the `report_check_library_options` command, for example,

```
report_check_library_options -option_name
```

This command has the following options:

`-physical`

Reports physical library checking options set by the `set_check_library_physical_options` command.

`-logical`

Reports logical library checking options set by the `set_check_library_logical_options` command.

`-logical_vs_physical`

Reports logical versus physical library checking options set by the `set_check_library_cross_options` command.

`-default`

Reports default values of `check_library` options. If not specified, current values of the options are reported.

The following reports are provided as examples. For more detailed information on a given check option report, see the man pages.

Report for -routeability Option

```
#BEGIN_CHECK_ROUTEABILITY
   Total number of pins without on-track routeability: 1 (out of 1125)
             List of pins with bad routeability
  ------------------------------------------------------------
   Cell Name          Pin Name  Layer     Direction   Track
  ------------------------------------------------------------
   SDN_ADDF_1         CI        METAL2    Input       H&V
  ------------------------------------------------------------
#END_CHECK_ROUTEABILITY
```

Report for -view_cmp Option

```
#BEGIN_CHECK_VIEWCMP
Total number of cells missing CEL view:  0 (out of 997)
Total number of cells missing FRAM view: 1 (out of 997)
Total number of cells with mismatched view (CEL vs. FRAM):   0 (out of 997)
   X - cell missing view in the Table

            List of cells with missing views
  ---------------------------------------------
  Cell Name          CEL           FRAM
  ---------------------------------------------
  cell_1             cell_1:1       X
  ---------------------------------------------
#END_CHECK_VIEWCMP
```

Report for -antenna Option

```
#BEGIN_CHECK_ANTENNA
             List of antenna rules
  ------------------------------------------------------------------------
                                    Default     Default
  Layer Name    Mode   Diode mode   metal ratio  cut ratio   Max ratio
  ------------------------------------------------------------------------
  M1             1         3           500.00      20.00      500.00
  M2             1         3           500.00      20.00      500.00
  ------------------------------------------------------------------------
The library is missing antenna properties

 #END_CHECK_ANTENNA
```

Report for -signal_em Option

```
Warning: Signal EM rules are missing in the library.

#BEGIN_CHECK_SIGNALEM
             List of signal EM data
  ------------------------------------------------------------
     Layer name      Current model      Model type
  ------------------------------------------------------------
     METAL1              peak            table
     METAL1              rms             table
     METAL1              static          table
   ------------------------------------------------------------
  #END_CHECK_SIGNALEM
```

Report for -same_name_cell Option

```
Total number of cells with same names: 2(out of 193)
             List of cells with same names
  ---------------------------------------------------
        Cell name     library list
  ---------------------------------------------------
        MyXOR         mainlib ref1 ref2
        DFX           ref1 ref2 ref3
  ---------------------------------------------------
  #END_CHECK_SAMENAMECELL
```

Report for -rectilinear_cell Option

```
#BEGIN_CHECK_RECTILINEARCELL
   Total number of rectilinear cells:1 (out of 53)
            List of cells with rectilinear boundaries
-------------------------------------------------------------------------------
Cell Name  Cell type  Number of points                   Coordinate
-------------------------------------------------------------------------------
datapath    Macro           17        (  78.750,   0.000) (  78.750, 490.760)
                                      (  44.435, 490.760) (  44.435, 915.035)
                                      (  27.440, 915.035) (  27.440,1143.605)
                                      (   0.000,1143.605) (   0.000,1313.200)
                                      ( 677.295,1313.200) ( 677.295,1143.605)
                                      ( 649.855,1143.605) ( 649.855, 915.035)
                                      ( 632.860, 915.035) ( 632.860, 490.760)
                                      ( 598.545, 490.760) ( 598.545,   0.000)
                                      (   0.017,   0.000)
-------------------------------------------------------------------------------
#END_CHECK_RECTILINEARCELL
```

Report for -phys_property Option

```
#BEGIN_CHECK_PHYSICALPROPERTY
             List of placement properties
-------------------------------------------------------------------------
Cell name  PR boundary   Cell height   Unit tile  Coordinate  Tile pattern
-------------------------------------------------------------------------
DFFX1      (0,0)(11.2,5.04)      1xH   unit       (0,0)       R0|R0_MX
-------------------------------------------------------------------------

#END_CHECK_PHYSICALPROPERTY

             List of routing properties
  ---------------------------------------------------------------
              Preferred      Track
   Layer      direction    direction   Offset    Pitch   Remarks
  ---------------------------------------------------------------
   METAL1         H            H       0.280     0.560   OK
   METAL2         V            V       0.330     0.660   OK
  ---------------------------------------------------------------
```

Report for -physical_only_cell Option

```
#BEGIN_CHECK_PHYSICALONLYCELL
   Total number of physical only cells:1 (out of 125)
             List of physical only cells
  ------------------------------------------------------
   Cell Name          Cell type       Property
  ------------------------------------------------------
   FILL8              FillerCell     With metal
  ------------------------------------------------------

#END_CHECK_PHYSICALONLYCELL
```

Report for -tech_consistency Option

Reports differences between main and reference libraries.

```
#BEGIN_CHECK_TECH_CONSISTENCY
Warning: Inconsistent Data for Layer 57
        Main Library (mw_design)    |   Reference Library (my_lib)
        Layer Name     GP1          |   KLLBUMP
        Mask Name      via999       |   kllbump         (abcd13)
Warnings found in technology consistency checking.
#END_CHECK_TECH_CONSISTENCY
```

A

Technology File

This appendix provides instructions for creating and loading a technology file and information about each of the technology file sections.

The appendix is organized in the following sections:

- General Information
- Creating and Loading a Technology File
- Technology File Contents

Important:

For data preparation involving the technology file, you need to use the latest version of Milkyway to ensure the most complete process technology support.

General Information

This section explains the conventions used in this appendix, describes the general format of a technology file, and explains how to create and load a technology file. It also lists the cell library characteristics that can be defined.

Conventions

The following conventions are used in the examples and syntax descriptions in this appendix:

- Functions and keywords appear in `code` font and must be typed with the exact spelling given, although the characters can be uppercase or lowercase.
- Examples and variables appear in `code` font.
- Parentheses ( ) should be typed as they appear in the syntax.

General Format

This section describes the general format of the technology file.

Technology File Extension

An editable technology file has a `.tf` extension.

Technology File Sections

Each technology file contains several sections. The sections should appear in the order in which they are listed in the first bullet point and should adhere to the following basic guidelines:

- Each section begins with a keyword, such as Technology, Stipple, or PrimaryColor.
- A keyword is followed by an open brace (`{`)
- The section attribute statements follow.
- Each section ends with a close brace (`}`)

Example

```
SectionKeyword{
      attribute statements
}
```

Note:
You can add comments to the technology file by placing a slash and an asterisk (/*) at the beginning of a comment and an asterisk and a slash (*/) at the end. All text between /* and *” is ignored.

Reserved Words

The argument variables in the technology file sections are reserved words and cannot be used outside the context specified in the documentation.

Note:
These reserved words are case-sensitive.

Cell Library Characteristics

A technology file defines the following characteristics of a cell library:

- Units
- Graphical characteristics, including
  - Colors
  - Stipple patterns
  - Line styles
- Layers
- Coupling capacitance (for timing-driven layout)
- Dielectric value used to calculate sidewall capacitance
- Devices
- Design rules
- Capacitance model specifications

Creating and Loading a Technology File

You create a technology file by using the syntax descriptions in this appendix or by editing an existing technology file, as shown in the following procedures. The procedures explain how to edit a technology file associated with a library and then how to load the edited file back to that library, or to another library.

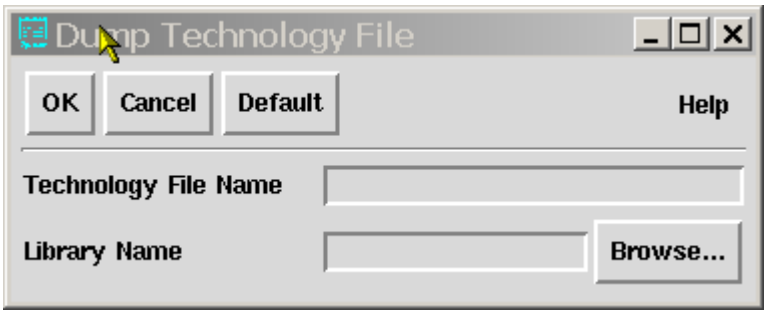
To perform the operations in this section, you use Milkyway. For information about starting Milkyway, see Chapter 1, “Starting the Data Preparation Process.”

Modifying an Existing Technology File

To modify the technology file associated with a library,

1. Enter `cmDumpTech` or choose Tech File > Write To File.

   The Dump Technology File dialog box appears.



2. Type the technology file name.

   This is the name you want to assign to the editable technology file to be created. Generally, this name has the extension .tf to identify the file as an editable technology file, but this extension is not mandatory.

3. Type the library name.

   This is the name of the library from which you want to unload the technology information.

4. Click OK.

   The `cmDumpTech` command extracts the technology information and places it in a file with the name you specified.

5. Using a UNIX text editor, edit the technology file.

   Using the syntax information in this appendix, you can add device definitions, change design rules, and make other changes to the specifications set in the technology file.

Loading a Technology File

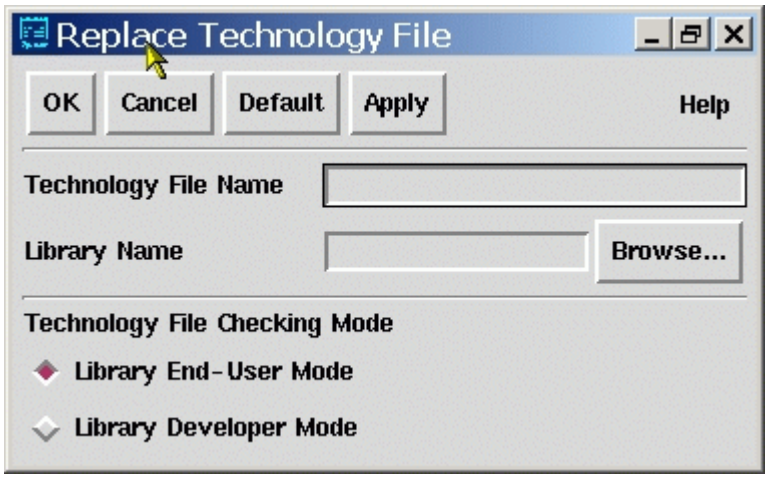
To load a technology file,

1. Enter `cmReplaceTech` or choose Tech File > Replace.

   The Replace Technology File dialog box appears.

Caution!

Running the `cmReplaceTech` command can change the definition of system layers in previous software versions. This command enabled the software and technology file to support more than six layers of metal. For more information regarding the automatic renaming of system layers, see "Layer Section" on page A-25.



2. Type the technology file name.

   This is the name of the technology file you created.

3. Type the library name.

   This is the name of the library into which you want to load the technology file.

4. Specify a technology file checking mode.

   Use the Technology File Checking Mode option to specify the severity of the technology file checker compliance checks.

   - Library End-User Mode (default): Specifies the least severe checks. If errors exist, the technology file data is not replaced.
   - Library Developer Mode: Specifies the most severe checks. This mode is for library developers. If any noncompliant issues exist, the technology file data is not replaced.

5. Click OK.

   The technology file information is added to the library information file.

Technology File Contents

The technology file specifications are divided into technology file sections, as listed in Table A-1.

Table A-1 Technology File Contents

Technology file section	Function
Technology Section	Specifies units and unit values
PrimaryColor and Color Sections	Defines the primary and display colors that a Synopsys tool uses to display designs in the library
Stipple Section	Defines the stipple patterns that a Synopsys tool uses to display designs in the library
LineStyle Section	Defines the line styles that a Synopsys tool uses to display designs in the library
Layer Section	Defines specific layer definitions
Tile Section	Defines the unit tiles
FringeCap Section	Defines capacitance information for interconnect layers when they are overlapping or parallel to each other
ContactCode Section	Defines the devices used in designs in the library
DesignRule Section	Defines the design rules that apply to designs in the library
PRRule Section	Defines cell row spacing
CapModel and CapTable Sections	Defines timing information
ResModel Section	Allows the resistance and temperature coefficient of the layer to be expressed as a function of wire width
DensityRule Section	Defines density rules
SlotRule Section	Defines slot rules

Database units stored in the database represent the user unit divided by a specified multiple of 10.

When you set the unit specifications for a library, you need to understand the limitations you are placing on your designs. These limitations are explained in “Design Considerations” on page A-7.

Technology Section

Use the Technology section of a Milkyway technology file to specify units and unit values.

Design Considerations

Before you set the unit specifications for a library, you need to understand the limitations you are placing on your designs. Because the unit specifications are stored as 32-bit integers, the combination of the unit specification and the precision specification sets the maximum dynamic range of a measurement.

The range of integers that you can store is from -2^{31} to $2^{31} - 1$ (–2,147,483,648 to +2,147,483,647). If you set the unit length specification to micron and the length precision specification to 1,000, the maximum dynamic range of distance is from –2,147,483.648 to +2,147,483.647 microns.

If you want to measure capacitance down to 0.0001 picofarad (pF), you need to set unit capacitance to pF and capacitance precision to 10,000. Consequently, the maximum capacitance that can be measured is 0.2147 microfarad (mF).

Defining Units

The units you can define for a Milkyway technology file are

- Dielectric
- Distance
- Time
- Capacitance
- Resistance
- Inductance
- Power
- Voltage
- Current
- Spacing

Example

Example A-1 shows how to define units in the Technology section of a technology file.

Example A-1 Technology Section of a Technology File

```
Technology {
      dielectric = 0.000000e+00
      unitLengthName = "micron"
      lengthPrecision = 1000
      gridResolution = 50
      unitTimeName = "ns"
      timePrecision = 100
      unitCapacitanceName = "pf"
      capacitancePrecision = 10000
      unitResistanceName = "kohm"
      resistancePrecision = 10
      unitInductanceName = "nh"
      inductancePrecision = 1000
      unitPowerName = "mw"
      powerPrecision = 10000
      unitVoltageName = "V"
      voltagePrecision = 1000
      unitCurrentName = "mA"
      currentPrecision = 1000000
      minBaselineTemperature = 25
      nomBaselineTemperature = 25
      maxBaselineTemperature = 25
      fatTblSpacingMode = 0
      fatWireExtensionMode = 0
}
```

Note:

Astro automatically creates layers for place and route functionality between 188 and 255. If your needs do not include Astro routing and you want to use these layers for other physical data, you can set the `useSystemLayers` flag to 1. When you set this flag, standard Astro place and route layers will not be created and layers 188 through 255 can be defined within the technology file.

Dielectric

This section describes the technology file statement you use to define the dielectric unit.

dielectric

The `dielectric` attribute specifies the relative permittivity of SiO2 that is to be used to calculate sidewall capacitance with a dielectric statement in the technology file.

You determine the dielectric unit by dividing the unit for measuring capacitance by the unit for measuring distance. For example,

$$\text{dielectric} = \frac{\text{capacitance unit}}{\text{distance unit}}$$

Distance

This section describes the technology file statements you use to define distance units.

unitLengthName

The `unitLengthName` attribute defines the linear distance unit. The valid values are

- `micron`
- `mil`

lengthPrecision

The `lengthPrecision` attribute sets the number of database units per user unit, defining how accurately distance measurements are stored in the database.

For example, if you set `unitLengthName` to micron and `lengthPrecision` to 1,000, the database unit will be 0.001 micron. Thus, all distance measurements are rounded to the nearest 0.001 micron.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

gridResolution

The `gridResolution` attribute sets the manufacturing grid resolution in database units. For example, if you set `unitLengthName` to micron and `lengthPrecision` to 1,000, the database unit will be 0.001 micron. If the minimum grid resolution of the manufacturing process is 0.1 micron, `gridResolution` should be set to 100.

When you are placing an object with a Synopsys tool. the tool places the point of origin on the grid.

Time

This section describes the technology file statements you use to define time units.

unitTimeName

The `unitTimeName` attribute defines the time unit.

Table A-2 shows the valid units for `unitTimeName`.

Table A-2 Valid Units for unitTimeName

Unit	Value
`fs`	1 x 10-15 second
`ps`	1 x 10^{-12} second
`ns`	1 x 10^{-9} second
`us`	1 x 10^{-6} second
`ms`	1 x 10^{-3} second
`s`	second

timePrecision

The `timePrecision` attribute defines how accurately time measurements are stored in the database.

For example, if you set `unitTimeName` to `ns` and `timePrecision` to 100, the database unit will be 0.01 ns. Thus, all time measurements are rounded to the nearest 0.01 ns.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Capacitance

This section describes the technology file statements you use to define capacitance units.

unitCapacitanceName

The `unitCapacitanceName` attribute defines the capacitance unit.

Table A-3 shows the valid units for `unitCapacitanceName`.

Table A-3 Valid Units for unitCapacitanceName

Unit	Value
`ff`	1 x 10^{-15} farad
`pf`	1 x 10^{-12} farad

Table A-3 Valid Units for unitCapacitanceName (Continued)

Unit	Value
nf	1 x 10^{-9} farad
uf	1 x 10^{-6} farad
mf	1 x 10^{-3} farad
f	farad

capacitancePrecision

The `capacitancePrecision` attribute defines how accurately capacitance measurements are stored in the database.

For example, if you set `unitCapacitanceName` to `pf` and `capacitancePrecision` to 10,000, the database unit will be 0.1 femtofarad (ff). Thus, all capacitance measurements are rounded to the nearest 0.1 ff.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Resistance

This section describes the technology file statements you use to define resistance units.

unitResistanceName

The `unitResistanceName` attribute defines the resistance unit.

Table A-4 shows the valid units for `unitResistanceName`.

Table A-4 Valid Units for unitResistanceName

Unit	Value
mohm	1 x 10^{-3} ohm
ohm	ohm
kohm	1 x 10^{3} ohm
Mohm	1 x 10^{6} ohm

resistancePrecision

The `resistancePrecision` attribute defines how accurately resistance measurements are stored in the database.

For example, if you set `unitResistanceName` to `ohm` and `resistancePrecision` to 1,000, the database unit will be 0.001 ohm. Thus, all resistance measurements are rounded to the nearest 0.001 ohm.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Inductance

This section describes the technology file statements you use to define inductance units.

unitInductanceName

The `unitInductanceName` attribute defines the inductance unit.

Table A-5 shows the valid units for `unitInductanceName`.

Table A-5 Valid Units for unitInductanceName

Unit	Value
`fH`	1 x 10^{-15} henry
`pH`	1 x 10^{-12} henry
`nH`	1 x 10^{-9} henry
`uH`	1 x 10^{-6} henry
`mH`	1 x 10^{-3} henry
`H`	henry

inductancePrecision

The `inductancePrecision` attribute defines how accurately inductance measurements are stored in the database.

For example, if you set `unitInductanceName` to `nH` and `inductancePrecision` to 1,000, the database unit will be 0.001 nanohenry (nH). Thus, all inductance measurements are rounded to the nearest 0.001 nH.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Power

This section describes the technology file statements you use to define power units.

unitPowerName

The `unitPowerName` attribute defines the power unit.

Table A-6 shows the valid units for `unitPowerName`.

Table A-6 Valid Units for unitPowerName

Unit	Value
`fw`	1 x 10^{-15} watt
`pw`	1 x 10^{-12} watt
`nw`	1 x 10^{-9} watt
`uw`	1 x 10^{-6} watt
`mw`	1 x 10^{-3} watt
`w`	watt

powerPrecision

The `powerPrecision` attribute defines how accurately power measurements are stored in the database.

For example, if you set `unitPowerName` to `mw` and `powerPrecision` to 1,000, the database unit will be 0.001 milliwatt (mw). Thus, all power measurements are rounded to the nearest 0.001 mw.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Voltage

This section describes the technology file statements you use to define voltage units.

unitVoltageName

The `unitVoltageName` attribute defines the voltage unit.

Table A-7 shows the valid units for `unitVoltageName`.

Table A-7 Valid Units for unitVoltageName

Unit	Value
`fV`	1 x 10-15 volt
`pV`	1 x 10-12 volt
`nV`	1 x 10-9 volt
`uV`	1 x 10-6 volt
`mV`	1 x 10-3 volt
`V`	volt

voltagePrecision

The `voltagePrecision` attribute defines how accurately voltage measurements are stored in the database.

For example, if you set `unitVoltageName` to `mV` and `voltagePrecision` to 1,000, the database unit will be 0.001 milivolt (mV). Thus, all power measurements are rounded to the nearest 0.001 mV.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Current

This section describes the technology file statements you use to define current units.

unitCurrentName

The `unitCurrentName` attribute defines the current unit.

Table A-8 shows the valid units for `unitCurrentName`.

Table A-8 Valid Units for unitCurrentName

Unit	Value
`fA`	1 x 10^{-15} ampere
`pA`	1 x 10^{-12} ampere
`nA`	1 x 10^{-9} ampere
`uA`	1 x 10^{-6} ampere
`mA`	1 x 10^{-3} ampere
`A`	ampere

currentPrecision

The `currentPrecision` attribute defines how accurately current measurements are stored in the database.

For example, if you set `unitCurrentName` to milliampere (`mA`) and `currentPrecision` to 1,000, the database unit will be 0.001 mA. Thus, all power measurements are rounded to the nearest 0.001 mA.

To minimize database translation problems, you should specify a number in which only the most significant digit is nonzero.

Spacing

This section describes the technology file statements you use to define multiple-spacing rules for metal within the range of fat metal spacing.

fatTblSpacingMode

The `fatTblSpacingMode` attribute defines multiple spacing between two adjacent metals.

Mode 0

(Default) Downgrade `fatTblSpacing` index according to the value of `fatTblParallelLength`.

Mode 1

Downgrade `fatTblSpacing` by one index only.

fatWireExtensionMode

The `fatWireExtensionMode` attribute defines the extension range for connected wires.

Mode 0

(Default) The extension area includes wires connected to fat metal only. See Figure A-1 on page A-16.

Mode 1

The extension area includes wires within the defined range of the fat metal, including connected and unconnected wires. See Figure A-2 on page A-17.

Mode 2

Variation of mode 1. See Figure A-3 on page A-17.

Mode 3

Variation of modes 1 and 2. See Figure A-4 on page A-18.

Figure A-1 shows only those wires that are connected to fat metal.

Figure A-1 fatWireExtension - Mode 0

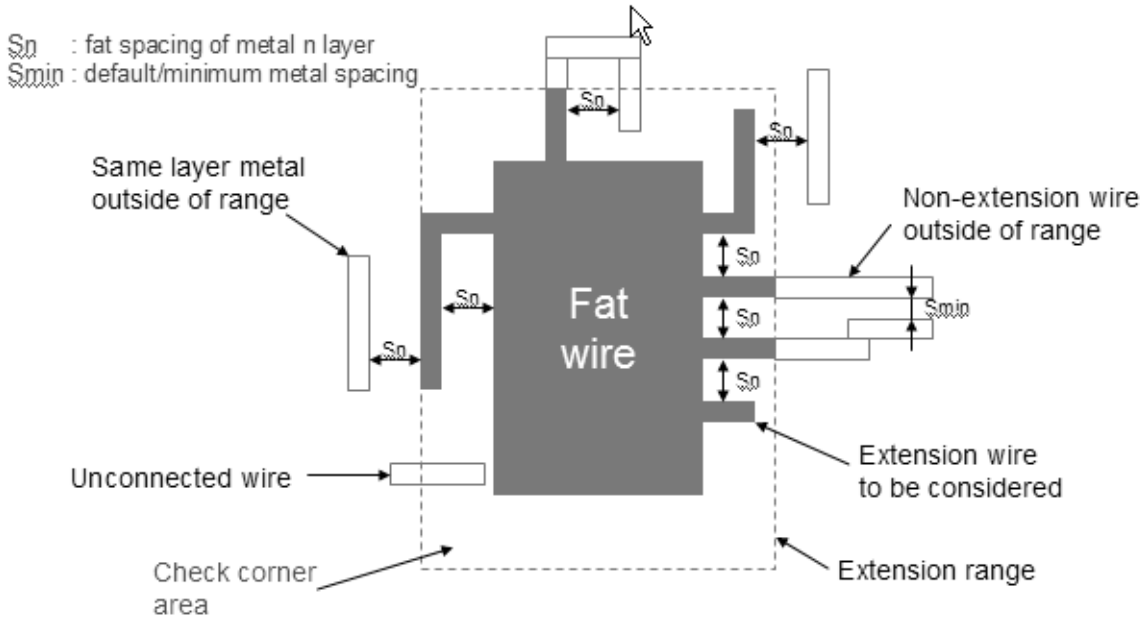


Figure A-2 shows wires that are within the extension range, including connected and unconnected. Figure A-3 and Figure A-4 show variations of Figure A-2.

Figure A-2 fatWireExtension–Mode 1

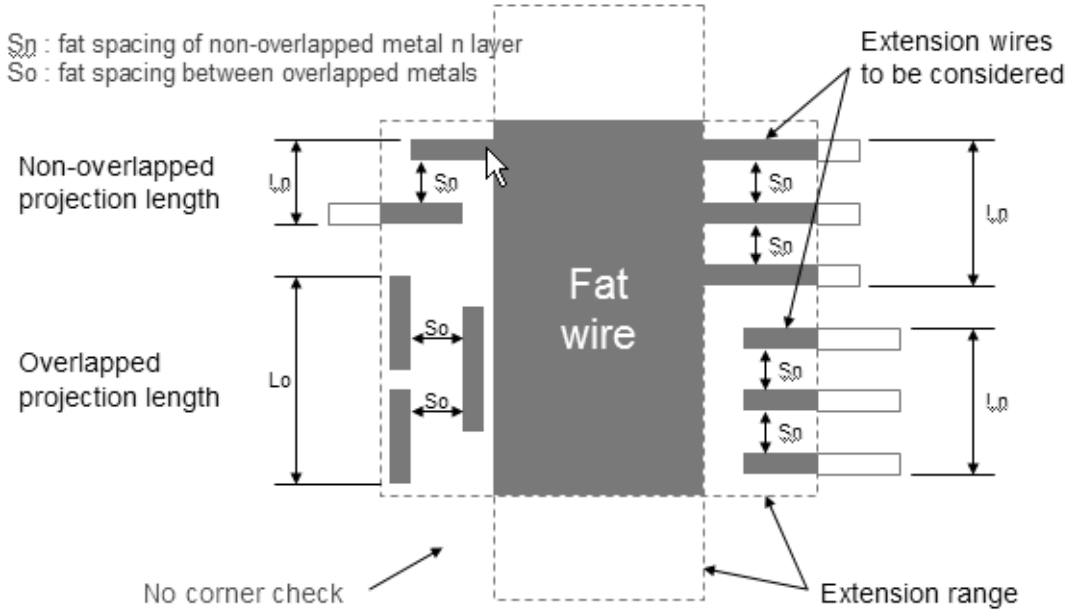


Figure A-3 fatWireExtension–Mode 2

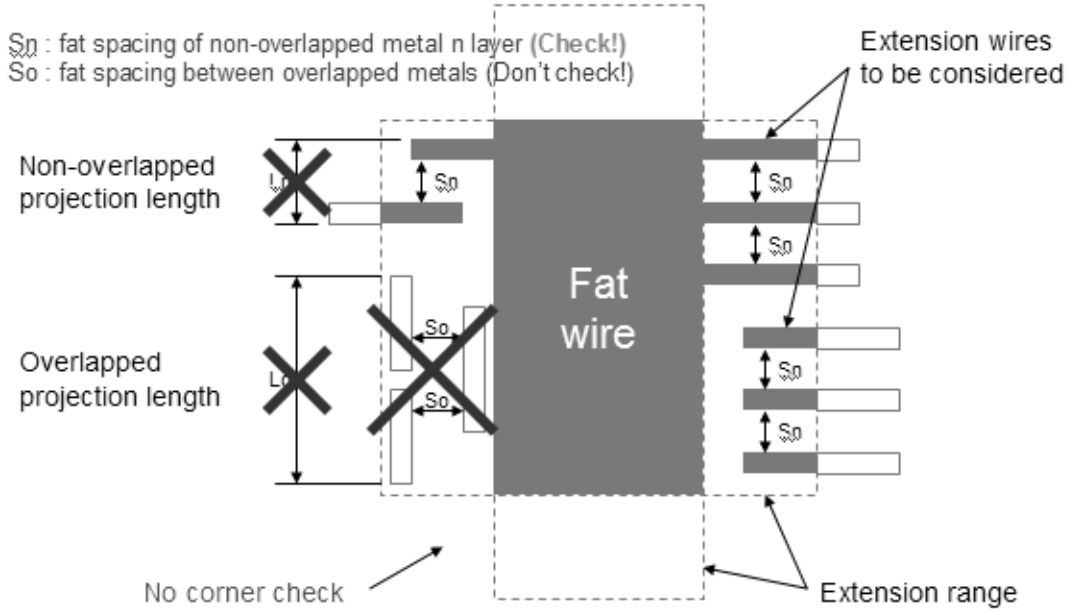
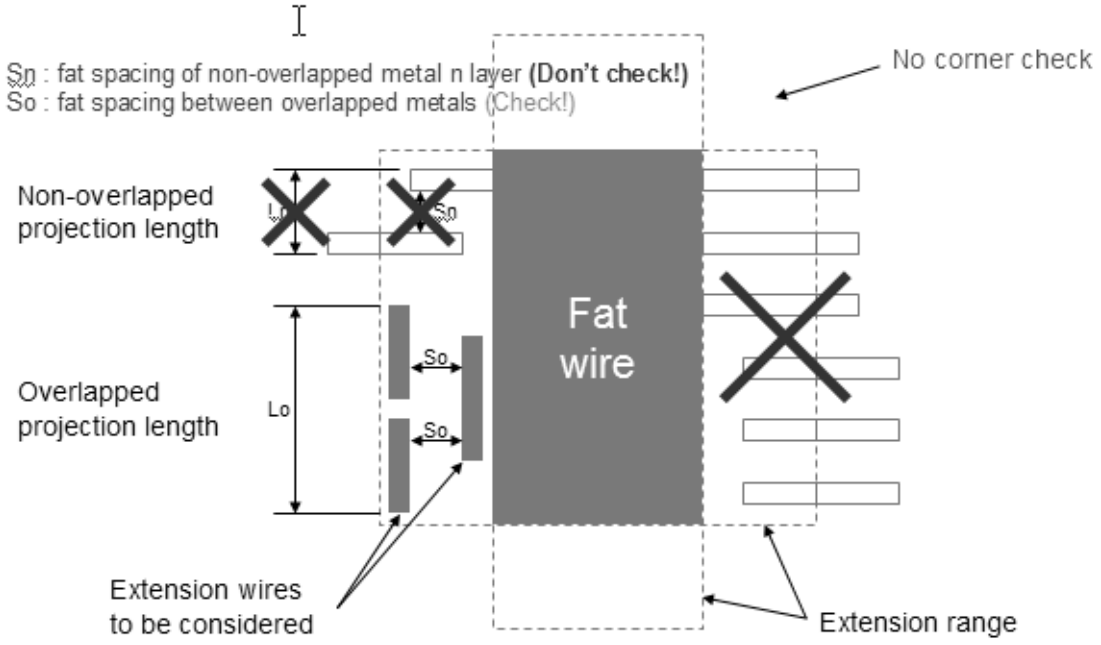


Figure A-4 fatWireExtension–Mode 3



minEdgeMode

A corner type can be concave or convex (see Figure A-5 on page A-19). The `minEdgeMode` value can be 0 or 1.

Violations of the minimum edge length rule are a function of the type of corner and the `minEdgeMode` value (see Figure A-6 on page A-19 and Figure A-7 on page A-20).

Mode 0

Any concave corner triggers a violation. A concave corner is formed by two adjacent edges when both are minimum length.

Mode 1

The minimum edge length is violated if the total number of consecutive minimum edges is greater than the value specified by `maxNumMinEdge`.

Figure A-5 shows the difference between concave and convex corners.

Figure A-5 Concave and Convex Corners

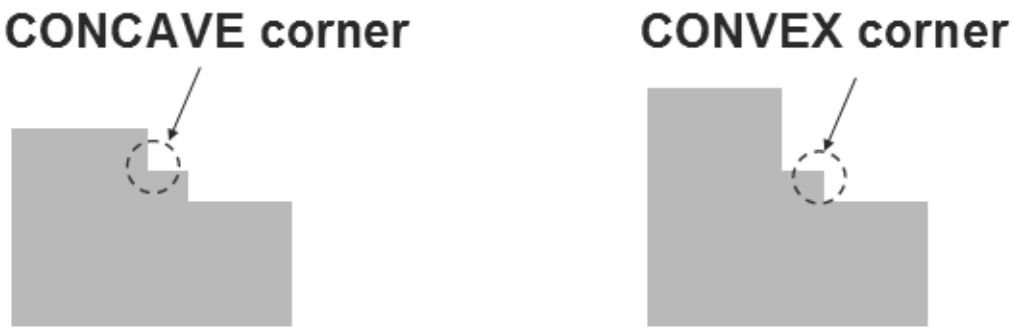


Figure A-6 and Figure A-7 show two scenarios for determining violations of the minimum edge length rule.

Figure A-6 minEdgeMode–Scenario 1

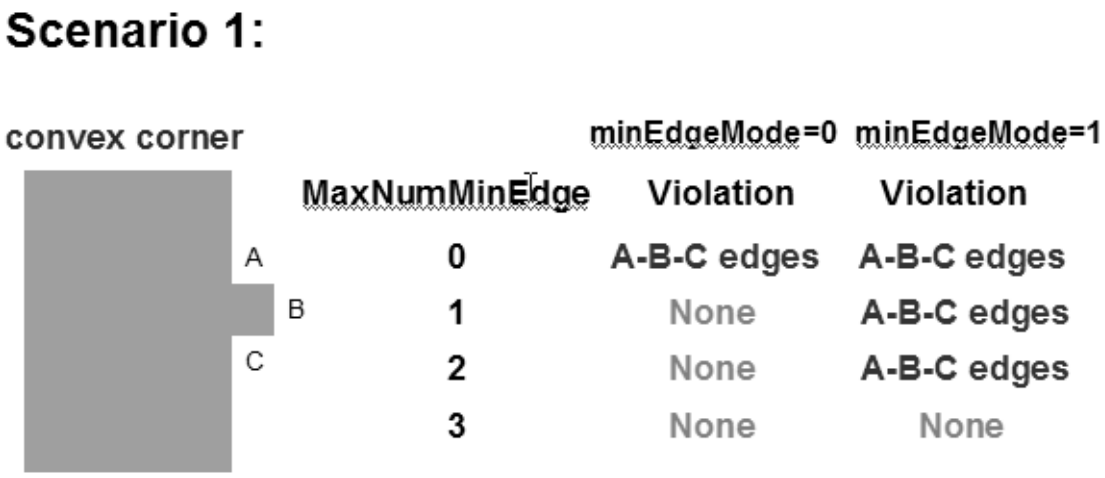
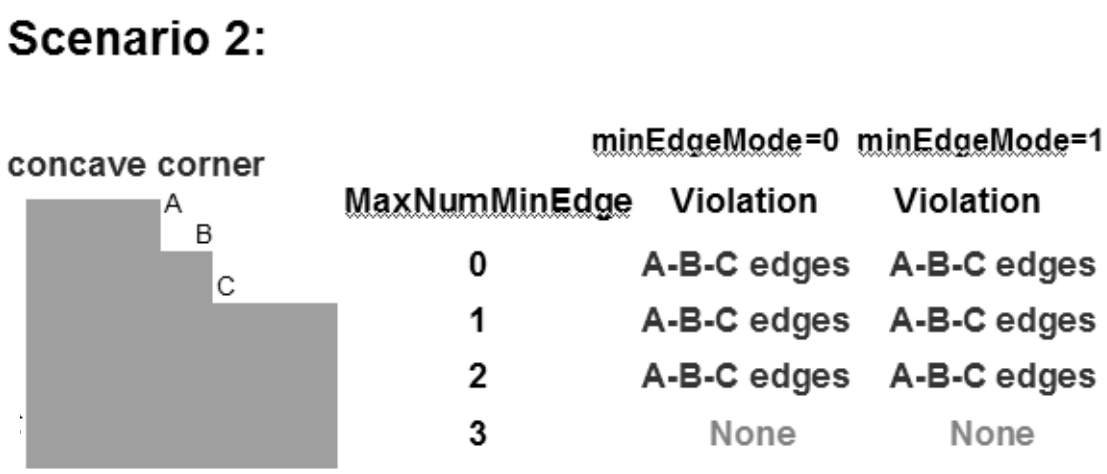


Figure A-7 minEdgeMode–Scenario 2



PrimaryColor and Color Sections

The PrimaryColor and Color sections of the technology file define the primary and display colors that a Synopsys tool uses to display designs in the library.

- The primary colors are the six basic colors used to create the display colors.
- The display colors are the colors you see when your Synopsys tool displays a cell in a graphics window.

To understand how the system combines primary colors to create display colors, you need to understand how a computer monitor creates colors on the screen.

Colors on a Computer Monitor

In any system, the term *primary colors* refers to the colors used to define all other colors. A computer monitor has three primary colors; a Synopsys tool has six.

All colors displayed on a computer monitor are created by combining red, green, and blue light.

Color Combinations

Combining colors of light produces results very different from combining the same colors of paint. Before you decide to change your colors, note the following results of combining colors of light:

- green + red = yellow
- green + blue = cyan
- blue + red = magenta

If you are working with computer graphics for the first time, you might find that combining colors can produce unexpected results.

Color Intensities

The intensity of any primary color on a computer monitor (red, green, or blue) can be from 0 to 255, with 255 producing the brightest intensity of the color.

Colors in a Synopsys Tool

The PrimaryColor section of a technology file defines two intensities each of red, green, and blue. These are the primary colors in a Synopsys tool. Using combinations of these six primary colors, the system creates the display colors you see when you display a cell in the graphics window.

The Layer section uses the display colors to specify how layers are displayed. Example A-3 defines only one display color. Your technology file can define up to 64 colors.

Color Combinations

A Synopsys tool stores each display color as a 6-bit binary number. Each bit of a 6-bit color number represents one of the primary colors, as follows:

Medium red	Light red	Medium green	Light green	Medium blue	Light blue

For example, color number 51, stored as the binary number 110011, consists of the primary colors represented by the bits that are set, including the following:

- Medium red
- Light red
- Medium blue

- Light blue

  When two colors overlap, the color produced is the result of an OR operation between the two color numbers. For example, if color number 51 (110011) overlaps color number 56 (111000), the result is color number 59 (111011).

Color Intensities

As described in Example A-2 on page A-22, the two intensities for red are

```
lightRed 90
mediumRed 180
```

If you add the two intensities together, the result is 270. Because a monitor cannot produce an intensity greater than 255, a display color that combines light red and medium red is red in an intensity of only 255.

In the case of color number 51, the combination of medium red and light red exceeds the maximum intensity of red the monitor can produce. Likewise, the combination of medium blue and light blue exceeds the maximum intensity of blue the monitor can produce. Therefore, creating color number 51 means combining red at an intensity of 255 and blue at an intensity of 255.

Predefined Colors

In the case of predefined colors, a Synopsys tool combines the primary colors in the intensities given in the PrimaryColor section to produce the display color.

PrimaryColor Section Example

Example A-2 shows a PrimaryColor section that defines the six primary colors:

Example A-2 Defining the Primary Colors

```
PrimaryColor {
      lightRed = 90
      mediumRed = 180
      lightGreen = 80
      mediumGreen = 175
      lightBlue = 100
      mediumBlue = 190
}
```

where the arguments for the PrimaryColor section of a technology file are the primary color names and intensities.

Custom Display Colors

Custom display colors are defined in the Color section of a technology file. You can create a custom display color that ignores the intensities given in the PrimaryColor section, by specifying the primary color intensities you want used for the color.

Color Section Example

Example A-3 shows a Color section that defines color 62 ("owhite"):

Example A-3 Defining a Color

```
Color 62 {
      name = "owhite"
      rgbDefined = 1
      redIntensity = 255
      greenIntensity = 255
      blueIntensity = 230
}
```

where Color is the technology file section name, 62 is the color number (can be any integer from 0 to 63), and the arguments are as follows:

`name`	The color name Valid values: Any string of up to 31 characters
`rgbDefined`	Indicates whether the color is red, green, blue (RGB)-defined Valid values: 1 (yes), 0 (no)
`redIntensity`	Identifies the color's red intensity Valid values: Any integer from 0 to 255
`greenIntensity`	Identifies the color's green intensity Valid values: Any integer from 0 to 255
`blueIntensity`	Identifies the color's blue intensity Valid values: Any integer from 0 to 255

Stipple Section

The Stipple section of the technology file defines the stipple patterns a Synopsys tool uses to display designs in the library. The Layer section uses the stipple patterns to specify how layers are displayed.

Examples

Example A-4 shows a “blank” stipple pattern. Example A-5 shows a “solid” stipple pattern. You need to create a similar Stipple section for each required pattern.

Example A-4 Specifying a Blank Stipple Pattern

```
Stipple "blank" {
      width = 2
      height = 2
      pattern = (0, 0, 0, 0)
}
```

Example A-5 Specifying a Solid Stipple Pattern

```
Stipple "solid" {
      width = 2
      height = 2
      pattern = (1, 1, 1, 1)
}
```

The preceding examples and their arguments are interpreted as follows:

`name`	Name that identifies the stipple pattern Valid values: Any string up to 15 characters
`width`	Horizontal dimension of the stipple pattern, in pixels
`height`	Vertical dimension of the stipple pattern, in pixels
`pattern`	Representation of the stipple pattern, with 1s representing pixels drawn with the color assigned to the layer and 0s representing pixels with no color (transparent) Note: The pattern representation is enclosed by parentheses.

LineStyle Section

The LineStyle section of the technology file defines the line styles a Synopsys tool uses to display designs in the library. The Layer section uses the line styles to determine how layers are displayed.

Example

Example A-6 shows how to define a line style. Because line styles are predefined, they do not actually need to appear in the technology file.

Example A-6 Specifying a Line Style

```
LineStyle "boundary" {
      width = 10
      height = 1
      pattern = (1, 0, 0, 1, 1, 1, 1, 1, 0, 0)
}
```

where "`boundary`" is the LineStyle name and the arguments are as follows:

`name`	Name that identifies a line style Valid values: Any string up to 15 characters
`width`	Horizontal dimension of the line style pattern, in pixels
`height`	Vertical dimension of the line style pattern, in pixels
`pattern`	Representation of the line style pattern, with 1s representing pixels drawn with the color assigned to the layer and 0s representing pixels with no color (transparent) Note: The pattern representation is enclosed by parentheses.

Layer Section

A technology file defines each layer by specifying variables for that layer. Those variables fall into several groupings, including

- Display attributes

  Display attributes specify how objects on the layer are displayed.

- Layout attributes

  Layout attributes associate a physical layer in the layout with the display layer.

- Parasitic attributes

  Parasitic attributes define the layer parasitics.

- Coupling capacitance attributes
- Physical attributes

  Physical attributes define physical characteristics of the layer and need to be specified if you are doing timing-driven layout.

You define physical attributes only when a physical layer is associated with the display layer and you plan to do timing-driven layout or layout parasitic extraction (LPE).

- Fat table attributes

  Fat table attributes define the fat wire dimension, threshold, and spacing.

- Enclosed area attributes
- Enclosed cut attributes
- Edge length attributes
- Via density attributes

If you plan to perform timing-driven layout, you need to provide complete specifications for all the interconnect layers.

Note:
If the layer is a contact cut, you should define the parasitic attributes in a ContactCode section.

Example

Example A-7 shows all the variables for a Layer section.

Example A-7 Layer Section

```
Layer "MET1" {
      layerNumber = 8
      isDefaultLayer = 0
      maskNameId = "metal1"
      color = "blue"
      lineStyle = "solid"
      pattern = "dot"
      blink = 0
      visible = 1
      selectable = 1
      panelNumber = 0
      defaultWidth = 1.0
      minArea = 1.0
      minWidth = 1.4
      minSpacing = 1.0
      sameNetMinSpacing = 0
      fatWireThreshold = 0
      fatThinMinSpacing = 0
      fatFatMinSpacing = 0
      stubThreshold = 0
      stubSpacing = 0
      maxLength = 0
      pitch = 2.2
      fatContactThreshold = 0
      maxSegLenForRC = 0
```

```
        capacitanceMultiplier = 0
        maxCurrDensity = 0
        maxIntraCapDistRatio = 0
        unitMinResistance = 0
        unitNomResistance = 0
        unitMaxResistance = 0
        temperatureCoeff = 0
        unitMinCapacitance = 0
        unitNomCapacitance = 0
        unitMaxCapacitance = 0
        unitMinInductance = 0
        unitNomInductance = 0
        unitMaxInductance = 0
        unitMinSideWallCap = 0
        unitNomSideWallCap = 0
        unitMaxSideWallCap = 0
        unitMinChannelCap = 0
        unitNomChannelCap = 0
        unitMaxChannelCap = 0
        unitMinChannelSideCap = 0
        unitMaxChannelSideCap = 0
        unitMinHeightFromSub = 0
        unitNomHeightFromSub = 0
        unitMaxHeightFromSub = 0
        unitMinThickness = 0
        unitNomThickness = 0
        unitMaxThickness = 0
        maxDeltaWidth = 0
        nomDeltaWidth = 0
        minDeltaWidth = 0
        defaultLayerCap = 0
        defaultInterFringeCap = 0
        fatTblDimension = 0
        fatTblThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
        fatTblSpacing = (0, 0, 0, 0, 0, 0, 0, 0)
}
```

Note that he maxLength attribute specifies the maximum length of a geometrical object (rectangle or polygon) allowed in a design. This rule applies to all geometrical objects on the current layer. It has the same unit as minWidth and defaultWidth. In Example A-7, maxLength is set to 0, which means this rule is unspecified on this layer. Therefore, the router will not do any checking.

Display Attributes

This section describes the display attributes and their valid values. The display attributes specify how objects on the layer are displayed.

`blink`

Sets the layer to blink (or not to blink) in the design window.

Valid values are 1 (on) or 0 (off).

`visible`

Sets the layer visibility.

Valid values are 1 (on) or 0 (off).

`selectable`

Sets the layer selectability.

Valid values are 1 (on) or 0 (off).

`color`

Defines the name or number of the color used to display the layer.

Valid values are any integer from 0 to 63 or the name of a color that is defined in the Color section.

If you specify a number, the color does not have to be defined in the Color section. See "Colors in a Synopsys Tool" on page A-21.

`pattern`

Defines the stipple pattern used to fill objects on the layer.

Valid values are the name of a stipple pattern defined in the Stipple section (see "Stipple Section" on page A-23).

`lineStyle`

Sets the defined line style for objects on the layer.

Valid values are the name of a style defined in the LineStyle section (see "LineStyle Section" on page A-24).

`panelNumber`

Specifies the subpanel for the layer.

Valid values are 0 through 3.

Layout Attributes

The layout attributes associate a physical layer in the layout with the display layer and define some design rules associated with objects on the layer. These include the following:

- Minimum width of objects on the layer
- Minimum spacing of objects on the layer
- Predominant separation distance between the centers of objects on the layer

You define the layout attributes of a layer in each layer section of a technology file.

The layout specifications are

`layerNumber`

Defines the number that identifies the layer.

Valid values are 1–160 (user defined) and 161–255 (system defined).

`isDefaultLayer`

Specifies the layer used for routing when there are multiple layers with the same `maskName` value.

Valid values are 0 or 1.

Note:

Only one layer with the same mask should be designated as the default.

`maskName`

Defines the physical layer associated with the display layer.

Valid values: In addition to the following predefined mask layers, you can specify any string of up to 31 characters.

- poly
- metal1... metal 12
- polyCont
- via1 ... via12
- passivation (for bonding pad pins)

`minArea`

Defines the minimum area rule of any dimension of an object on the layer.

`minWidth`

Defines the minimum width of any dimension of an object on the layer.

`maxWidth`

Allows the `geNewDRC` command to check the result of issuing the `axgSlotWire` command.

Note:

The signal router does not recognize the `maxWidth` rule.

`defaultWidth`

Defines the default width of any dimension of an object on the layer.

The default width is used with all operations except the design rule checker (DRC), which uses minimum width.

`minSpacing`

Defines the minimum separation distance between the edges of objects on the layer (if the objects are on different nets).

`sameNetMinSpacing`

Defines a smaller spacing rule than the default rule for two shapes belonging to the same net.

`maxStackLevel`

Defines the maximum number of vias that can stack at the same point.

Valid values are any whole number.

`pitch`

Defines the predominant separation distance between the centers of objects on the layer.

Astro uses the pitch you specify to generate wire tracks in the unit tile.

Note:

You cannot change the metal2 pitch after data preparation, because Milkyway uses the metal2 pitch during pin and blockage extraction.

Parasitic Attributes

The parasitic attributes define the layer parasitics. This section describes the parasitic attributes and their valid values.

Resistance per Square Attributes

Resistance is defined as the resistance per square (length divided by width) of a poly or metal layer.

The resistance specifications are

`unitMinResistance`

A floating-point number representing the minimum resistance.

`unitNomResistance`

A floating-point number representing the nominal resistance.

`unitMaxResistance`

A floating-point number representing the maximum resistance.

For contact layers, you specify the resistance of the contact cut in the ContactCode section (see “ContactCode Section” on page A-50).

Temperature Coefficient

Use the `temperatureCoeff` attribute to define the coefficient of the first-order correction to the resistance per square when the operating temperature is not equal to the nominal temperature at which the resistance per square variables are defined. Specifying a value for `temperatureCoeff` is optional.

During parasitic extraction, the resistance values are adjusted according to the operating temperature values specified by the `ntTimingSetup` command.

Note:
When you do not specify a value for `temperatureCoeff`, a value of 0.0 is used.

Capacitance per Unit Attributes

Capacitance is defined per square user unit of a poly or metal layer in a cell instance (or over a macro).

The capacitance specifications are

`unitMinCapacitance`

A floating-point number representing the minimum capacitance.

`unitNomCapacitance`

A floating-point number representing the nominal capacitance.

`unitMaxCapacitance`

A floating-point number representing the maximum capacitance.

For contact layers, you specify the capacitance of the contact cut in the ContactCode section (see “ContactCode Section” on page A-50)

Inductance per Unit Length Attributes

Inductance is defined per unit length of a poly or metal layer.

Note:
Synopsys tools do not use the inductance values.

The inductance specifications are

`unitMinInductance`

A floating-point number representing the minimum inductance.

`unitNomInductance`

A floating-point number representing the nominal inductance.

`unitMaxInductance`

A floating-point number representing the maximum inductance.

For contact layers, you specify the inductance of the contact cut in the ContactCode section (see “ContactCode Section” on page A-50).

Capacitance Multiplier

The `capacitanceMultiplier` attribute specifies the scaling factor by which you want to multiply the capacitance calculated for the layer. This line of the layer definition is optional. You specify this number only when a physical layer is associated with the display layer and you plan to do timing-driven layout or LPE.

The capacitance multiplier specification is

`capacitanceMultiplier`

A floating-point number representing the scaling factor. If you do not specify a value, 1.0 will be used.

Sidewall Capacitance per Unit Length

Sidewall capacitance is defined as the total sidewall capacitance per unit length (for both sides) of a poly or metal layer in a cell instance (or over a macro).

The sidewall capacitance specifications are

`unitMinSideWallCap`

A floating-point number representing the minimum capacitance.

`unitNomSideWallCap`

A floating-point number representing the nominal capacitance.

`unitMaxSideWallCap`

A floating-point number representing the maximum capacitance.

If you specify a zero (0), the Synopsys tool will calculate sidewall capacitance based on the height from the substrate and the thickness of the layer, as specified in the physical attributes (see “Physical Attributes” on page A-33).

Routing Channel Capacitance

Routing channel capacitance is defined per square user unit of a poly or metal layer in a routing channel.

The routing channel capacitance specifications are

`unitMinChannelCap`

A floating-point number representing the minimum capacitance.

`unitNomChannelCap`

A floating-point number representing the nominal capacitance.

`unitMaxChannelCap`

A floating-point number representing the maximum capacitance.

If you specify a zero (0), the value specified for the `cap` argument will be used.

Total Sidewall Routing Channel Capacitance

Total sidewall routing channel capacitance is defined as the total sidewall capacitance per unit length (for both sides) of a poly or metal layer in a routing channel.

The sidewall routing channel capacitance specifications are

`unitMinChannelSideCap`

A floating-point number representing the minimum capacitance.

`unitNomChannelSideCap`

A floating-point number representing the nominal capacitance.

`unitMaxChannelSideCap`

A floating-point number representing the maximum capacitance.

When you specify a zero (0), the value specified for the `sideCap` argument will be used.

Maximum Current Density

Use the `maxCurrDensity` attribute to define the maximum current density for a layer.

The current density specification is

`maxCurrDensity`

A floating-point number representing, in amperes per centimeter, the maximum current density the layer can carry.

Maximum Intracapacitance Distance Ratio

Use the `maxIntraCapDistRatio` attribute to specify the distance ratio required to control the LPE range. For example, where three times the `minSpacing` is required to control the extraction, specify a value of 3 for `maxIntraCapDistRatio`.

The intracapacitance specification is

`maxIntraCapDistRatio`

A floating-point number representing the distance ratio.

Physical Attributes

The physical attributes define physical characteristics of the layer and must be specified if you are doing timing-driven layout.

Height From Substrate

The height from the substrate is the distance, in user units, between the layer and the substrate.

The height specifications are

`unitMinHeightFromSub`

A floating-point number representing the minimum distance.

`unitNomHeightFromSub`

A floating-point number representing the nominal distance.

`unitMaxHeightFromSub`

A floating-point number representing the maximum distance.

These values are used for calculating the sidewall capacitance when you do not specify sidewall capacitance (see “Parasitic Attributes” on page A-30).

The height from substrate value is stored internally as a double-precision number and therefore is not limited by the database per-user unit limitations described in “Design Considerations” on page A-7.

Thickness

Thickness is defined as the user units of objects on the layer.

The thickness specifications are

`unitMinThickness`

A floating-point number representing the minimum thickness.

`unitNomThickness`

A floating-point number representing the nominal thickness.

`unitMaxThickness`

A floating-point number representing the maximum thickness.

This value is stored internally as a double-precision number and therefore is not limited by the database per-user unit limitations described in “Design Considerations” on page A-7.

Amount of Expansion and Shrinkage

Use the following attributes to specify the distance, in user units, by which objects on the layer will expand or shrink (on each side) from the design to the fabricated chip.

The expansion and shrinkage specifications are

`minDeltaWidth`

A floating-point number representing the maximum differential.

`nowDeltaWidth`

A floating-point number representing the nominal differential.

`maxDeltaWidth`

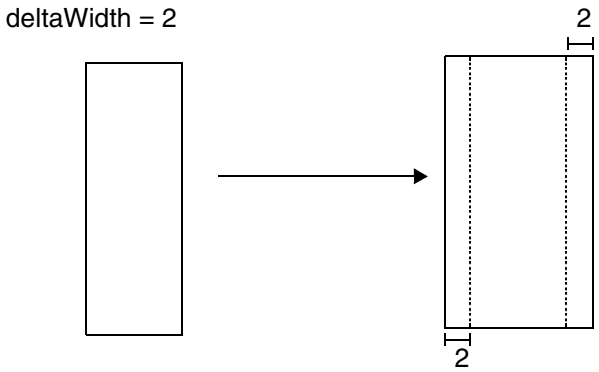
A floating-point number representing the minimum differential.

These values are used to adjust the width specified in the physical attributes for more accurate capacitance calculation.

The sign of a number has the following meaning:

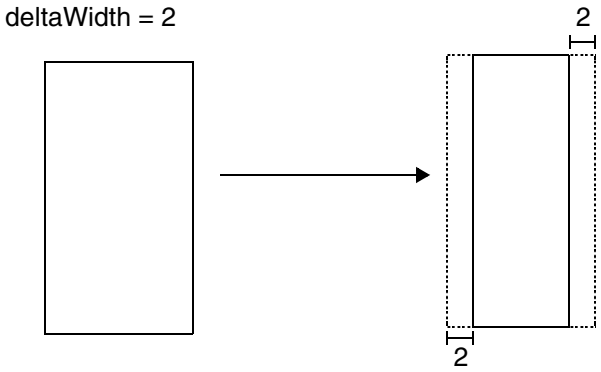
- A positive number indicates that objects expand, as shown in Figure A-8.

Figure A-8 Positive deltaWidth Example



- A negative number indicates that objects shrink, as shown in Figure A-9.

Figure A-9 Negative deltaWidth Example



Wire Segment Length in an LPE Model

Use the `maxSegLenForRC` attribute to define the maximum length, in user units, of a wire segment on the layer in an LPE model.

During LPE, the Synopsys tool splits wire segments longer than the specified length into segments of the specified length or less, to compute a more accurate model. For information on LPE, see the `aprLPE` description in Physical Implementation Online Help.

The wire segment length specification is

`maxSegLenForRC`

A floating-point number representing the wire segment length. If you specify a zero (0), the wire segments on the layer will not be split.

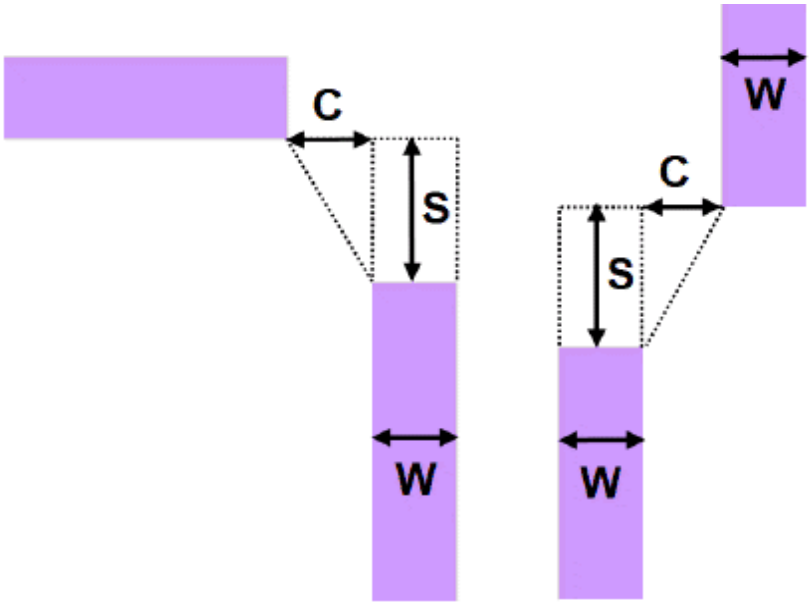
Stub Spacing

Stub spacing defines the minimum distance required between two objects on a layer when the distance the objects run parallel to each other is less than or equal to the specified threshold. Stub spacing lets you specify that objects that run parallel to each other for only a short distance (for example, a wire and a via) are to be spaced closer than the minimum spacing specified for the layer section.

stubMode

Specifies the spacing between a stub wire and other wires. There are four end-of-line spacing rules:

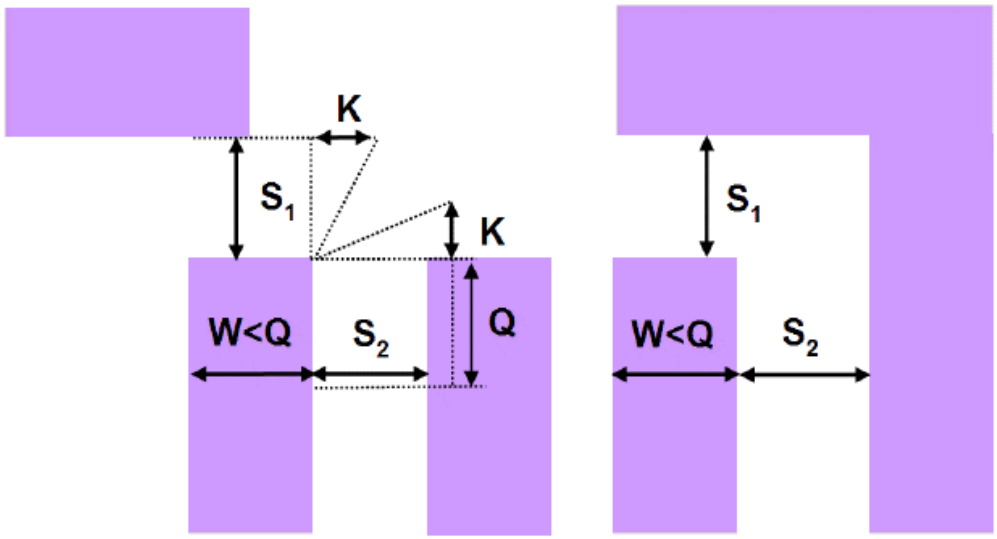
stubMode = 1




```
Technology {
        stubMode = 1
        }
Layer "Metal1" {
        stubSpacing = Se
        stubThreshold = Q
        endOfLineCornerKeepoutWidth = K
        }
```

Setting stubMode to 1 specifies the minimum spacing (stubSpacing) between two end-of-lines metal segments whose edge width W <= Q (stubThreshold). The end-of-line of the metal segment is measured from corner to distance K (endOfLineCornerKeepoutWidth). Otherwise, the default minimum spacing (minSpacing) is used.

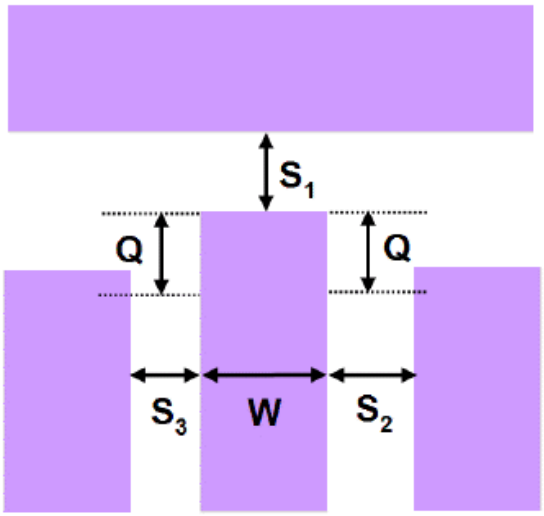
stubMode = 2




```
Technology {
          stubMode =  2
        }
Layer "Metal1" {
        stubSpacing = Se
        stubThreshold = QendOfLineCornerKeepoutWidth = K
        }
```

Setting stubMode to 2 specifies the minimum spacing at the dense end-of-line configuration. If a metal of width W <= Q (stubThreshold) has neighboring metal along two adjacent edges or any one edge less than Q (stubThreshold) distance from the corner of two adjacent edges, one of the spacing S_1 or S_2 should be >= stubSpacing. The neighboring metal is measured from corner to distance K (endOfLineCornerKeepoutWidth).

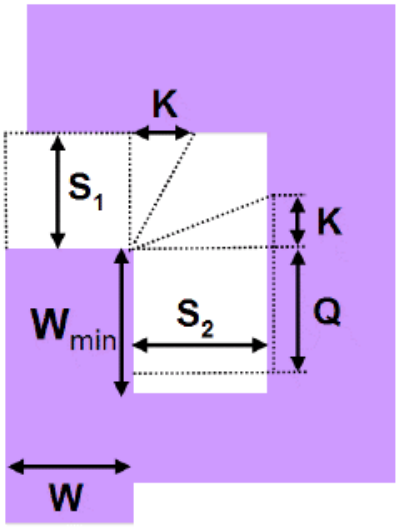
stubMode = 3




```
Technology {
      stubMode = 3
      }
Layer "Metal1" {
      stubSpacing = Se
      stubThreshold = Q
      }
```

Setting stubMode to 3 specifies the minimum spacing at the dense end-of-line configuration. If a metal of width (W) <= Q (stubThreshold) has neighboring metals along three adjacent edges and has neighboring metal along two adjacent edges within < Q (stubThreshold), one of the spacings S_1, S_2, or S_3 must be >= S_e.

stubMode = 4




```
Technology {
        stubMode = 4
        }
Layer "Metal1" {
        minWidth = Wmin
        stubSpacing = Se
        stubThreshold = Q
        endOfLineCornerKeepoutWidth = K
        }
```

Setting stubMode to 4 specifies the minimum spacing at the dense end-of-line configuration. If a metal has width of W < Q (stubThreshold) and there is no connecting metal within W_{min} (minWidth), and if it has neighboring metal along two adjacent edges or any one edge < Q (stubThreshold distance from the corner of two adjacent edges, one of the spacing S_1 or S_2 should be >= stubSpacing. The neighboring metal is measured from corner to distance K.

`stubSpacing`

The minimum distance required between the edges of two objects on the layer when the distance the objects run parallel to each other is equal to or less than the `stubThreshold` specification.

Valid values are any floating-point number less than the minimum spacing specified for the layer in the Layer section.

`stubThreshold`

The maximum distance two objects on the layer must run parallel to each other for the `stubSpacing` specification to apply.

Valid values are any floating-point number representing the distance.

Fat Wire Rules

Any wire exceeding the fat wire threshold defined in the technology file is interpreted by the router as a fat wire.

The fat wire specifications are

`fatContactThreshold`

The threshold for using a fat wire contact instead of the default contact.

Any wire on the specified layer whose width is equal to or greater than this threshold requires a fat wire contact. For more information, see the description of `isFatContact` in "ContactCode Section" on page A-50.

`fatFatMinSpacing`

The minimum distance required between wires on a layer when the widths of both wires are greater than or equal to `fatWireThreshold`.

`fatThinMinSpacing`

The minimum distance required between wires on a layer when the width of one of the wires is greater than or equal to `fatWireThreshold`.

`fatWireExtensionRange`

The extension range required for using the `fat wire` spacing rule instead of the default spacing rule, even when the wire in the extension portion is no longer fat. Any metal extending out from the fat wire must be treated as fat within the specified extension range. Figure A-10 shows a fat metal wire that has a projection wire with a normal width and another normal wire.

Figure A-10 Fat Metal Wire Having Projection Wire With Normal Width

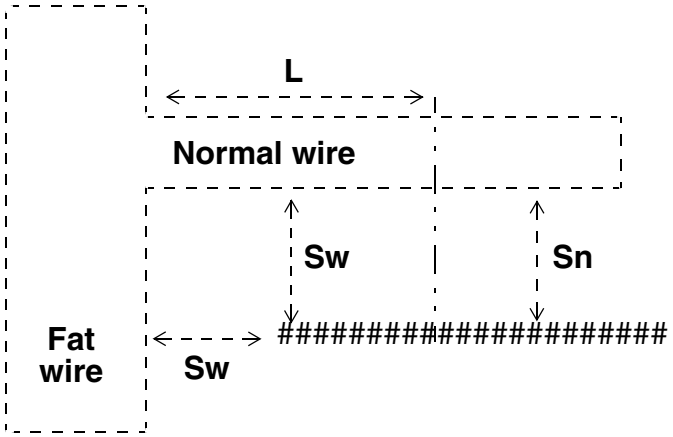
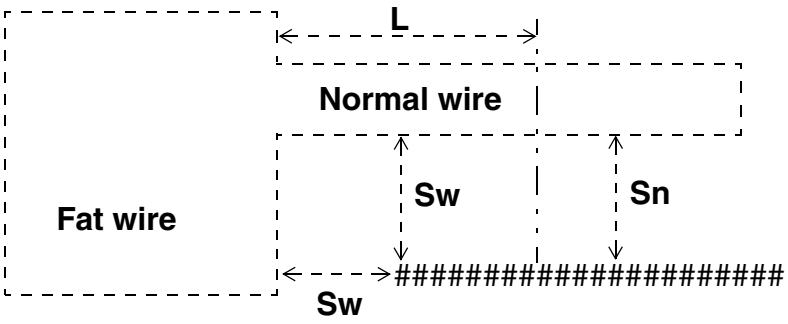


Figure A-11 shows that when the projection wire length is longer than threshold L, the fat wire spacing rule Sw is applied for the projection wire inside L. For wires outside L, the normal spacing rule, Sn, applies.

Figure A-11 Projection Wire Longer Than Threshold



`fatWireThreshold`

The threshold for using the fat wire spacing rule instead of the default spacing rule.

Fat Table Rules

The fat table specifications are

`fat2DTblFatContactMinCuts`

The minimal number of cuts (two-dimension).

`fat2DTblFatContactNumber` (two-dimension).

The contact code numbers for the fat wires (two-dimension).

`fatTblDimension`

The dimension of the fat table.

`fatTblExtensionRange`

The extension ranges required for a metal of the corresponding fat degree.

`fatTblExtensionMinCuts`

The minimum number of cuts for the wires within the extension range of fat wires.

`fatTblFatContactNumber`

The contact code numbers for the fat wires (one-dimension).

`fatTblMinEnclosedArea`

The minimum enclosed area for the fat wires.

`fatTblMinEnclosedAreaMode`

The minimum enclosed area for the fat wires.

`fatTblParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal is adjacent to another metal.

`fatTblSpacing`

The spacing required for the corresponding fat condition when two metals are adjacent.

`fatTblThreshold`, fatTblThreshold2

The thresholds used to define the different fat degrees of a metal in 90nm and 65nm designs, respectively.

Given a metal piece, the width of the metal is defined as
MIN(x_size, y_size).

According to the `fatTblThreshold` rule, the fat degree of a metal is determined by its width, satisfying a certain fat threshold. Define a metal of fat degree *i* when its width is greater than or equal to `fatThreshold[i]` and is less than `fatThreshold[i+1]`, when `fatThreshold[i+1]` exists; where `fatThreshold[i]` is less than `fatThreshold[i+1]`, are nonnegative integers; *i* belongs to `[0, fatTblDimension)` and is an integer.

When two metals are adjacent, assume that one metal is of fat degree `fatTblIdx1` and the other is of fat degree `fatTblIdx2`.

When no parallel length rules exist in the technology file, the spacing required between the two metals is the value from `fatTblSpacing[fatTblIdx1][fatTblIdx2]`.

When the fatTblParallelLength parallel length rule exists for two adjacent metals, the fat degree of each metal, fatTblIdx1 and fatTblIdx2, will be adjusted by the following parallel length conditions. Note that both adjacent metals will be adjusted individually. Now, assume you are looking at the metal of the fat degree fatTblIdx.

1. If the parallel length between the two metals is greater than or equal to the value of fatTblParallelLength[fatTblIdx], the fat degree of the current metal will still be the same as fatTblIdx.
2. Otherwise, when the parallel length between the two metals is less than the value of fatTblParallelLength[fatTblIdx], the fat degree of the current metal will continuously decrease by 1 (fatTblIdx_new = fatTblIdx -1) until the parallel length is greater than or equal to the value of fatTblParallelLength[fatTblIdx_new] or until the fat degree of the current metal becomes 0.

Assume that the fat degrees of two adjacent metals become fatTblIdx_1 and fatTblIdx_2 after the application of condition 1 or 2 to each metal, respectively. The spacing required between the two metals is the value from fatTblSpacing[fatTblIdx_1][fatTblIdx_2].

Example

Example A-8 shows the fatTable section from the list of variables in Example A-7:

Example A-8

```
fatTblDimension =    3
fatTblThreshold =    (0, 10, 20)
fatTblParallelLength =     (0, 15, 20)
fatTblExtensionRange =     (0, 2, 4)
fatTblSpacing =      (6, 7, 9, 7, 8, 10, 9, 10, 11)
```

A metal is of fat degree 1 when its width is greater than or equal to 10 but is less than 20. A metal is of fat degree 2 when its width is greater than or equal to 20. When two metals are adjacent, assume that one is of fat degree *i* and that the other is of fat degree *j*. When no parallel length rules exist, the spacing required between the two metals is fatTblSpacing[i][j].

The cases in Table A-9 show how the fat degree of each metal is affected according to the parallel length and the final spacing rule. Table A-8 is equivalent to the values in this table.

Table A-9 How Length and Width Affect Spacing

width1	width2	parallel_length	spacing
5	6	28	fatTblSpacing [0][0] = 6
24	24	13	fatTblSpacing [0][0] = 6

Table A-9 How Length and Width Affect Spacing

4	26	18	fatTblSpacing [0][1] = 7
24	26	18	fatTblSpacing [1][1] = 8
24	16	18	fatTblSpacing [1][1] = 8
24	16	28	fatTblSpacing [2][1] = 10
24	26	28	fatTblSpacing [2][2] = 11

Currently, the router (including `axDrouteDRC`) and `geNewDRC` obey this rule.

In the case of the `fatTblExtensionRange` rule, the extension range required for a metal of fat degree 0, 1, and 2 is 0, 2, and 4, respectively. That is, any metal extending out of a fat metal of fat degree 1 must be treated as fat metal of fat degree 1 within the specified extension range 2. Similarly, any metal extending out of a fat metal of fat degree 2 must be treated as fat metal of fat degree 2 within the specified extension range 4.

Currently, the router (including `axDrouteDRC`) obeys this rule.

Enclosed Area Rules

The `minEnclosedArea` attribute specifies the minimum area enclosed by ring-shaped wires or vias. This rule should be added in the Layer section of the technology file. For example,

```
Layer "METAL1" {
  ....
  minEnclosedArea = 0.6
}
```

Be sure to turn on `viaLoop` and check `axSetIntParam droute avoidViaLoop 1`.

The router checks for these kinds of loops and fixes them. The current implementation checks these kinds of loops whenever the `minEnclosedArea` attribute is specified. The value is ignored for the time being. In other words, the router might also flag violations when the enclosed area is 1.0, even though the rule is 0.6.

Enclosed Cut Rules

The enclosed cut specifications are

`enclosedCutMinSpacing`

The minimum spacing between two enclosed cuts.

`enclosedCutNeighborRange`

The distance range to define neighboring cuts.

`enclosedCutNumNeighbor`

The number of adjacent cuts for an enclosed cut.

`enclosedCutToNeighborMinSpacing`

The minimum spacing between the enclosed cut and the neighboring cuts.

Minimum Edge Length Rule

The minimum edge length specifications are

`maxNumMinEdge`

The maximum number of consecutive short edges.

`minEdgeLength`

The length for defining short edges.

Via Density Rule

The via density specifications are

`maxNumAdjacentCut`

The maximum number of adjacent cuts.

`adjacentCutRange`

The distance for defining adjacent cuts.

LayerDataType Section

Use the LayerDataType section to specify values from 0 to 255 for each layer. There are 47,872 (187 x 256) LayerDataType layers available. However, routing mask layers such as metal1 or via1 must be defined on a layer with a data type number of 0.

For manipulating data objects with nonzero data type numbers, you can define LayerDataType in conjunction with other layers in the technology file. The visibility, selectability, and display features can be independent from the corresponding layer for a LayerDataType.

The data objects (geometry or text) on a LayerDataType can be honored or skipped on its corresponding layer, depending on your requirements, if you indicate the `nonMask` field in the technology file.

By default, for reasons of backward compatibility, the `nonMask` field is FALSE for any layer data type. That is, all the geometries and texts on any layer data type will be honored on its corresponding layer.

If you prefer that the data objects of some LayerDataType not appear on its corresponding routing mask, you need to add `nonMask = 1` for that LayerDataType in the technology file.

Example

Example A-9 shows a typical LayerDataType section.

Example A-9 LayerDataType Section of a Technology File

```
LayerDataType "MET1ZONE" {
      layerNumber = 4
      dataTypeNumber = 3
      nonMask = 1
      visible = 1
      selectable = 1
      blink = 0
      color = "red"
      lineStyle = "solid"
      pattern = "dot"
}
```

Tile Section

A Tile section defines the unit tiles.

Example

Example A-10 shows a typical Tile section.

Example A-10 Tile Section of a Technology File

```
Tile "unit" {
      width = 16
      height = 168
}
```

where `unit` is the name of the unit tile (any string up to 31 characters) and the arguments are as follows:

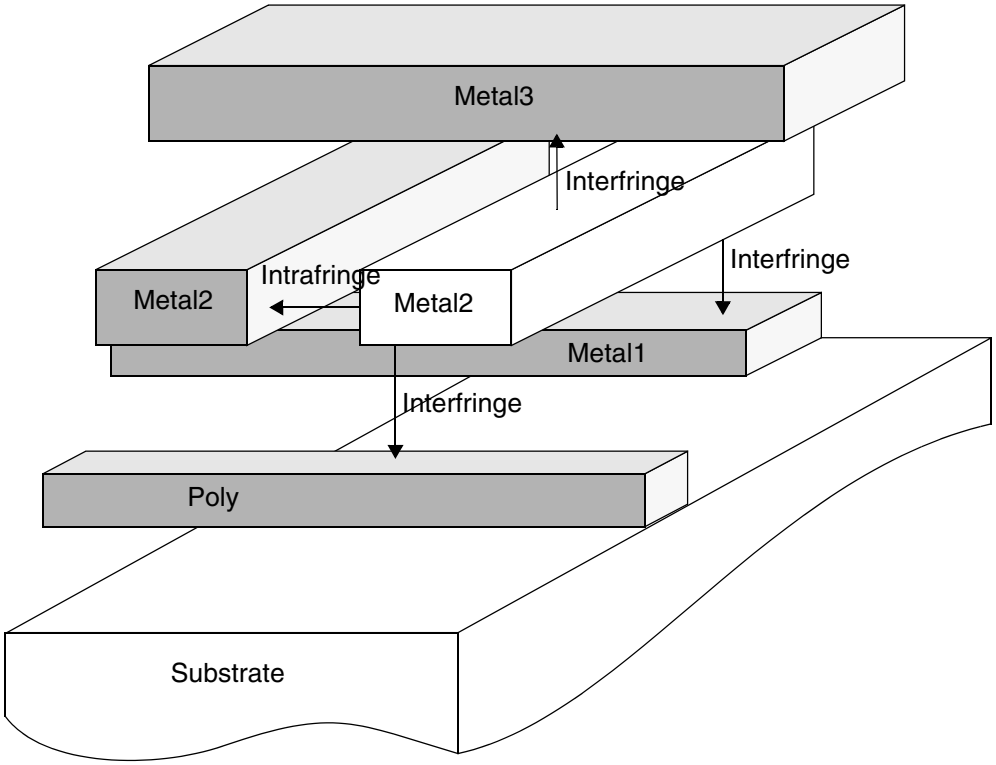
`tileWidth`	Width of a tile
`tileHeight`	Height of a tile

FringeCap Section

A FringeCap section specifies capacitance information for interconnect layers when they are overlapping or parallel to each other. This information includes the following:

- Capacitance per unit area when objects on different layers overlap (interfringe in Figure A-12)
- Capacitance per unit length when objects on the same layer are separated by the minimum spacing specified in the technology file Layer section (intrafringe in Figure A-12)

Figure A-12 Interfringe and Intrafringe Examples



Default values for coupling capacitance between adjoining parallel objects on the same layer or adjoining objects on different layers are defined in the respective Layer sections of the technology file. Other coupling capacitance values are entered in the FringeCap section.This information is used for calculating coupling capacitance for timing-driven layout.

Examples

Example A-11 shows a typical interfringe FringeCap section.

Example A-11 FringeCap Section (Interfringe)

```
FringeCap 4 {
      number = 4
      layer1 = "M2"
      layer2 = "M3"
      minFringeCap = 0.00022
      nomFringeCap = 0.00022
      maxFringeCap = 0.00022
}
```

Example A-12 shows a typical intrafringe FringeCap section.

Example A-12 FringeCap Section (Intrafringe)

```
FringeCap 4 {
      number = 4
      layer1 = "M2"
      layer2 = "M2"
      minFringeCap = 0.00017
      nomFringeCap = 0.00017
      maxFringeCap = 0.00017
}
```

The variables in these examples are defined as follows:

`number`	Coupling capacitance number Valid values: A whole number at least 1 less than the number of metal layers
`layer1`	One of the two metal layers on which different metal layers cross each other Valid values: M1, M2, and so on
`layer2`	One of the two metal layers on which different metal layers cross each other Valid values: M1, M2, and so on
`minFringeCap`	Minimum capacitance for the layer Valid values: A floating-point number
`nomFringeCap`	Nominal capacitance for the layer Valid values: A floating-point number

`maxFringeCap`	Maximum capacitance for the layer Valid values: A floating-point number

ContactCode Section

The ContactCode section of the technology file defines the devices used in designs in the library.

Examples

Example A-13 shows a typical ContactCode section of a technology file for DefaultContact.

Example A-13 ContactCode Section (DefaultContact)

```
ContactCode "PCON" {
      contactCodeNumber = 1
      cutLayer = "CONT"
      lowerLayer = "M1"
      upperLayer = "M2"
      isDefaultContact = 1
      upperLayerEncWidth = 0.6
      upperLayerEncHeight = 0.6
      lowerLayerEncWidth = 0.6
      lowerLayerEncHeight = 0.6
      cutWidth = 0.8
      cutHeight = 0.8
      minCutSpacing = 1.2
      maxNumRowsNonTurning = 1
      unitMinResistance = 0.00025
      unitNomResistance = 0.00025
      unitMaxResistance = 0.00025
      temperatureCoeff = 2.5e-06
      unitMinCapacitance = 0.0004
      unitNomCapacitance = 0.0004
      unitMaxCapacitance = 0.0004
}
```

Example A-14 shows a typical ContactCode section of a technology file for FatContact.

Example A-14 ContactCode Section (FatContact)

```
ContactCode "PCON" {
      contactCodeNumber = 1
      cutLayer = "CONT"
      lowerLayer = "M1"
      upperLayer = "M2"
      isFatContact = 1
      upperLayerEncWidth = 0.6
      upperLayerEncHeight = 0.6
      lowerLayerEncWidth = 0.6
```

```
        lowerLayerEncHeight = 0.6
        cutWidth = 0.8
        cutHeight = 0.8
        minCutSpacing = 1.2
        maxNumRowsNonTurning = 1
        unitMinResistance = 0.00025
        unitNomResistance = 0.00025
        unitMaxResistance = 0.00025
        temperatureCoeff = 2.5e-06
        unitMinCapacitance = 0.0004
        unitNomCapacitance = 0.0004
        unitMaxCapacitance = 0.0004
}
```

ContactCode Definitions and Values

The ContactCode definitions and values are as follows:

`contactCodeNumber`	Number that identifies the device. Valid values: Any integer from 1 to 255
`cutLayer`	Name of the display layer used for the cut.
`lowerLayer`	Name of the display layer used for one of the conduction layers.
`upperLayer`	Name of the display layer used for the other conduction layer.
`isDefaultContact`	Specifies the default contact. Valid values: 0,1 (for default contact) No more than one contact with the same cutlayer should be marked as the default. When no default contact is specified, the contact with the lowest number is the default.
`isFatContact`	Specifies that the contacts should be used when the wire is wider than the fatContactThreshold specification. Valid values: 0,1 (fat wire contact)—see “Fat Wire Rules” on page A-41.
`cutHeight`	Vertical dimension of the cut.
`cutWidth`	Horizontal dimension of the cut.

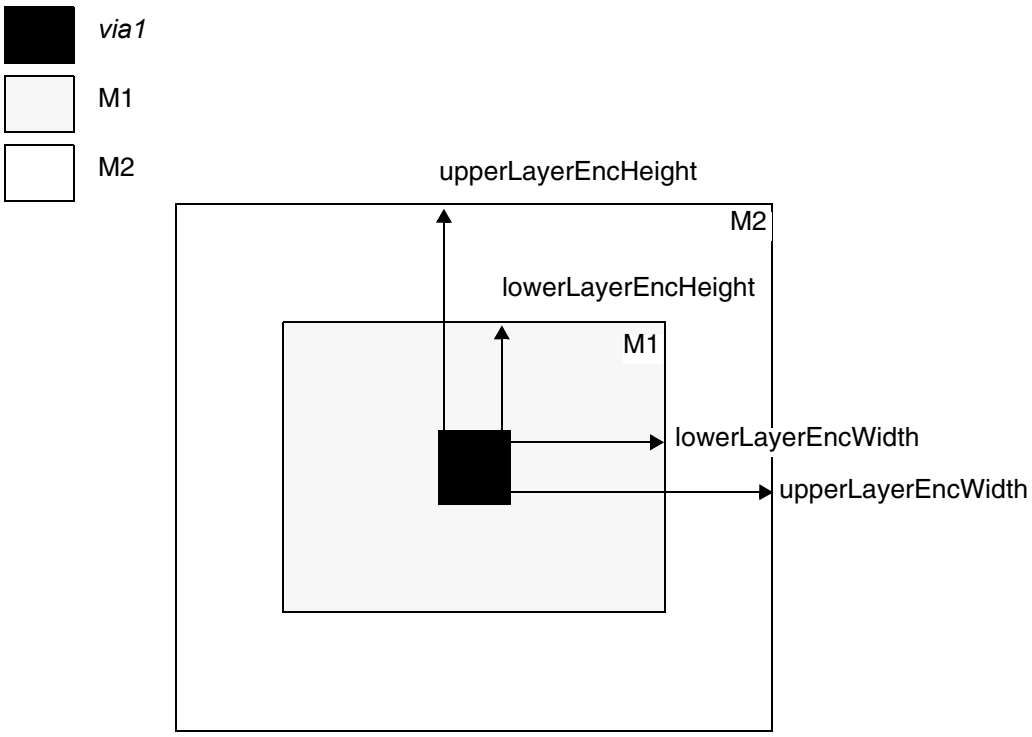
`upperLayerEncHeight`	Distance at which the first of two layers encloses the cut in the vertical dimension. Valid values: A floating-point number
`upperLayerEncWidth`	Distance at which the first of two layers encloses the cut in the horizontal dimension. Valid values: A floating-point number
`lowerLayerEncHeight`	Distance at which the second of two layers encloses the cut in the vertical dimension. Valid values: A floating-point number
`lowerLayerEncWidth`	Distance at which the second of two layers encloses the cut in the horizontal dimension. Valid values: A floating-point number
`minCutSpacing`	Minimum separation distance between the edges of the contact cut.
`viaFarmSpacing`	When defined as a nonzero value, the contactCode device can be a via.
`maxNumRows`	When viaFarmSpacing is defined in the same contactCode construct, the maxNumRows value represents the exact number of rows in each via farm in a via farm array; otherwise, the maxNumRows value represents the maximum number of rows in a via array.
`maxNumRowsNonTurning`	Restricts a via size by placing an upper limit on the number of rows of cuts allowed in the routing direction. Note: When the prerouter needs to switch layers without changing the routing direction, it uses a square for a drop via. The width and height of the square is the (fixed) width of the wires. Valid values: A positive integer (default is 1)

Figure A-13 shows a contact named via1 with the following physical attributes:

- `cutLayer` = v1
- `upperLayer` and `lowerLayer` = M2 and M1
- `cutWidth` = 1.2

- `cutHeight` = 1.2
- `upperLayerEncHeight` = 1.5
- `upperLayerEncWidth` = 1.8
- `lowerLayerEncHeight` = 2.8
- `lowerLayerEncWidth` = 3.9

Figure A-13 via 1 Contact



Parasitic Attributes

Device layer parasitics are defined in the ContactCode section of a technology file.

Resistance

Resistance is defined as the resistance per contact.

The resistance specifications are

`unitMinResistance`

A floating-point number representing the minimum resistance.

`unitNomResistance`

A floating-point number representing the nominal resistance.

`unitMaxResistance`

A floating-point number representing the maximum resistance.

Temperature Coefficient

The `temperatureCoeff` attribute defines the coefficient of the first-order correction to the resistance per square when the operating temperature does not equal the nominal temperature at which `unitMinResistance`, `unitNomResistance`, and `unitMaxResistance` are defined. This part of the layer definition is optional. During parasitic extraction, the resistance values are adjusted according to the operating temperature values specified by the `ntTimingSetup` command.

Note:
If you do not specify a `temperatureCoeff` value, 0.0 will be used.

Capacitance

Capacitance is defined as the capacitance per contact in a cell instance (or over a macro).

The capacitance specifications are

`unitMinCapacitance`

A floating-point number representing the minimum capacitance.

`unitNomCapacitance`

A floating-point number representing the nominal capacitance.

`unitMaxCapacitance`

A floating-point number representing the maximum capacitance.

Maximum Current Density

Use the maxCurrDensity attribute to define the maximum current density for a contact.

The current density specification is

`maxCurrDensity`

A floating-point number representing, in amperes per centimeter, the maximum current density the contact can carry.

DesignRule Section

The DesignRule section of the technology file defines the design rules that apply to designs in the library. All measurements in this section are in the user units specified in the Technology section (see “Technology Section” on page A-7) of the technology file.

Note:

You specify the minimum width allowed for an object on a layer as described in “Layer Section” on page A-25.

Syntax

Define the design rules as follows:

```
DesignRule {
          layer1 = "name"
          layer2 = "name"
          ruleType = rule attributes
}
```

where the arguments are as follows:

`layer1` — One of the layers to which the rule applies

Valid values: Name or number of a defined cut layer (must be different from layer2)

`layer2` — Another layer to which the rule applies

Valid values: Name or number of a defined cut layer

`ruleType` — Specifies the type of rule you are defining

Valid variables:

`minSpacing` defines the minimum distance required between via cut layers.
`diffNetMinSpacing` instructs the detail router to use a different spacing rule to check same-net/diff-net cut-to-cut spacing.

`stackable` indicates that Astro can stack two vias if their cuts are on layer1 and layer2.

`minEnclosure` defines the minimum distance at which a layer must enclose another layer when the two layers overlap.

`endOfLineEnclosure` defines an enclosure size to specify the end-of-line rule for routing wire segments.

`ruleAttributes`	Attributes of the rule Valid values: Depends on `ruleType`

Examples

Example A-15 shows a typical DesignRule section of a technology file:

Example A-15 DesignRule Section of a Technology Library

```
DesignRule {
      layer1 = "VIA1"
      layer2 = "VIA2"
      minSpacing = 0.0
      diffNetMinSpacing = 0.4
}
```

Minimum Spacing

The `minSpacing` variable specifies the minimum spacing between the specified layers.

Valid values are any floating-point number.

Use Different Spacing

The `diffNetMinSpacing` variable instructs the detail router to use a different spacing rule to check same-net or different-net cut-to-cut spacing.

Valid values: To avoid a BPV-generated via region from becoming too large, set m1 – m1 spacing to 0.36 to run BPV (m1 – m1 spacing + m1 – contact enc + m1 – via1 enc >= cont via1 diffnet spacing).

Stackable

The `stackable` variable indicates whether Astro can stack two vias if their cuts are on the two layers identified in the DesignRule section.

Valid values are 1 (yes), 0 (no).

When no spacing rule exists between the two layers, the `stackable` attribute does not need to be specified and the router allows contacts to overlap arbitrarily. When a spacing rule exists between the two layers and `stackable` is set to 1, the route does one of the following:

- Separates the contacts with the spacing rule
- Stacks the two contacts, aligning the centers

Minimum Enclosure

The `minEnclosure` variable specifies the minimum distance at which layer1 must enclose layer2 when the two layers overlap.

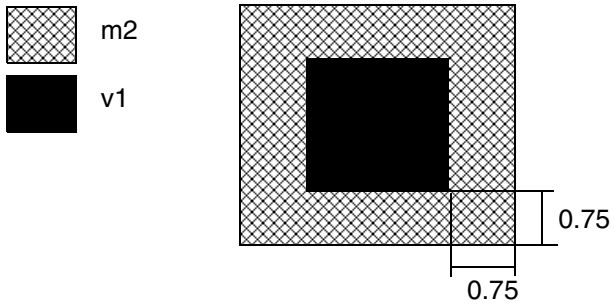
Valid values are any floating-point number.

Note:

Predefined layers are layers with numbers greater than 160. To see the names of these layers, display the Layer Panel, as explained in the “Physical Implementation Getting Started” topic in Physical Implementation online Help.

Figure A-14 represents the minimum enclosure rule, which says that when the m2 and v1 layers overlap, the m2 layer must enclose the v1 layer at a distance of 0.75 user units.

Figure A-14 minEnclosure Rule Example



End-of-Line Enclosure

The `endofLineEnclosure` variable defines an enclosure size to specify the end-of-line rule for routing wire segments.

Example A-16 shows how to define an end-of-line rule in a DesignRule section of a technology file:

Example A-16 Defining the End-of-line Rule in the Designrule Section of a Technology Library

```
DesignRule {
   layer1 = "Metal2"
   layer2 = "Via2"
   endOfLineEnclosure = 0.15
}
```

PRRule Section

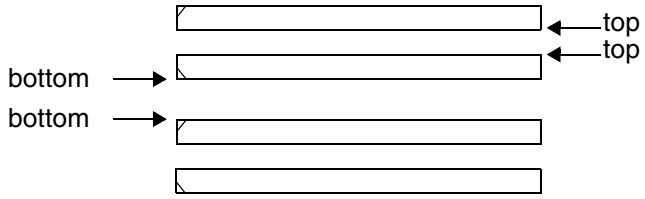
The PRRule section of a technology file defines cell row spacing.

Cell Row Spacing

When using double-back cell rows (see Figure A-15) in your floorplan, you specify the following row spacing rules

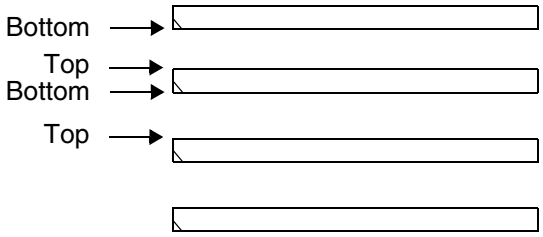
- Between top edge and top edge
- Between bottom edge and bottom edge

Figure A-15 Row Spacing Rule for Double-Back Cells



When not using double-back cell rows in your floorplan, you specify a row spacing rule between the top edge and the bottom edge (see Figure A-16).

Figure A-16 Row Spacing Rule for Non-Double-Back Cells



Example

Example A-17 shows a typical PRRule section of a technology file.

Example A-17 PRRule Section of a Technology File

```
PRRule {
      rowSpacingTopTop = 2.8
      rowSpacingTopBot = 2.8
      rowSpacingBotBot = 1.4
      abuttableTopTop = 1
      abuttableTopBot = 0
      abuttableBotBot = 1
}
```

where the variables indicate the following:

`rowSpacingTopTop`	Spacing between the top edges when you are using double-back cell rows in your floorplan Valid values: A floating-point number
`rowSpacingTopBot`	Spacing between the top edge and the bottom edge when you are not using double-back cell rows in your floorplan Valid values: A floating-point number
`rowSpacingBotBot`	Spacing between the bottom edges when you are using double-back cell rows in your floorplan Valid values: A floating-point number
`abuttableTopTop`	Whether or not the two top edges can be abutted when you are using double-back cell rows in your floorplan Valid values: 1 (yes), 0 (no)
`abuttableTopBot`	Whether or not the top edge and the bottom edge can be abutted when you are not using double-back cell rows in your floorplan Valid values: 1 (yes), 0 (no)
`abuttableBotBot`	Whether or not the two bottom edges can be abutted when you are using double-back cell rows in your floorplan Valid values: 1 (yes), 0 (no)

CapModel and CapTable Sections

The CapModel and CapTable sections of a technology file let you specify the material type (conductor or dielectric), thickness, and dielectric value for each design layer, as well as the wire width and spacing for each layer.

You can create these two sections manually with a text editor, but you normally create them by using the `cmCreateCapModel`.

Examples

Example A-18 is an example of the CapModel section.

Example A-18 CapModel Section

```
CapModel "polyConfig1" {
      refLayer = "poly"
      groundPlaneBelow = ""
      groundPlaneAbove = "metal1"
      bottomCapType = "Table"
      bottomCapDataMin = "poly_C_BOTTOM_GP"
      bottomCapDataNom = "poly_C_BOTTOM_GP"
      bottomCapDataMax = "poly_C_BOTTOM_GP"
      topCapType = "Table"
      topCapDataMin = "poly_C_TOP_GP"
      topCapDataNom = "poly_C_TOP_GP"
      topCapDataMax = "poly_C_TOP_GP"
      lateralCapType = "Table"
      lateralCapDataMin = "poly_C_LATERAL"
      lateralCapDataNom = "poly_C_LATERAL"
      lateralCapDataMax = "poly_C_LATERAL"
}
```

Example A-19 is an example of the CapTable section.

Example A-19 CapTable Section

```
CapTable "metal3_C_BOTTOM_GP_21" {
         wireWidthSize = 3
         wireSpacingSize = 5
         wireWidth = (0.6, 1.2, 1.8)
         wireSpacing = (0.5, 1, 1.5, 2, 2.5)
         capValue = (
                     2.4798e-05, 3.05934e-05, 3.5548e-05, 3.99643e-05,
                     4.39498e-05, 3.1404e-05, 3.74482e-05, 4.2877e-05,
                     4.77828e-05, 5.22164e-05, 3.87834e-05, 4.50642e-05,
                     5.07664e-05, 5.59294e-05, 6.05912e-05
                     )
}
```

Example A-20 is an example of a table lookup capacitance model.

Example A-20 Table Lookup Capacitance Model

```
Technology {
      name = ""
      dielectric = 0
}

Stipple "solid" {
      width = 1
      height = 1
      pattern = (1, 1, 1, 1)
}

CapTable "poly_C_TOP_GP" {
      wireWidthSize = 2
      wireSpacingSize = 3
      wireWidth = (0.4, 0.8)
      wireSpacing = (0.5, 1, 1.5)
      capValue = (4.68407e-05, 5.76469e-05,
                  6.10734e-05, 6.97226e-05,
                  8.06076e-05, 8.40257e-05)
}
......
```

ResModel Section

The ResModel section in the technology file allows the resistance and temperature coefficient of the layer to be expressed as a function of wire width.

When the ResModel section for a given layer is present, it overrides the values for the following attributes in the corresponding Layer definition: `unitMinResistance`, `unitNomResistance`, `unitMaxResistance`, and `temperatureCoeff`.

Example

Example A-21 shows a ResModel section.

Example A-21 ResModel Section

```
ResModel "metal1ResModel" {
      layerNumber = 1
      size = 3
      wireWidth = (0.48, 0.6, 0.72)
      tempCoeff = (0.01, 0.01, 0.01)
      minRes = (0.0003, 0.0002, 0.0001)
      nomRes = (0.0003, 0.0002, 0.0001)
      maxRes = (0.0003, 0.0002, 0.0001)
}
```

DensityRule Section

The DensityRule section in a technology file defines the following attributes:

`layerNumber`	The metal layer
`windowSize`	The size of the window for the density check; the check setup is assumed to be half of the window size
	The size of the area for density checking
`minDensity`	The minimum density percentage
`maxDensity`	The maximum density percentage
`maxGradientDensity`	Specifies that the fill density between adjacent windows should not be more than the density gradient specified. When the density gradient information is read in, the fill density of each window is compared with the corresponding density of its neighbors. If the percentage difference between them exceeds the density gradient of that layer, the fill is trimmed to satisfy the density gradient rule.

Example

Example A-22 shows a DensityRule section.

Example A-22 DensityRule Section

```
DensityRule "" {
      layerNumber =
      windowSize =
      minDensity =
      maxDensity =
    maxGradientDensity =
}
```

SlotRule Section

The SlotRule section in a technology file defines the following attributes:

layerNumber	The layer number
lengthThreshold	The length threshold of metal wires requiring slotting
widthThreshold	The width threshold of metal wires requiring slotting

minSlotWidth	The minimum width of the slot
maxSlotWidth	The maximum width of the slot
minSlotLength	The minimum length of the slot
maxSlotLength	The maximum length of the slot
minSideSpace	The minimum space between adjacent slots in a direction perpendicular to the wire (current flow) direction
maxSideSpace	The maximum space between adjacent slots in a direction perpendicular to the wire (current flow) direction
minSideClearance	The minimum space from the side edge of a wire to its outermost slot
maxSideClearance	The maximum space from the side edge of a wire to its outermost slot
minEndSpace	The minimum space between adjacent slots in the wire (current flow) direction
maxEndSpace	The maximum space between adjacent slots in the wire (current flow) direction
minEndClearance	The minimum space from the end edge of a wire to its outermost slot
maxEndClearance	The maximum space from the end edge of a wire to its outermost slot
maxMetalDensity	The maximum metal density allowed for a slotted wire

Example

Example A-23 shows a SlotRule section.

Example A-23 SlotRule Section

```
SlotRule "" {
      layerNumber =
      lengthThreshold =
      widthThreshold =
      minSlotWidth =
      maxSlotWidth =
      minSlotLength =
      maxSlotLength =
      minSideSpace =
```

```
      maxSideSpace =
      minSideClearance =
      maxSideClearance =
      minEndSpace =
      maxEndSpace =
      minEndClearance =
      maxEndClearance =
      maxMetalDensity =
}
```

Deep Submicron Design Rule Support

For information about routing design rules for advanced technologies, as defined with technology file syntax, see the *Astro User Guide* and the *IC Compiler User Guide*.

B

Standard and Macro Cell Design

This appendix provides general information for designing two cell types, in the following sections:

- Designing Standard Cells
- Cell and Grid Size
- Design Spacing Rules
- Over-the-Cell Routing
- Metal Pins
- Designing Macro Cells

Designing Standard Cells

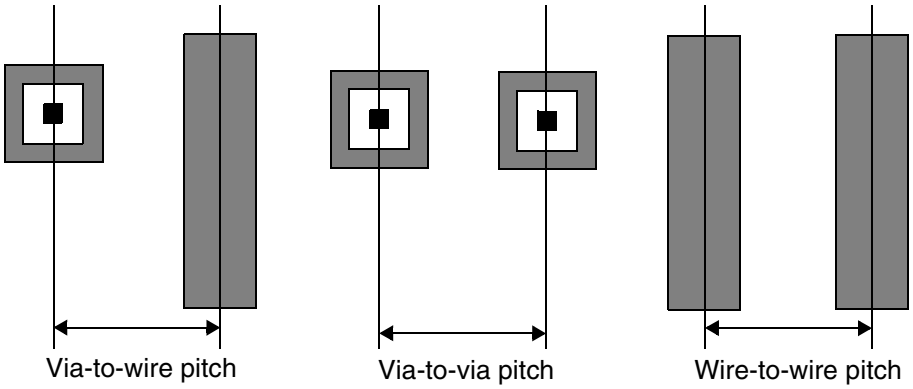
Because minimizing routing area is more important to achieving a small die size than minimizing the size of the cells, it is best to design cells so that they are best suited for the place and route tools you use. This appendix provides tips for designing a standard cell library for use with Astro.

The following notation is used in this document:

Notation	Indicates
M1	Metal1
M2	Metal2
M3	Metal3
V1	Metal1-metal2 via
V2	Metal2-metal3 via
PCON	Poly-to-metal1 contact

Unless otherwise noted, all spacing rules and recommendations are given as pitch (center-to-center spacing), as shown in Figure B-1.

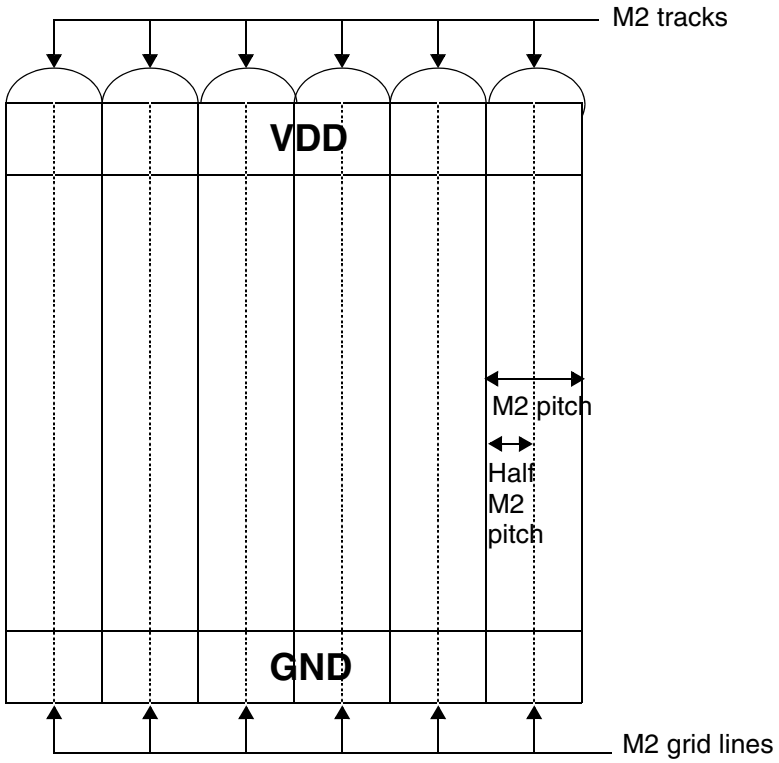
Figure B-1 Pitch (Center-to-Center Spacing) Examples



Cell and Grid Size

Make cells a multiple of half M2 pitch (strongly recommended).

Figure B-2 Half M2 Pitch Example



Spacing of the grid lines must be at least via-to-wire (mandatory), to make the most of the routing space.

Note:

If you want to create a cell that contain pins with both top and bottom access, the cell width must be greater than or equal to the following:

$$numPins \times m2Pitch$$

where *numPins* is the number of pins in the cell and *m2Pitch* is the M2 pitch.

Design Spacing Rules

Do not set design spacing rules for the following:

- PCON-to-M2
- PCON-to-M3

- V2-to-Poly
- V1 to M3
- V2 to M1

Over-the-Cell Routing

Because over-the-cell routing is free, Astro routes over the cell (on metal2 and metal3) as much as possible. Therefore, you should

- Minimize or eliminate M2 and M3 inside cells.
- Remove all M2 feedthroughs from cells.
- Minimize the pin size. Extra pin pieces become blockages and introduce redundant capacitance.
- Avoid M2 pins if possible. Let the router drop V1s.

Metal Pins

Access

In Astro, the router drops vias from metal3 to metal2 and metal1 pins. Leave free space around the pins, so the router can access them easily.

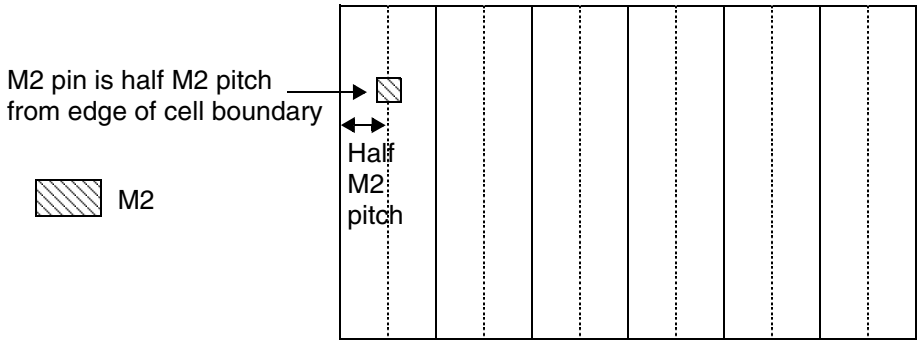
Spacing

The recommended spacing for M2 pins is the greater of V1-to-V1 and M2-to-V2. Minimally, you should space M2 pins at the greater of M2-to-V1 and M2-to-V2.

Astro performs a majority of the routing along the grid lines defined by the M1, M2, and M3 pitches. Therefore,

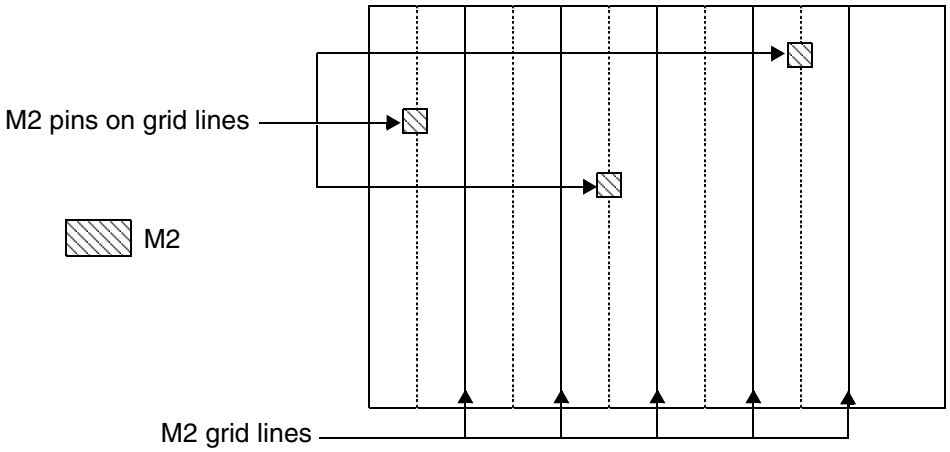
- You should place M2 pins at a distance of at least half the M2 pitch from the cell boundary, as shown in Figure B-3.

Figure B-3 Recommended M2 Pin Spacing



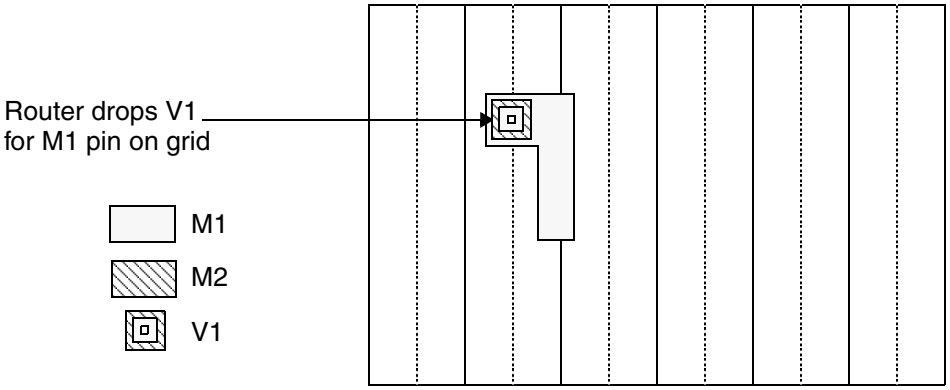
- You should place M2 pins on M2 grid lines whenever possible, as shown in Figure B-4.

Figure B-4 Recommended M2 Pin Placement



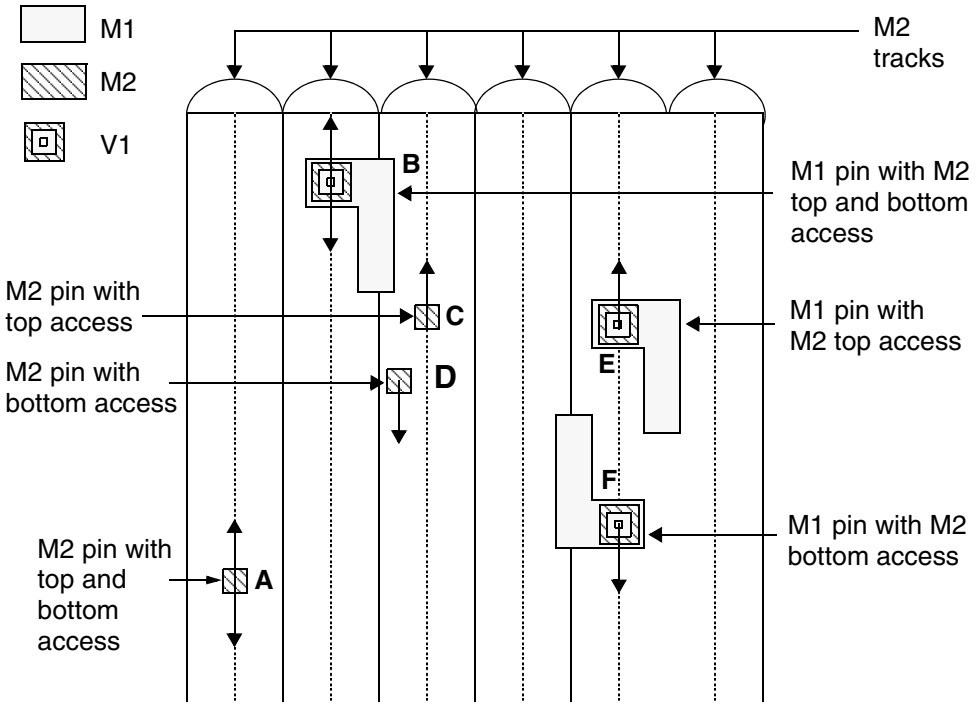
- You should place M1 pins so that the router can drop V1s on M2 grid lines, as shown in Figure B-5.

Figure B-5 Recommended M1 Pin Placement



- You should place no more than one pin with M2 top access and no more than one pin with M2 bottom access in any M2 track, as shown in Figure B-6.

Figure B-6 Recommend M2 Pin Top and Bottom Access Placement



In Figure B-6, pins A and B have no top or bottom blockages. However, the other pins are blocked as follows:

- C is blocked on the bottom by D.
- D is blocked on the top by C.
- E is blocked on the bottom by F.
- F is blocked on the top by E.

Horizontal Via Grid

If you set the M2 pitch at less than V1-to-V1 and greater than or equal to the greater of V1-to-M2 and V2-to-M2, you should place the M1 pins so that the router can place the vias on horizontal grid lines. A typical spacing for the grid lines is M1 or M3 pitch.

The following figures show two of the possible via solutions for pin placement.

Figure B-7 Typical Grid Line Spacing

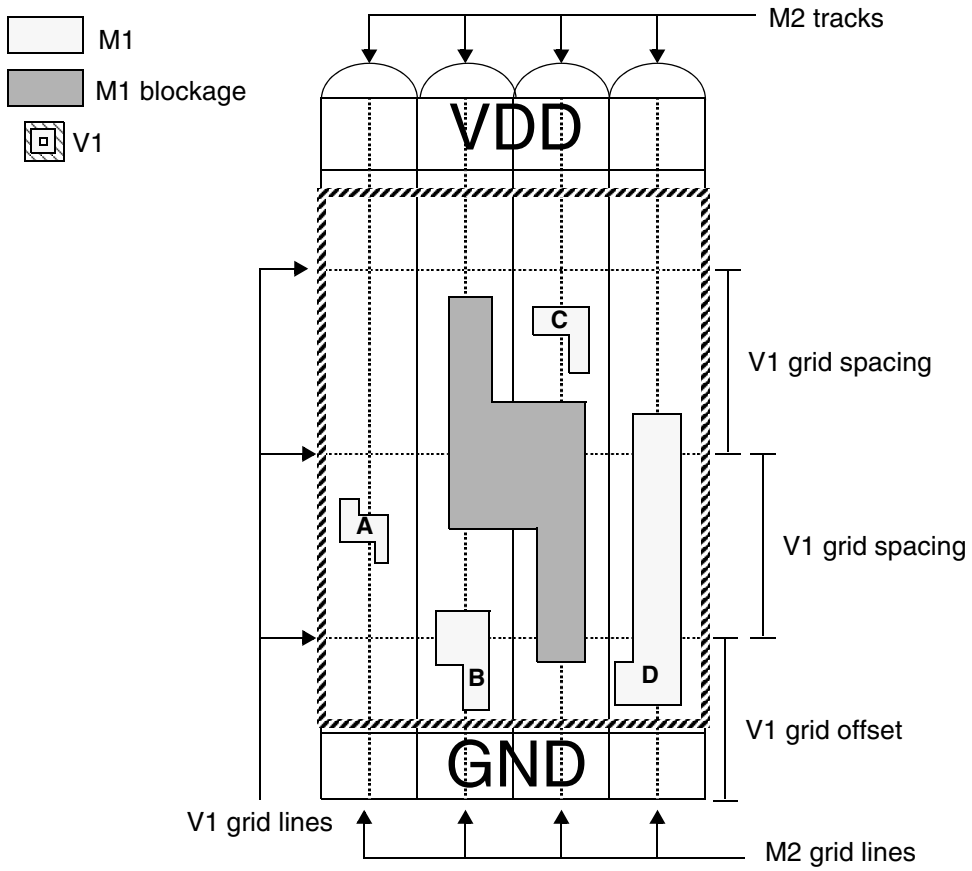
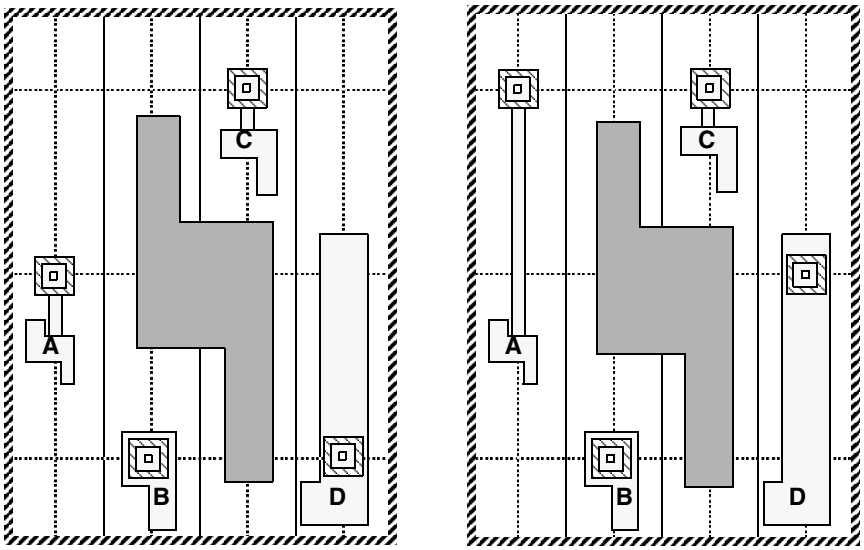


Figure B-8 Two Possible Via Solutions for Pin Placement



Via solution 1 **Via solution 2**

If the M2 pitch is less than V1-to-V1, you should place the M1 pins so that the router can place vias for the leftmost and rightmost pins in abutting cells on different horizontal grid lines. Otherwise, the router must insert feedthroughs to avoid violating design spacing rules between the vias.

Figure B-9 Via Solution for Pin Placement 1

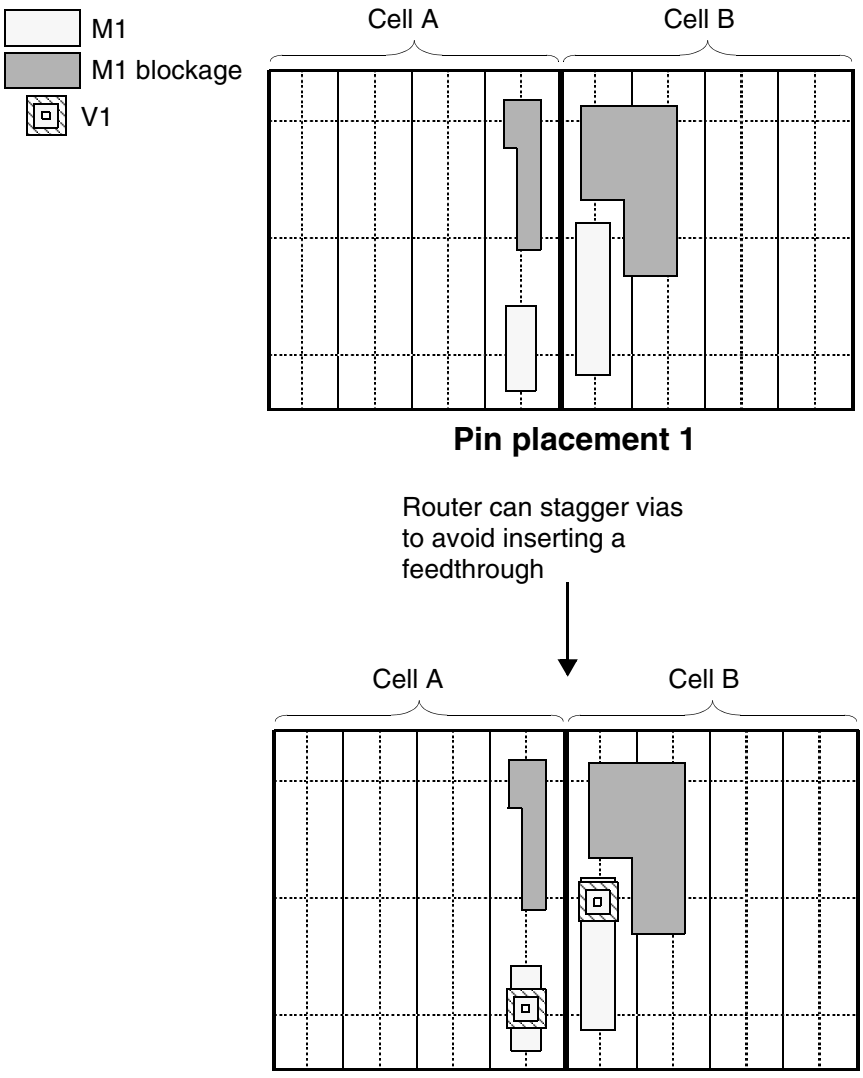
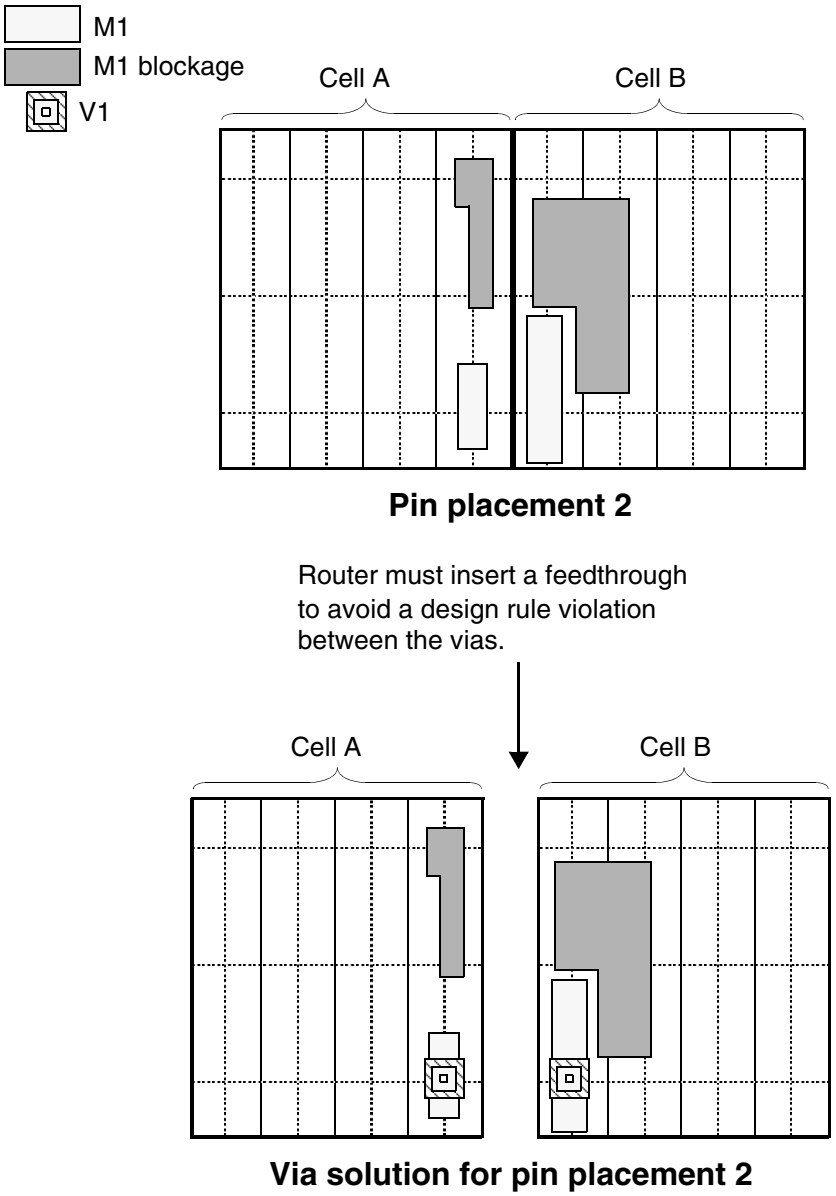
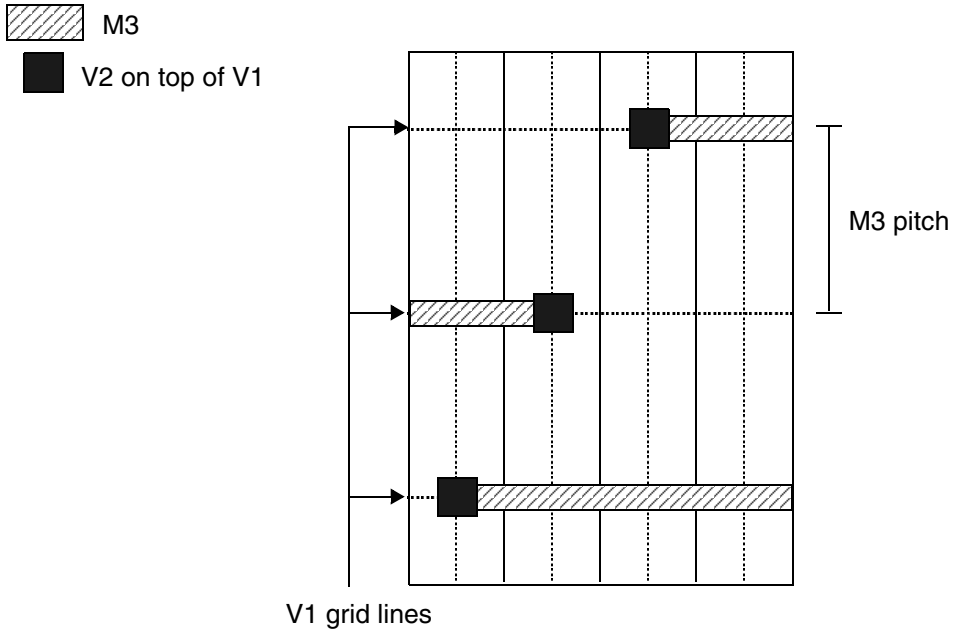


Figure B-10 Via Solution for Pin Placement 2



If your technology allows via stacking, you should set the spacing for the horizontal V1 grid to M3 pitch, thus allowing the router to place V2s on top of V1s without the addition of M2 wire, as shown in Figure B-11.

Figure B-11 Via Stacking

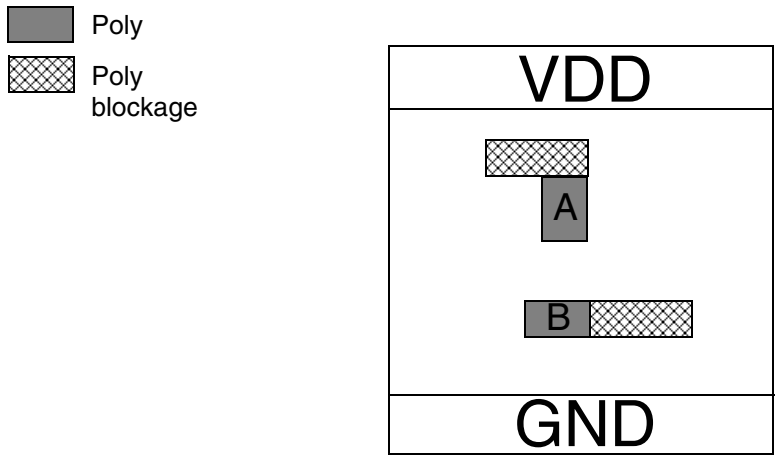


Poly Pins

The following section covers accessing, converting, and spacing poly pins.

Access

Poly pins must be accessible directly from the top or bottom (mandatory). The following cell is not acceptable, because poly pin A cannot be directly accessed from either the top or the bottom.

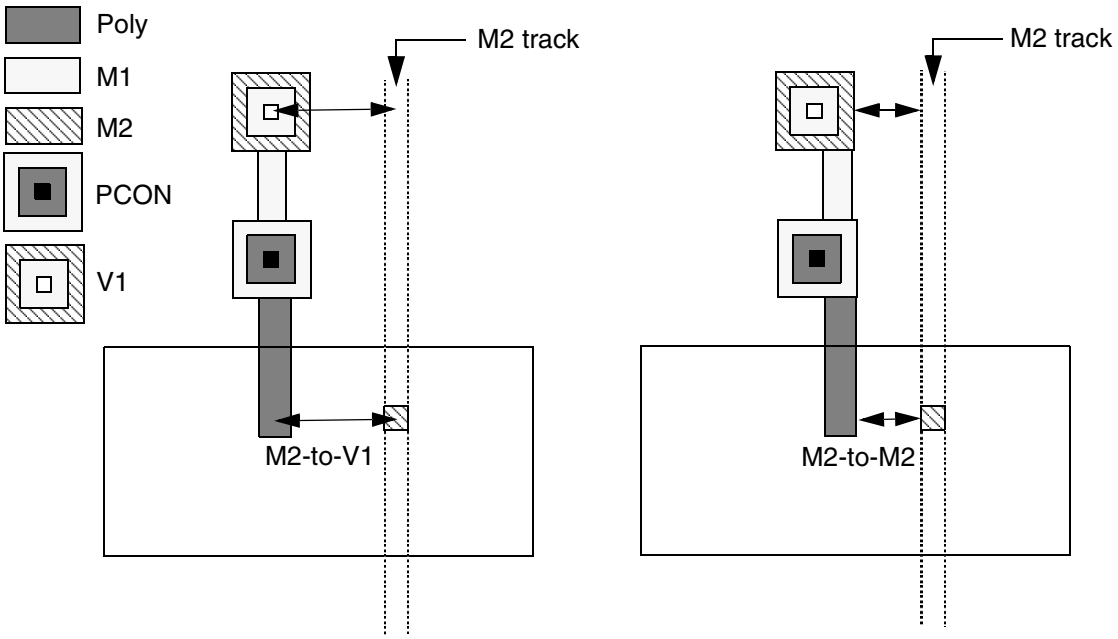


If you design cells so that all (or most) of the poly pins can be accessed from the same direction, Astro can further reduce the number of channels (and therefore reduce the routing area) by placing cell rows back to back.

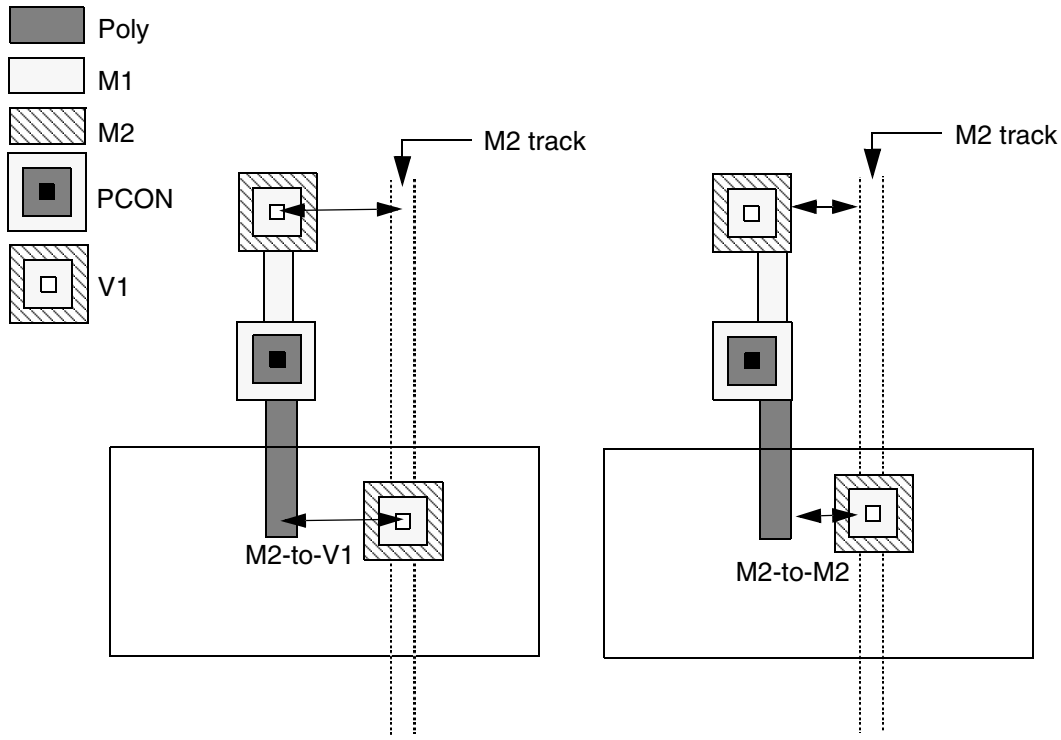
Converting to M2

If possible, you should design cells to allow the router to easily convert poly pins to M2 without causing any constraints on the neighboring pins. This is especially important for 3.5-layer (three metal layers plus poly) cell design, because the router has to raise the pins to M2 before it can connect them to M3.

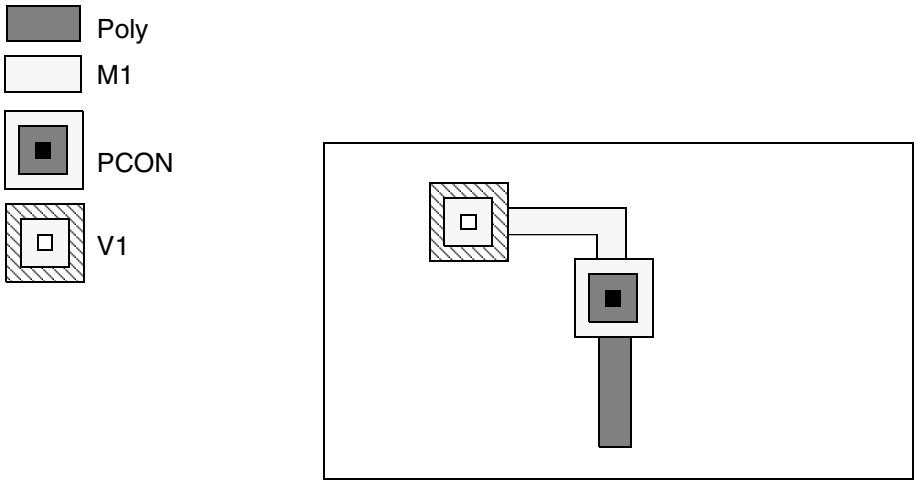
Place poly pins at least M2-to-V1or M2-to-M2 away from M2 pins.



Place poly and M1 pins so that when the router drops a V1 onto the M1 pin and converts the poly pin to an M2 pin, the edge-to-edge distance between the two M2 pins is at least the minimum design rule spacing for M2 pins.



If possible, place poly and M1 pins so that the router can convert the poly pins to M2 inside the cell.

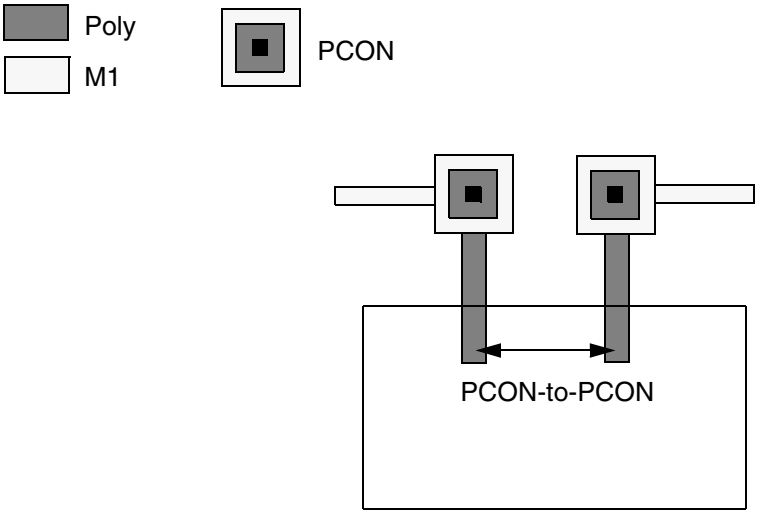


Furthermore, because poly has a high RC (resistance times capacitance) value, the router assumes that only input pins are on poly.

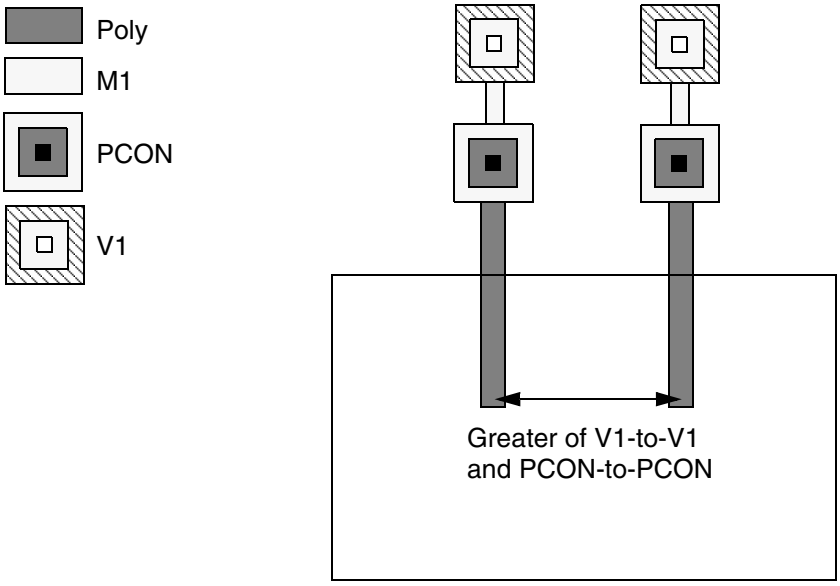
Spacing to Poly Pins

The recommended spacing for poly pins is the greater of PCON-to-PCON and V1-to-V1. Minimally, you should space poly pins poly-to-PCON or V1-to-M2.

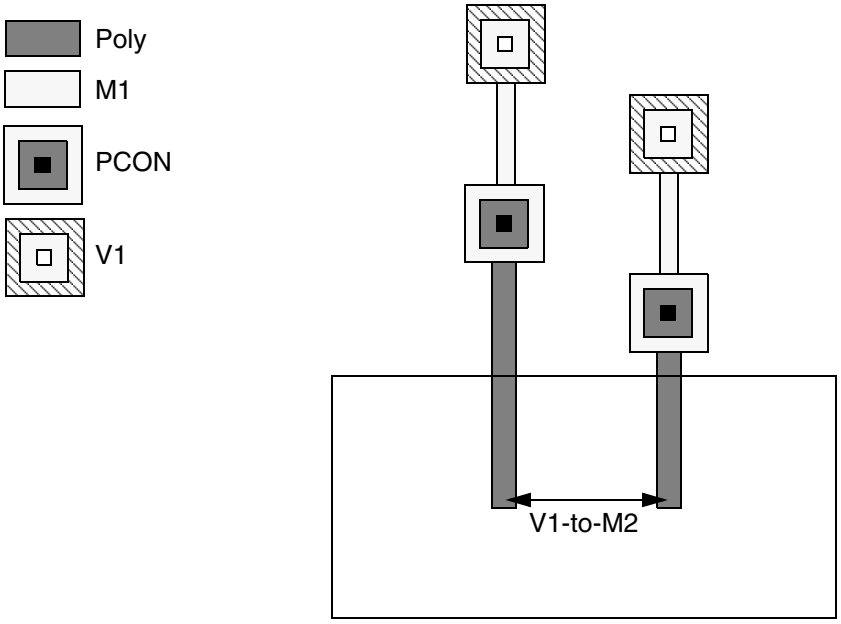
If you space poly pins at least PCON-to-PCON, Astro can reduce the routing area by using the same track for routing M1, as shown below.



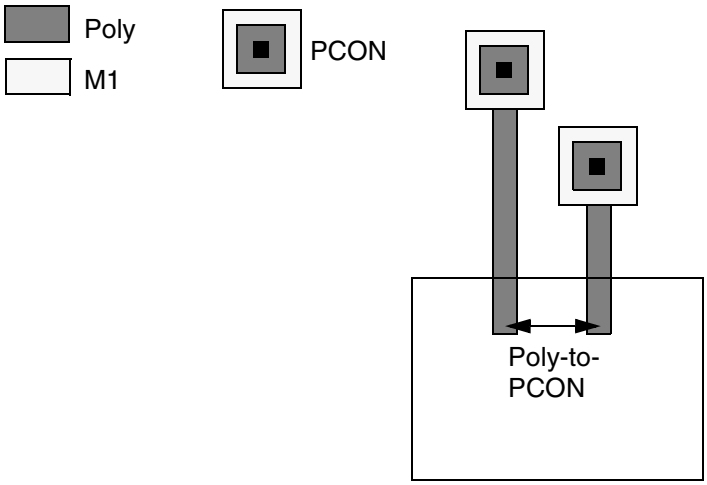
The more poly pins the Astro router can convert to M2 (and thus connect to M3), the more over-the-cell area it can use. Therefore, it is best to space poly pins at least at the greater of V1-to-V1 and PCON-to-PCON.



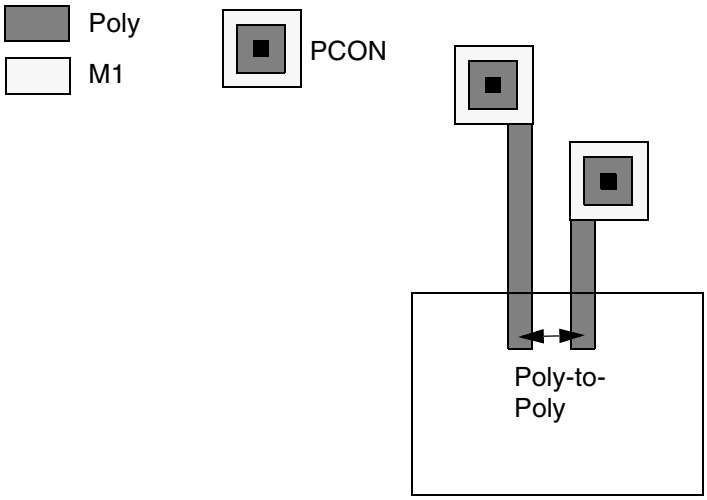
If poly-to-PCON is less than M2-to-V1, you should space poly pins at least V1-to-M2.



If poly-to-PCON is greater than V1-to-M2, you should space poly pins at least poly-to-PCON.



If you space poly pins closer than poly-to-PCON, you need to allow surrounding space for the situations shown in the following figure:



Designing Macro Cells

This section provides tips for designing macro cells for placement in Astro and AstroGA designs.

Shape

Astro lets you create rectilinear macros, but some operations do not work with them.

metal3 Usage

Because Astro can route over blocks, you should avoid metal3 usage inside the blocks. Otherwise, you limit the metal3 paths available for over-the-block routing.

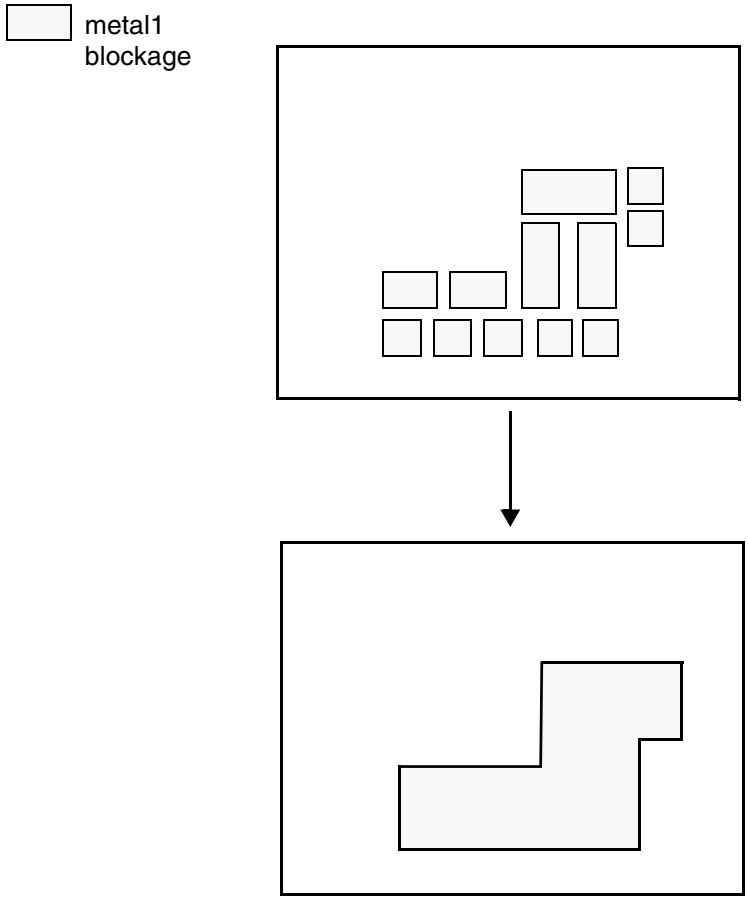
Power and Ground Routing

It is not necessary for you to add power and ground routing rings inside macro blocks. Astro can route power and ground outside the blocks when they are placed in the chip.

Blockage

Merging clusters of small blockage areas into larger blockage areas, as shown in Figure B-12, saves considerable CPU time during routing. Although pin and blockage extraction can merge blockage areas for you, you will save processing time during extraction if you design your blocks with larger blockage areas.

Figure B-12 Merged Blocks



C

Cell Library Format File

This appendix provides instructions for creating and loading a CLF file and provides CLF function descriptions for cell definition functions used by Astro. It contains the following sections:

General Information

This section explains the conventions used in this chapter, describes the general format of a CLF file, and explains how to create and load a CLF file.

Conventions

The following conventions are used in the examples and syntax descriptions in this appendix:

- Functions appear in text font and must be typed exactly as they are given.
- Variables appear in code font. When the value of a variable is a string, the string must be enclosed in quotation marks.
- Examples appear in code font and use the following units (which would be specified in the technology file):
  - micron—distance
  - ns—time
  - pF (picofarad)—capacitance
  - ohm—resistance
  - nH (nanohenry)—inductance
  - mw (milliwatt)—power

General Format

Before you create the layout cells (translate GDSII Stream), you need to load the following:

- `definePad`, which defines the orientation of pad cells

After you create the pin/blockage cells (extraction), you need to load the following for clock tree synthesis:

- `definePortCapacitance`, which defines the actual capacitance of a port
- `definePortPhaseDelay`, which defines the phase delays for macro cells

  Note:

  The `defineMinClkPulseWidth` function sets the minimum pulse width (high and low) for a clock input pin of a flip-flop.

You can add a comment to the CLF file by placing a semicolon (;) at the beginning of the line. All text between the semicolon and the end of the line is ignored.

Creating and Loading a CLF File

Using the syntax information in this appendix, you can create a CLF file by using a UNIX text editor. You can also create a CLF/port information file by translating a Synopsys timing file.

This section contains procedures for

- Loading a CLF file
- Writing CLF information to a file

Use Milkyway to perform the procedures in this section. From the directory containing your cell libraries, type the following:

```
Milkyway
```

Milkyway displays its window. From here you can access the commands referenced in these procedures.

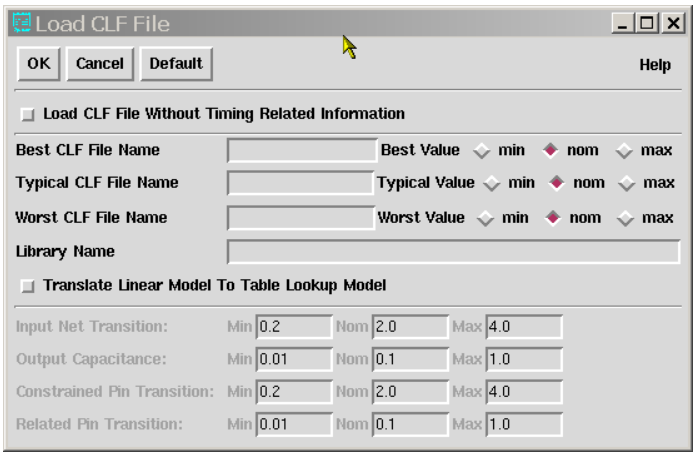
Loading a CLF File

After you create your CLF file, you load it by using CLF > Load, as described.

To load a CLF file,

1. Enter `auLoadCLF` or choose CLF > Load.

   The Load CLF File dialog box appears.



2. Enter the CLF file name.

   This is the name of the CLF file you created.

3. Enter the library name.

   This is the name of the library to which you want to load the CLF file.

4. Decide whether you want to select the Translate Linear Model To Table Lookup Model option.

   If your CLF file specifies delay information by using the linear delay model, you can select this option to translate it to the table lookup delay model.

5. Click OK.

   The CLF file information is added to the library information file.

Writing CLF Information to a File

To write CLF information to a file, use CLF > Write To File. For more information, see `auDefIn` in Physical Implementation Online Help.

Cell Definition Functions

The purpose of a CLF file is to provide timing information for timing-driven place and route. The CLF file provides Astro with information about cells in the library. The Synopsys application uses this information during

- Pin and blockage extraction
- Pad placement
- Clock tree synthesis

This section provides syntax information for functions used in a CLF file for Astro. These functions are arranged alphabetically.

defineMinClkPulseWidth

The `defineMinClkPulseWidth` function sets minimum pulse widths (high and low) for a clock input pin of a clocked cell.

Syntax

```
defineMinClkPulseWidth cellName portName High Low
```

where the arguments are as follows:

`cellName`

Specifies the name of the cell containing the port for which you are specifying pulse widths

Valid values: Name of any clocked cell in the library

`portName`

Specifies the name of the port for which you are specifying pulse widths

Valid values: Name of any port of type clock on the clocked cell specified by `cellName` For port type information, see “Specifying Port Information for Advanced Operations” on page 7-11.

`High`

Specifies the minimum time the clock pulse must be high

Valid values: Any nonnegative floating-point number

`Low`

Specifies the minimum time the clock pulse must be low

Valid values: Any nonnegative floating-point number

Example 1

```
defineMinClkPulseWidth "reg" "cp" 2 3
```

The `defineMinClkPulseWidth` function indicates the following:

- The clock pulse at port `cp` of cell `reg` must be high for at least 2 ns.
- The clock pulse at port `cp` of cell `reg` must be low for at least 3 ns.

definePad

The `definePad` function defines the orientation of pad cells. This information is for placing pads around the chip boundary.

Syntax

`definePad` *`padName padRotation`*

where the arguments are as follows:

`padName`

Specifies the name of the pad cell to be defined

Valid values: Name of any pad cell in the library, or an asterisk (*) to indicate all pad cells in the library

`padRotation`

Specifies the rotation necessary to place the pad on the left side (or on the corner pad in the upper-left corner) of the boundary

Valid values: Any of the following:

0 – no rotation is necessary

90 – a 90-degree counterclockwise rotation is necessary

180 – a 180-degree counterclockwise rotation is necessary

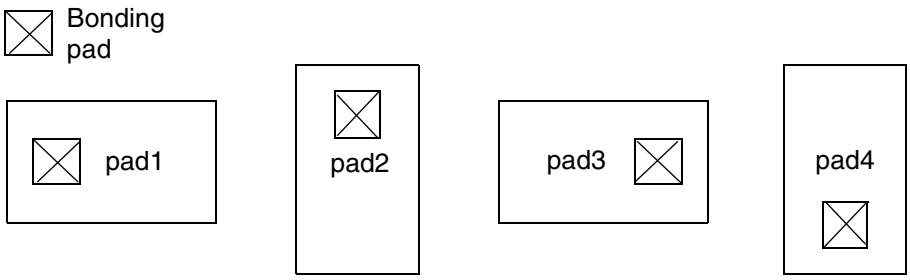
270 – a 270-degree counterclockwise rotation is necessary

Example 1

```
definePad "pad1" "0"
definePad "pad2" "90"
definePad "pad3" "180"
definePad "pad4" "270"
```

These `definePad` functions indicate the rotation necessary for placing the following pads on the left side of the boundary, as shown in Figure C-1.

Figure C-1 Left Side of Boundary Placement



Example 2

```
definePad "*" "0"
```

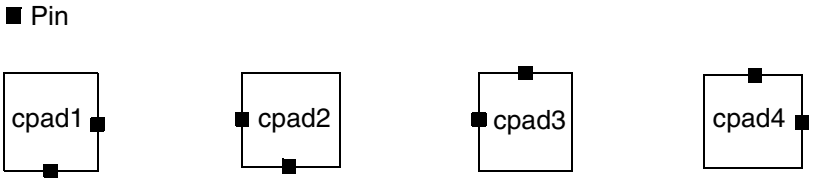
This `definePad` function indicates that no rotation is necessary for placing any pads in the library on the left side of the boundary.

Example 3

```
definePad "cpad1" "0"
definePad "cpad2" "90"
definePad "cpad3"
"180"
definePad "cpad4" "270"
```

These `definePad` functions indicate the rotation necessary for placing the following corner pads in the upper-left corner of the boundary, as shown in Figure C-2.

Figure C-2 Upper-Left Corner of Boundary Placement



D

Top Design Format File

This appendix provides instructions for creating and loading a Top Design Format (TDF) file and contains function descriptions for the TDF functions. It contains the following sections:

General Information

This section explains the conventions used in this appendix, describes the general format of a TDF file, and explains how to create and apply a TDF file.

Conventions

The following conventions are used in the examples and syntax descriptions in this appendix:

- Functions and keywords must be typed exactly as they are given.
- Variables appear in code font. When the value of a variable is a string, the string must be enclosed with quotation marks.
- Optional arguments appear inside square brackets ([ ]).
- Parentheses (( )) should always be typed as they appear in the syntax.
- Examples appear in code font and use the following units (which would be specified in the technology file):
  - micron—distance
  - ns—time
  - pF (picofarad)—capacitance
  - ohm—resistance
  - nH (nanohenry)—inductance
  - mw (milliwatt)—power

General Format

Synopsys provides more than 20 TDF functions for creating and updating TDF files. For a detailed description of those functions, see Physical Implementation Online Help.

In addition to the TDF functions, you can include database functions (functions with the prefix `db`) in a TDF file. When including database functions that require a cell ID, you can substitute the following for the cell ID:

```
tdfGetTDFCellId
```

Note:
You can add comments to the TDF file by placing a semicolon (;) at the beginning of a comment line. All text between the semicolon and the end of the line is ignored.

Creating TDF Files

You can create TDF files by using any of the following:

- Any UNIX text editor
- The `tdfDump` command

Using a UNIX Text Editor

You can use the UNIX text editor you generally use on your workstation to create a TDF file. See "Common Functions" on page D-4 for syntax information.

Writing Manual Placement to a TDF File

You can move a module cell by using Edit > Snap To Tile, Edit > Move, or Edit > Transform. After moving it, you can constrain it to its current location by using Constraints > Fix Cell. You can then write this constraint to a TDF file by typing the following command in the input area of the application window:

```
tdfDump (geGetEditCell) fileName
```

where *fileName* is the name you want to give to the TDF file.

For example,

```
tdfDump (geGetEditCell) "fp.tdf"
```

writes the current TDF information, including location constraints for the moved and fixed cell, to a file called fp.tdf.

Applying TDF Directives to a Design

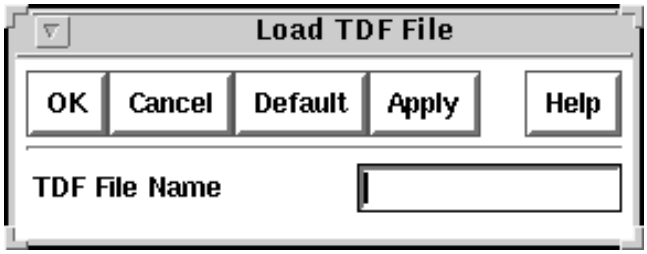
As you proceed through the design flow, you can update the TDF information for the design.

To load a TDF file,

1. Using Astro, choose Design Setup > TDF-Load TDF (`axgLoadTDF`). The Load TDF File dialog box appears.

Note:

When loading TDF file information from Jupiter, choose Timing Setup > TDF-LoadTDF (`ioLoadTDF`).



2. Type the TDF file name for the file containing the TDF information you want to load.
3. Click OK.

Common Functions

This section contains syntax information for functions used in the TDF file. The functions are arranged alphabetically.

netWeight

The `netWeight` function sets the net priorities (or weights) for placing cells. During placement, the Milkyway-based tool attempts to place cells connected to high-priority nets closer together.

The primary goal of your Synopsys application, regardless of the specified net weights, is to achieve the minimum total net length for the design. Therefore, the Synopsys application weighs any reduction in the length of a high-priority net against the effect on the overall net length.

Nets with no priority set have the lowest priority (net weight of 1).

You can set net weights while running a Synopsys application by using Constraints > Set Net Weight.

Syntax

```
netWeight netName netWtHorizontal netWtVertical
```

where the arguments are as follows:

netName	Name of the net to be prioritized. Valid values: Name of any net in the design
netWtHorizontal	Weight to be given to horizontal wires on the net. The higher the value, the greater priority the Synopsys application gives the net. Valid values: Any positive integer
netWtVertical	Weight to be given to vertical wires on the net. The higher the value, the greater priority the Synopsys application gives the net. Valid values: Any positive integer

Example

```
netWeight "CTL2" 10 8
netWeight "CTL3" 4 2
netWeight "CTL4" 6 6
```

Using these net priority directives, the Synopsys application gives the highest priority to horizontal wires on the CTL2 net and the lowest priority to vertical wires on the CTL3 net (except for nets not specified, which receive the lowest priority). For nets CTL2 and CTL3, horizontal wires are given higher priority than vertical wires. For net CTL4, horizontal and vertical wires are given equal priority.

tdfPurgeCellConstr

The `tdfPurgeCellConstr` function instructs the Synopsys application to delete all cell location constraints created previously by locate functions, `aprFixPlacement` functions, or Edit > Location - Fix.

Syntax

```
tdfPurgeCellConstr
```

tdfPurgeNetConstr

The `tdfPurgeNetConstr` function instructs the Synopsys application to delete all net weights previously set by `netWeight` functions.

Syntax

```
tdfPurgeNetConstr
```

Astro Functions

This section contains syntax information for Astro functions used in the TDF file.

insertPad

The TDF function `insertPad` adds a pad to the design. To add a pad to a net not defined in the netlist, you must create the net by using PreRoute > Connect Ports to P/G (see "Specifying Global Net Connections" on page 9-2).

The `insertPad` function is useful for adding extra power and ground pads around the chip boundary.

Syntax

`insertPad` *`netName padCellName padName connectPin`*

where the arguments are as follows:

`netName`	Name of the net to which the pad should be connected Valid values: Name of any net in the design
`padCellName`	Name of the pad cell master to be used for the inserted pad Valid values: Name of any pad cell in the library or reference library
`padName`	Name to be assigned to the inserted pad Valid values: Any string of up to 32 characters
`connectPin`	Name of the pin inside the pad to be connected to the macros and cell regions in the core Valid values: Name of any pin in the pad cell master

Example

```
insertPad "VDD" "m1pad" "vdd1" "PAD"

insertPad "GND" "m1pad" "gnd1" "PAD"
```

Using these pad insertion directives, the Synopsys application adds two pads to the design:

- vdd1 is an instance of the pad cell master m1pad and is connected to the VDD net through its pin named PAD.
- gnd1 is an instance of the pad cell master m1pad and is connected to the GND net through its pin named PAD.

pad

The `pad` function lets you set the following constraints for placement of a pad:

- Side (or corner) of the chip for placing the pad
- Order relative to other pads on the same side
- Offset from the bottom or left edge of the boundary

To set constraints on a pad not defined in the netlist, you must add the pad by using the `insertPad` function before setting the constraints.

Syntax

```
pad padName padSide [padOrder] [padOffset] ["reflect"]
```

where the arguments are as follows:

padName	Name of the pad to be constrained. Valid values: Name of any pad in the design, including any pad added with an `insertPad` function The name of a pad consists of the names of all cell instances in the netlist hierarchy, top to bottom, preceded by slashes and followed by a slash and the name of the pad. For example, a cell instance A contains a cell instance B, which contains a pad P1. The hierarchical name of the pad P1 is /A/B/P1.
padSide	Side or corner of the chip on which the pad should be placed. Valid values: Any of the following strings: • right • left • top • bottom • none Note: To eliminate a previously loaded side constraint for a pad, specify a padSide of none.

The following table shows how to specify the placement for a corner pad.

Pad location	Specification
Lower-left corner	Bottom
Lower-right corner	Right
Upper-right corner	Top
Upper-left corner	Left

`padOrder`

Order of the pad placement relative to the other pads on the same side.

Valid values: Any integer from 0 to 32,767

Pads with lower `padOrder` values are placed closer to the bottom or left edge of the boundary than pads with higher values. Otherwise, `padOrder` values have the following effects:

- If a pad has a `padOrder` value of 0, the placement of the pad relative to the other pads is not important.
- If a pad has a `padOrder` value of 1, no pad (except a corner pad) can be placed between the pad and the corner of the boundary.
- If a pad has a `padOrder` value of 2, any number of pads can be placed between the pad and the corner of the boundary.
- If two pads have consecutive `padOrder` values, no pad can be placed between them.

`padOffset`

Relative offset of the pad placement from the bottom edge or left edge of the cell boundary.

If the pads in a group have the same `padOrder` value, no other pads can be placed between any of the pads in the group. In this case, the ordering inside the group is not important. However, the placement of the group relative to the other pads on the same side is determined by the `padOrder` value.

Valid values: Either of the following:

- Any number (in distance units) specifying the distance within the cell boundary that includes the pad width
- A negative number, indicating that you do not want to place an offset constraint on the pad

In the case of contradictions between constraints,

- If you specify a `padOffset` value that contradicts the specified pad order, the Synopsys application will relax the `padOffset` constraint.
- If you specify two `padOffset` values that contradict each other, the Synopsys application will give priority to the `padOffset` constraint on the pad with the lowest `padOrder` value.

Note: To eliminate a previously loaded offset constraint for a pad, specify a negative number for `padOffset`.

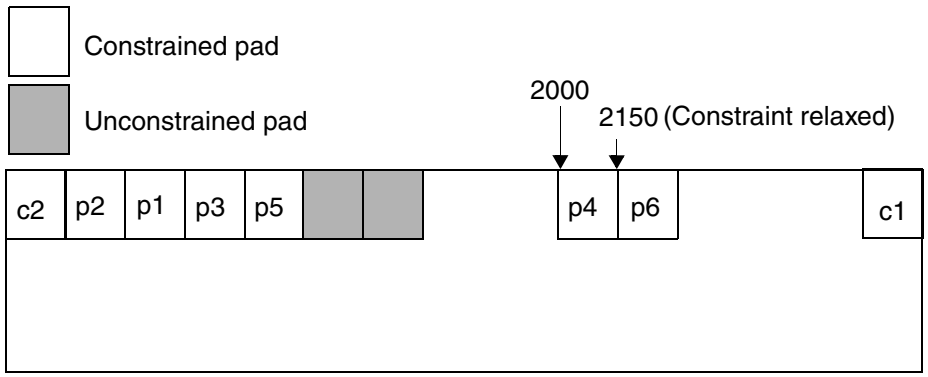
`"reflect"` Keyword for indicating that the pad should be flipped. If you do not want to flip the pad, omit this keyword.

Example 1

```
pad "p2" "top" 1
pad "p1" "top" 2
pad "p3" "top" 3
pad "p4" "top" 5 2000
pad "p5" "top"
pad "p6" "top" 7 1000
pad "c1" "top"
pad "c2" "left"
```

Note:
Pads c1 and c2 are corner pads.

Using these pad location directives, the tool based on Milkyway places pads on the top side of the chip.

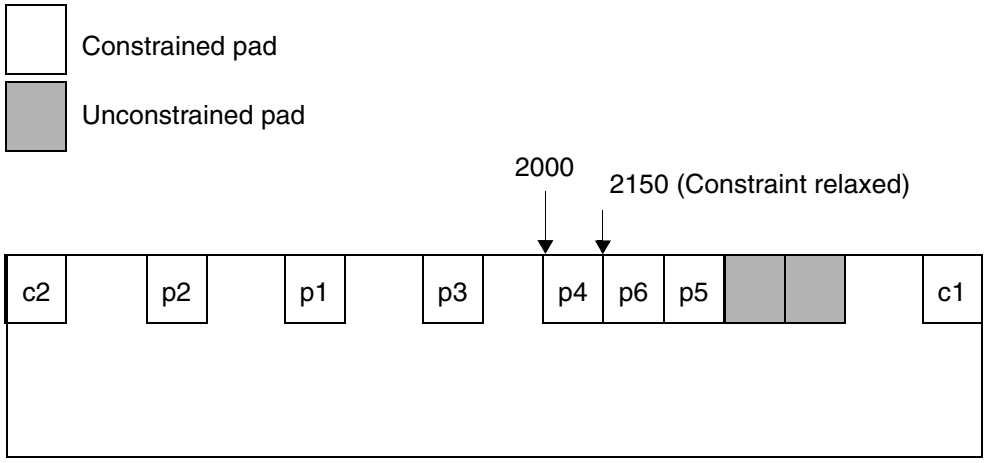


Example 2

```
pad "p2" "top" 2
pad "p1" "top" 3
pad "p3" "top" 4
pad "p4" "top" 5 2000
pad "p5" "top"
pad "p6" "top" 8 1000
pad "c1" "top"
pad "c2" "left"
```

Note:
Pads c1 and c2 are corner pads.

Using these pad location directives, the tool based on Milkyway places pads on the top side of the chip.

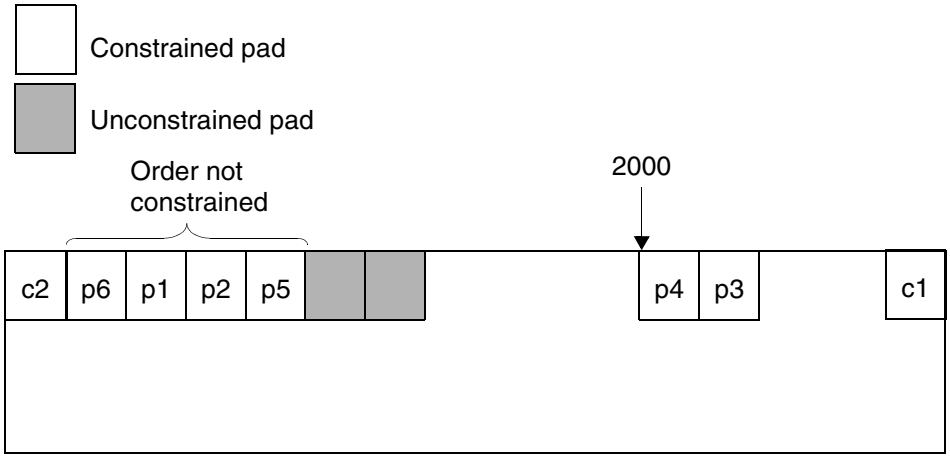


Example 3

```
pad "p6" "top" 2
pad "p1" "top" 2
pad "p2" "top" 2
pad "p4" "top" 4 2000
pad "p3" "top" 5
pad "p5" "top"
pad "c1" "top"
pad "c2" "left"
```

Note:
Pads c1 and c2 are corner pads.

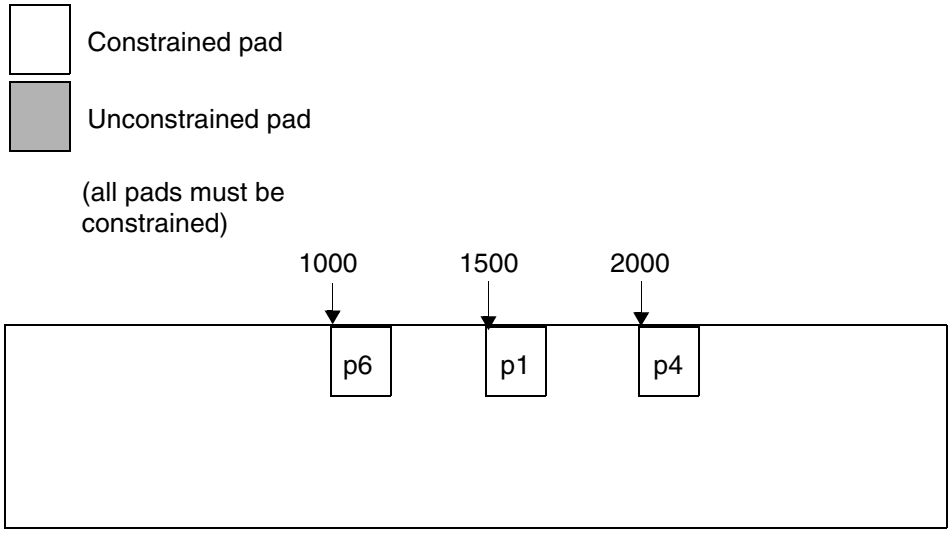
Using these pad location directives, the tool based on Milkyway places pads on the top side of the chip.



Example 4

```
pad "p6" "top" 1 1000
pad "p1" "top" 0 1500
pad "p4" "top" 0 2000
```

Using these pad location directives, the tool based on Milkyway places pads on the top side of the chip.



pin

The `pin` function lets you set the following constraints for placement of a pin:

- Layer
- Dimensions
- Side of the block for placing the pin
- Order relative to other pins on the same side
- Offset from the bottom or left edge of the boundary

To set constraints on a pin not defined in the netlist, you must add the pin by using Enter > Pin before setting the constraints.

Note:
To set pin constraints for rectilinear designs, use `dbSetPinLocation` (see Physical Implementation Online Help).

Syntax

`pin` *`pinName layer width height pinSide`* [*`pinOrder`*] [*`pinOffset`*]

where the arguments are as follows:

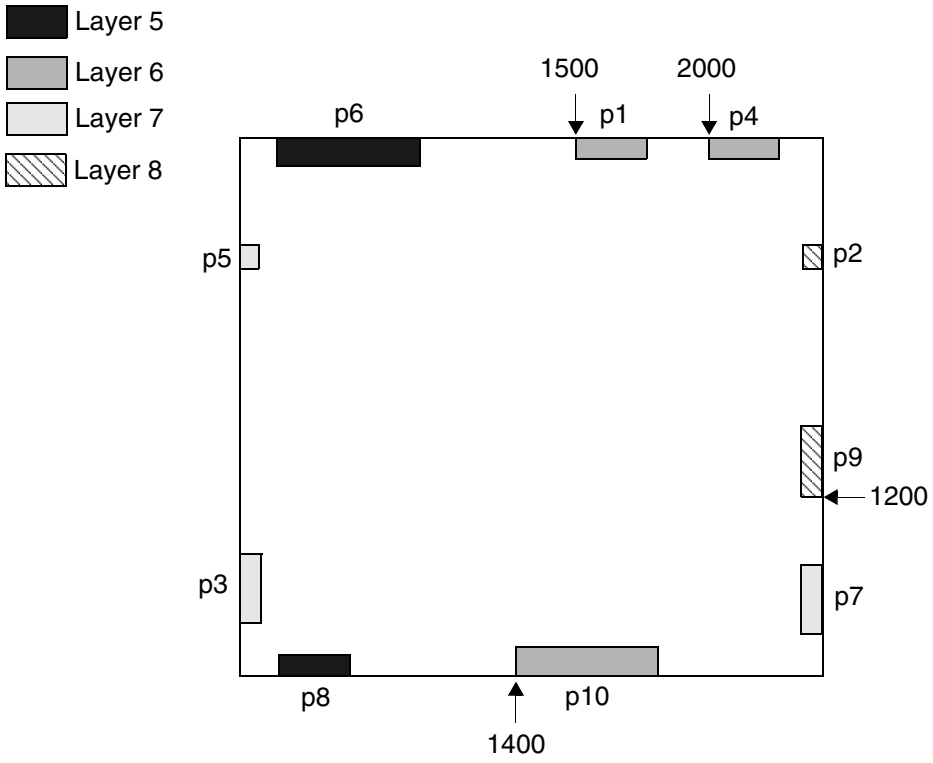
`pinName`	Name of the pin to be constrained Valid values: Name of any pin in the design
`layer`	Name or number of the layer on which you want the pin Valid values: Name or number of any layer defined in the technology file or –1 to indicate that any layer can be used
`width`	Dimension of the pin along the cell boundary Valid values: Any positive floating-point number, or 0 to indicate the minimum possible width
`height`	Dimension of the pin from the boundary to the center of the cell Valid values: Any positive floating-point number, or 0 to indicate the minimum possible height Note: All but those designers very experienced with the tool should use a height equal to the width.

pinSide	Side of the block on which the pin should be placed Valid values: • right • left • top • bottom • none Note: To eliminate a previously loaded side constraint for a pin, enter none for *pinSide*.
pinOrder	Order of the pin placement relative to the other pins on the same side Valid values: Any integer from 0 to 32,767 Pins with lower *pinOrder* values are placed closer to the bottom or left edge of the boundary than pins with higher values. Otherwise, *pinOrder* values have the following effects: • If a pin has a *pinOrder* value of 0, the placement of the pin relative to the other pins is not important. • If a pin has a *pinOrder* value of 1, no pin can be placed between the pin and the corner of the boundary. • If a pin has a *pinOrder* value of 2, any number of pins can be placed between the pin and the corner of the boundary. • If two pins have consecutive *pinOrder* values, no pin can be placed between them. • If a group of pins has the same *pinOrder* value, no other pins can be placed between any two of the pins in the group. In this case, the ordering inside the group is not important. However, the placement of the group relative to the other pins on the same side is determined by the *pinOrder* value.
pinOffset	Relative offset of the pin placement from the bottom edge or the left edge of the cell boundary Valid values: Either of the following: • Any number (in distance units) that is greater than the height of the tallest pin in the layout • A negative number to indicate that you do not want to place an offset constraint on the pin In the case of contradictions between constraints, • If you specify a *pinOffset* value that contradicts the specified pin order, the Synopsys application will relax the *pinOffset* constraint • If you specify two *pinOffset* values that contradict each other, the Synopsys application will give priority to the *pinOffset* constraint on the pin with the lowest *pinOrder* value Note: To eliminate a previously loaded offset constraint for a pin, specify a negative number for *pinOffset*.

Example

```
pin "p1" 6 6 2 "top" 0 1500
pin "p2" 8 0 0 "right" 3 0
pin "p3" 7 9 3 "left" 1 0
pin "p4" 6 6 2 "top" 0 2000
pin "p5" 8 0 0 "left" 3 0
pin "p6" 5 18 3 "top" 1 0
pin "p7" 7 9 3 "right" 1 0
pin "p8" 5 6 2 "bottom" 1 0
pin "p9" 5 6 2 "right" 0 1200
pin "p10" 6 18 3 "bottom" 0 1400
```

Using these pin location directives, the tool based on Milkyway places pins as shown here.



rPin

The `rPin` command lets you set the following constraints for placement of a pin:

- Layer
- Dimensions

- Side number defining the side edge
- Order relative to other pins on the same side
- Offset from the bottom or left edge of the boundary

To set constraints on a pin not defined in the netlist, you must add the pin by using Enter > Pin before setting the constraints.

Note:
The rPin command is also covered in the `axgRectiPlanner` command documentation (see Physical Implementation Online Help).

Syntax

`rPin` *`pinName layer width height sideNumber`* `[pinOrder] [pinOffset]`

where the arguments are as follows:

`pinName`	Name of the pin to be constrained Valid values: Name of any pin in the design
`layer`	Name or number of the layer on which you want the pin Valid values: Name or number of any layer defined in the technology file or –1 to indicate that any layer can be used
`width`	Dimension of the pin along the cell boundary Valid values: Any positive floating-point number, or 0 to indicate the minimum possible width
`height`	Dimension of the pin from the boundary to the center of the cell Valid values: Any positive floating-point number, or 0 to indicate the minimum possible height Note: All but those designers very experienced with the tool should use a height equal to the width.
sideNumber	The side number is a positive integer that starts from 1. Given a rectangular shape, the lower left-most edge is the starting edge (side number 1). The side number of the next edge, going clockwise, is 2, and so on.

pinOrder

Order of the pin placement relative to the other pins on the same side

Valid values: Any integer from 0 to 32,767

Pins with lower *pinOrder* values are placed closer to the bottom or left edge of the boundary than pins with higher values. Otherwise, *pinOrder* values have the following effects:

- If a pin has a *pinOrder* value of 0, the placement of the pin relative to the other pins is not important.
- If a pin has a *pinOrder* value of 1, no pin can be placed between the pin and the corner of the boundary.
- If a pin has a *pinOrder* value of 2, any number of pins can be placed between the pin and the corner of the boundary.
- If two pins have consecutive *pinOrder* values, no pin can be placed between them.
- If a group of pins has the same *pinOrder* value, no other pins can be placed between any two of the pins in the group. In this case, the ordering inside the group is not important. However, the placement of the group relative to the other pins on the same side is determined by the *pinOrder* value.

pinOffset

Relative offset of the pin placement from the bottom edge or the left edge of the cell boundary

Valid values: Either of the following:

- Any number (in distance units) that is greater than the height of the tallest pin in the layout
- A negative number to indicate that you do not want to place an offset constraint on the pin

In the case of contradictions between constraints,

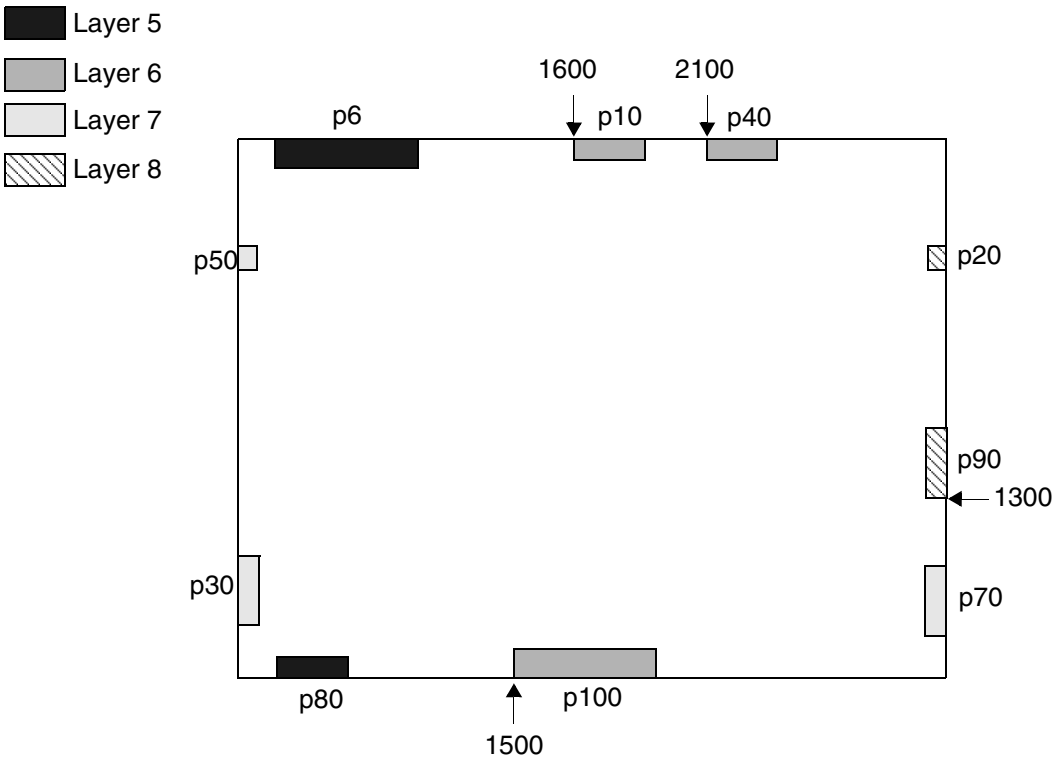
- If you specify a *pinOffset* value that contradicts the specified pin order, the Synopsys application will relax the *pinOffset* constraint
- If you specify two *pinOffset* values that contradict each other, the Synopsys application will give priority to the *pinOffset* constraint on the pin with the lowest *pinOrder* value

Note: To eliminate a previously loaded offset constraint for a pin, specify a negative number for *pinOffset*.

Example

```
rpin "p10" 6 6 2 2 0 1600
rpin "p20" 8 0 0 3 3 0
rpin "p30" 7 9 3 1 1 0
rpin "p40" 6 6 2 2 0 2100
rpin "p50" 8 0 0 1 3 0
rpin "p60" 5 18 3 2 1 0
rpin "p70" 7 9 3 3 1 0
rpin "p80" 5 6 2 4 1 0
rpin "p90" 5 6 2 3 0 1200
rpin "p100" 6 18 3 4 0 1500
```

Using these pin location directives, the tool based on Milkyway places pins as shown in the above example.



E

Preparing a Library and Generating an LM View

This chapter contains information about preparing a library for use in the Milkyway database. It contains the following sections:

Library Preparation for Milkyway

The tasks necessary to prepare for using a Milkyway database are described in the following sections:

- Library Preparation Overview
- Steps for Preparing a Reference Library
- Library Preparation for a Library With Multi-Height Cells
- Library Preparation for a Library With Multi-Height Sites
- Setting Up Design Libraries
- Generating the Technology File from LEF Information
- Creating the Milkyway Design Library Directory

Library Preparation Overview

This section provides an overview of library preparation and the process of importing an existing .db file to create an LM (logic model) view. For steps that guide you through the library preparation process and the importing of .db files, see “Steps for Preparing a Reference Library” on page E-5.

You need to prepare each reference library. However, for each reference library, perform the library preparation only once.

Library preparation accomplishes the following:

- Imports .db data into Milkyway (using the `gePrepLibs` command), which creates an LM view that includes the logic and timing libraries.
- Registers minimum, typical, and maximum logic libraries in the LM view.
- Creates the .pdb file.

Figure E-1 shows the high-level flow for preparing a reference library.

Figure E-1 Using the gePrepLibs Command

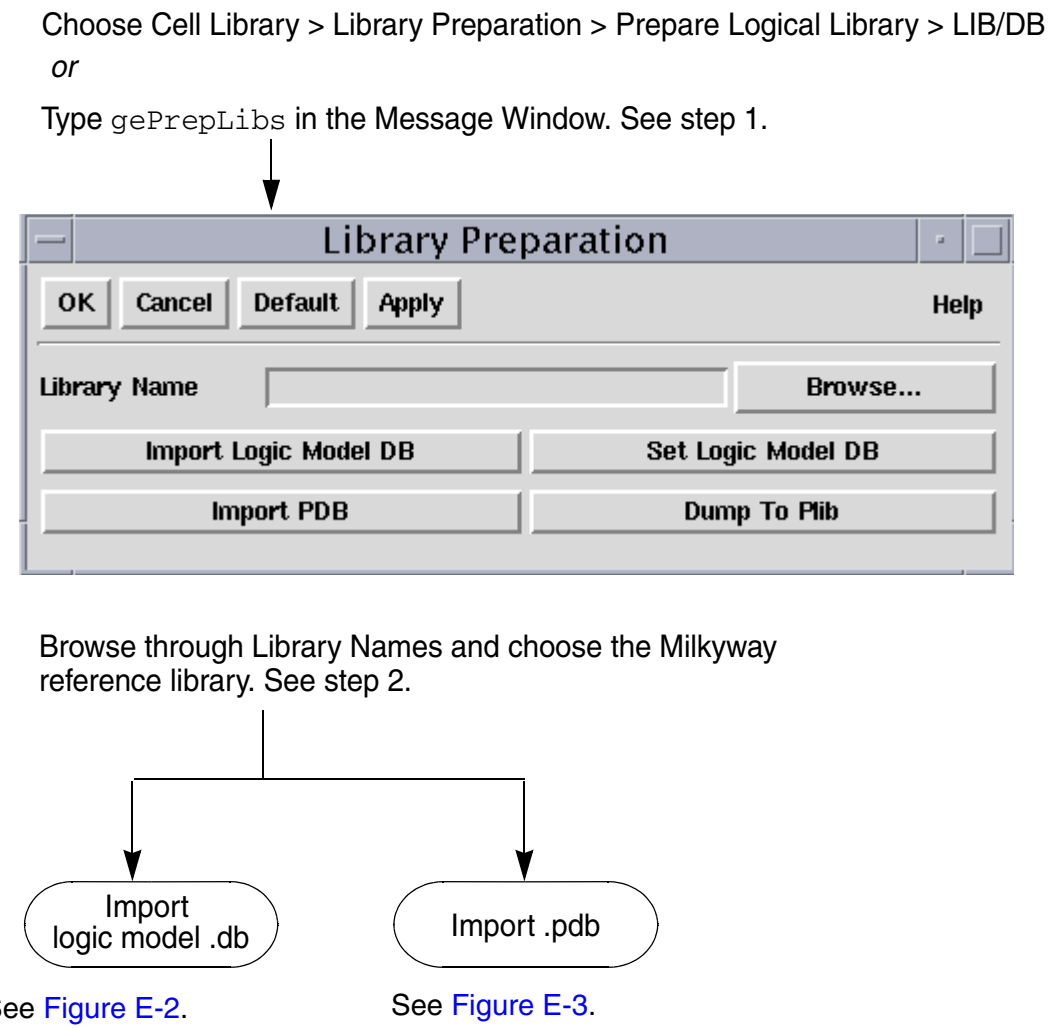


Figure E-2 shows the Library Preparation dialog box for importing the logic database and the two ways you can import the database.

Figure E-2 Importing the Logic Database

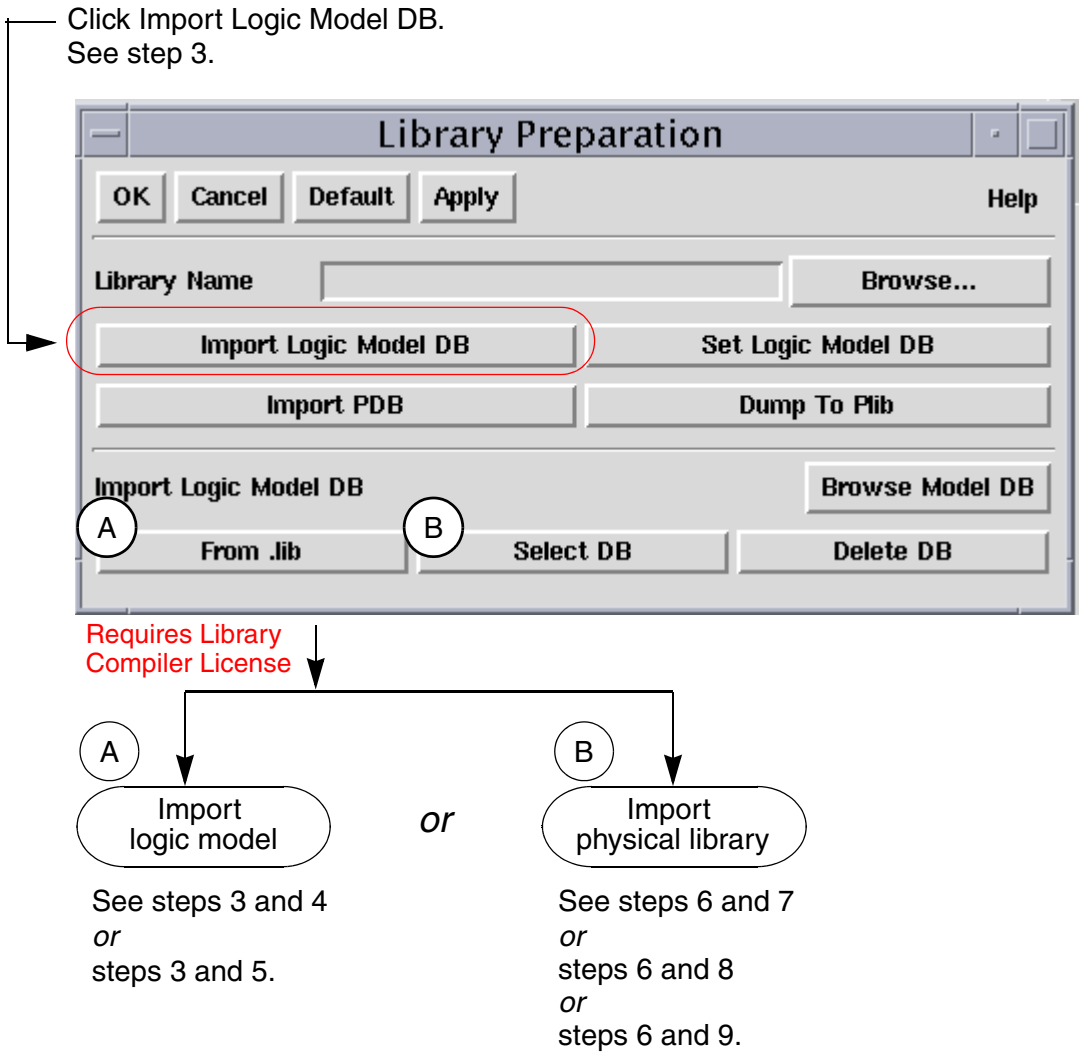
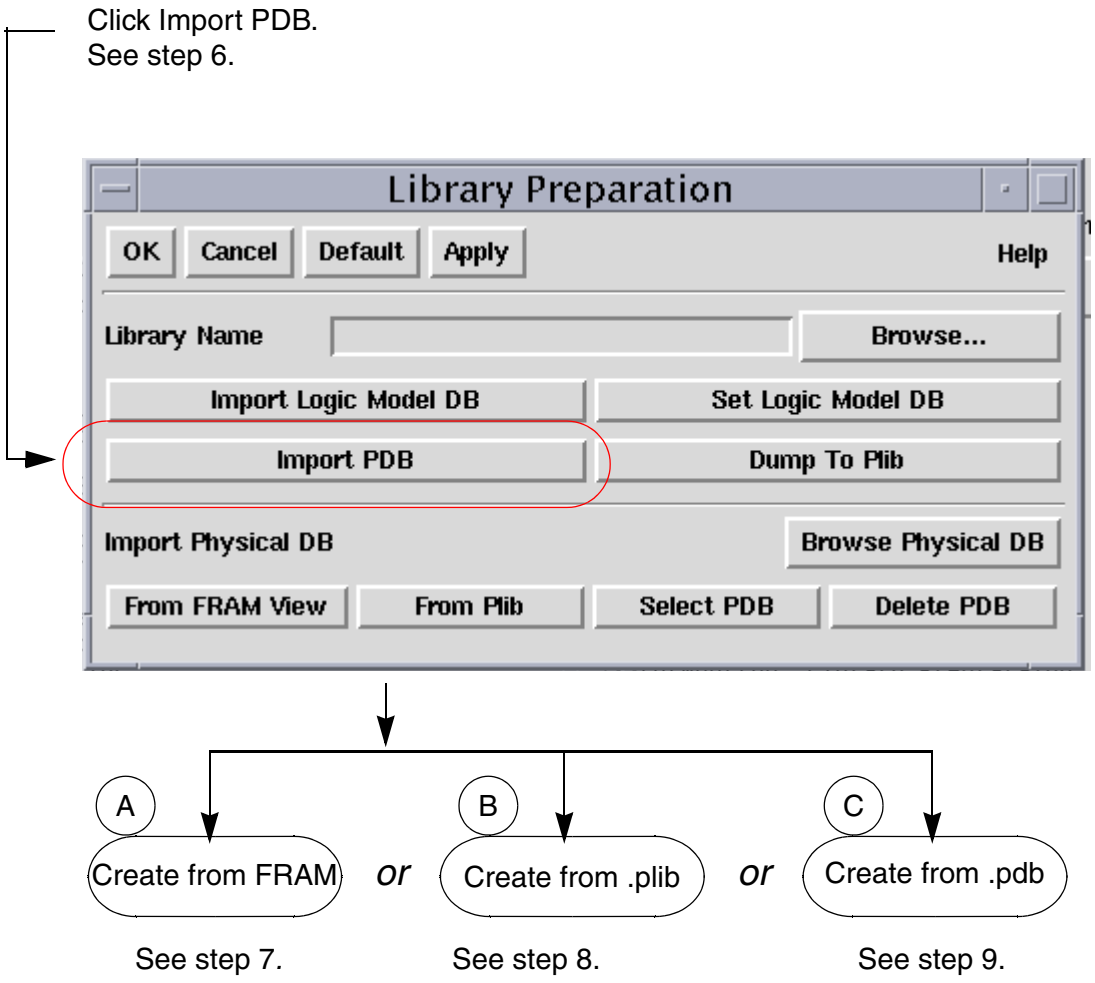


Figure E-3 shows the Library Preparation dialog box for importing the physical database and the three ways you can use to import the database.

Figure E-3 Importing the Physical Database



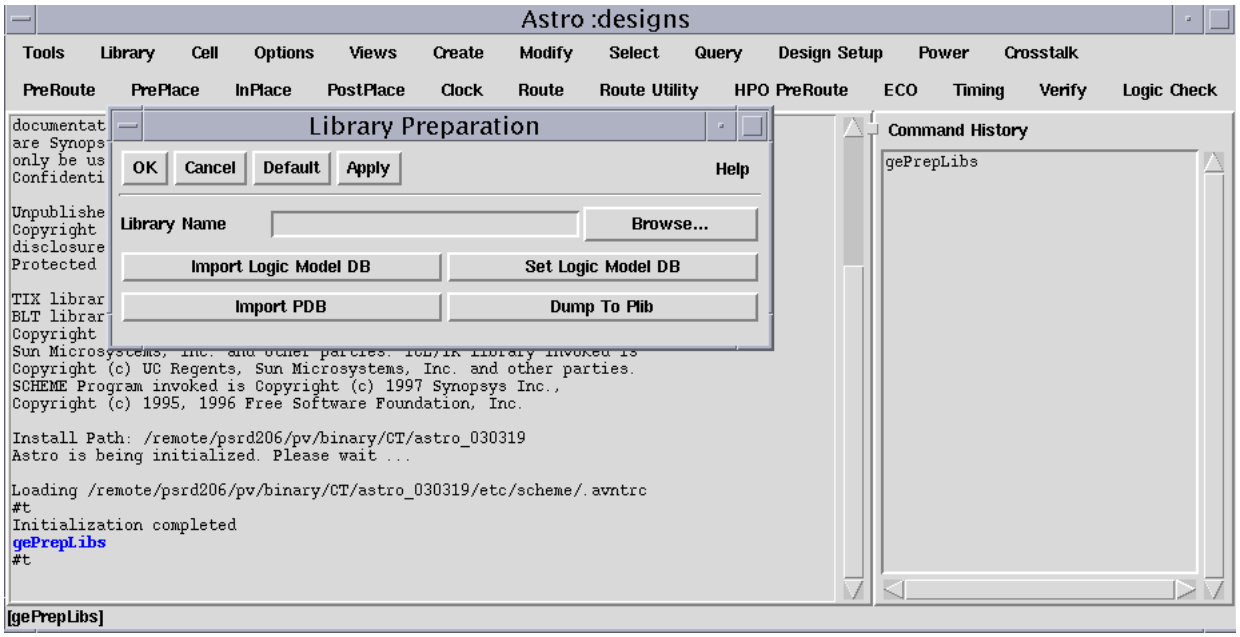
The following section guides you through the steps required to prepare a reference library and create an LM view.

Steps for Preparing a Reference Library

To prepare a reference library and create an LM view,

1. Run the `gePrepLibs` command to open the Library Preparation dialog box.

   Or, you can choose Cell Library > Library Preparation in the menu, which opens the Read Library dialog box. Click the Prepare Logical Library button, and then click LIB/DB to open the Library Preparation dialog box.

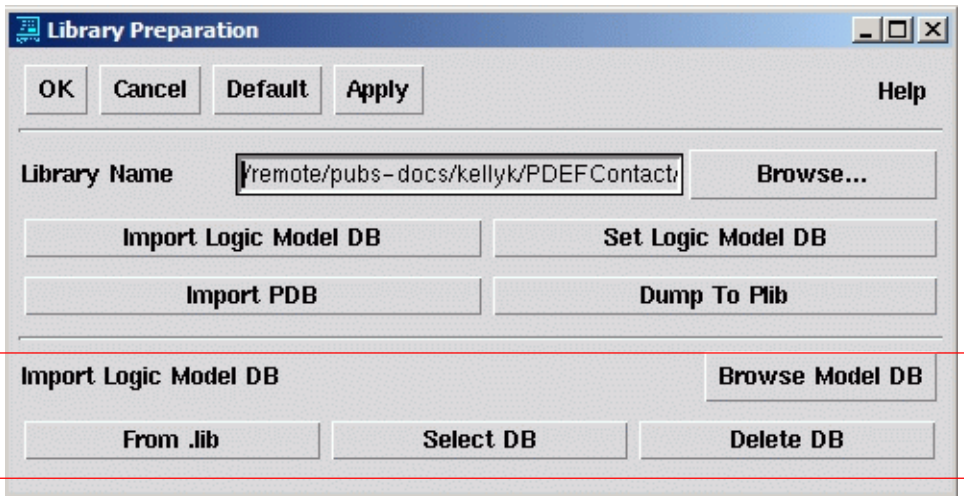


2. Browse the library names and choose the Milkyway reference library.

   You must specify the library name to proceed with the library preparation steps and to be able to run any command.

3. Click Import Logic Model DB.

   The Library Preparation dialog box expands to enable importing the logic libraries, as shown below.



Important:

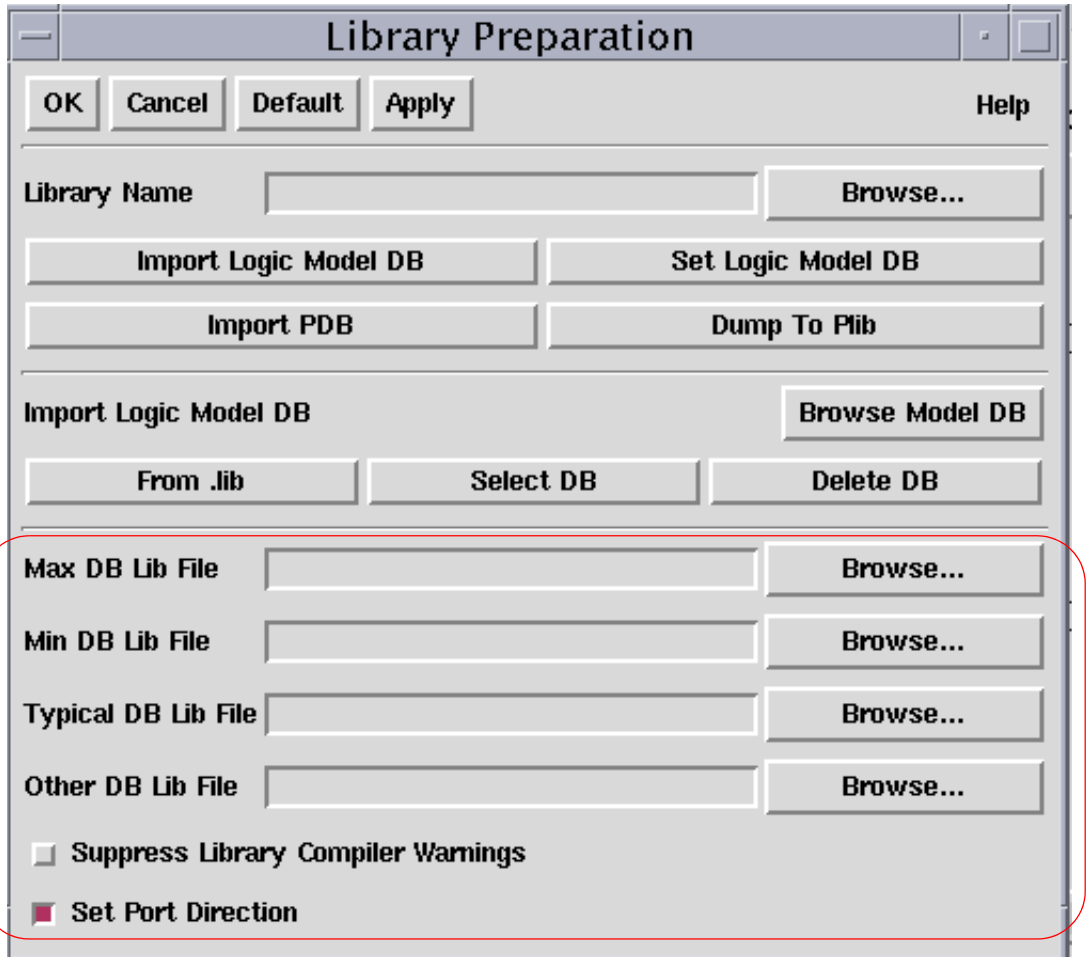
You must perform either step 4 or step 5. The step you choose depends on whether you have a .lib (ASCII) file as the timing library (choose step 4) or a .db file (choose step 5).

4. If you do not have a .lib file as the timing library, do not use this step; instead, use step 5. This step requires a Library Compiler license.

   If you have a .lib file as the timing library,

   - Click “From .lib.”

     The Library Preparation dialog box expands, providing text boxes for typing in names, or you can browse to choose the maximum, minimum, and typical .lib files.



- Specify the UNIX file names for the .lib files representing maximum, minimum, and typical operating conditions. To do this, type the path names in the Max DB Lib File, Min DB Lib File, and Typical DB Lib File text boxes. You can specify more than one file of each type. If a library is not available, leave that text box empty.

  The Set Port Direction check box is turned on by default and port directions of all the cells in the FRAM view are set from the corresponding port direction of that cell from Lib files. If you are sure the port directions of all cells in the Milkyway FRAM view are correct, you can click Set Port Direction off.

- Click Apply.

Note:

To delete a specified timing library from Milkyway, click Delete DB and select the file you want to delete.

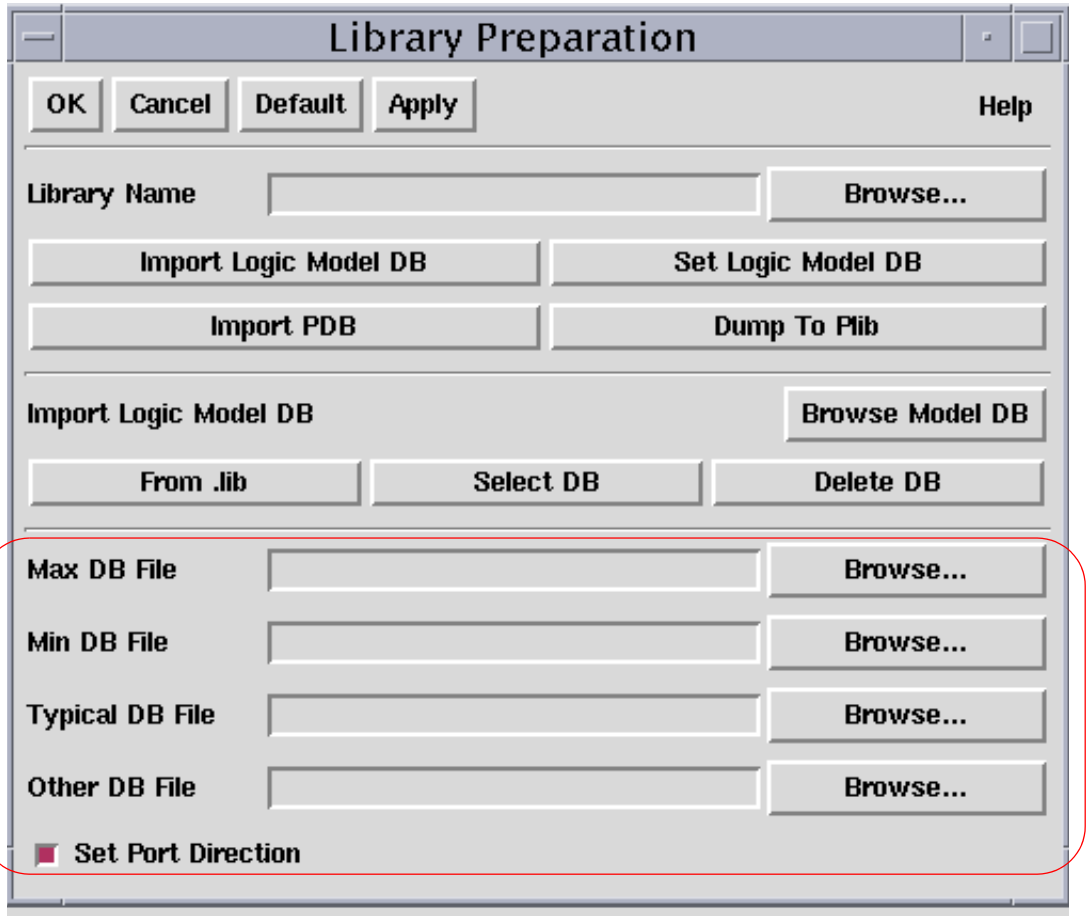
5. If you do not have a .db file as the timing library, do not use this step; instead, use step 4.

   If you have a .db file as the timing library,

   - Click Select DB.

     The Library Preparation dialog box expands, providing text boxes for typing names, or you can browse to choose maximum, minimum, or typical .db files.

     The Set Port Direction check box is turned on by default, and port directions of all the cells in the FRAM view are set from the corresponding port direction of that cell from .lib files. If you are sure the port directions of all cells in the Milkyway FRAM view are correct, you can click Set Port Direction to turn the option off.



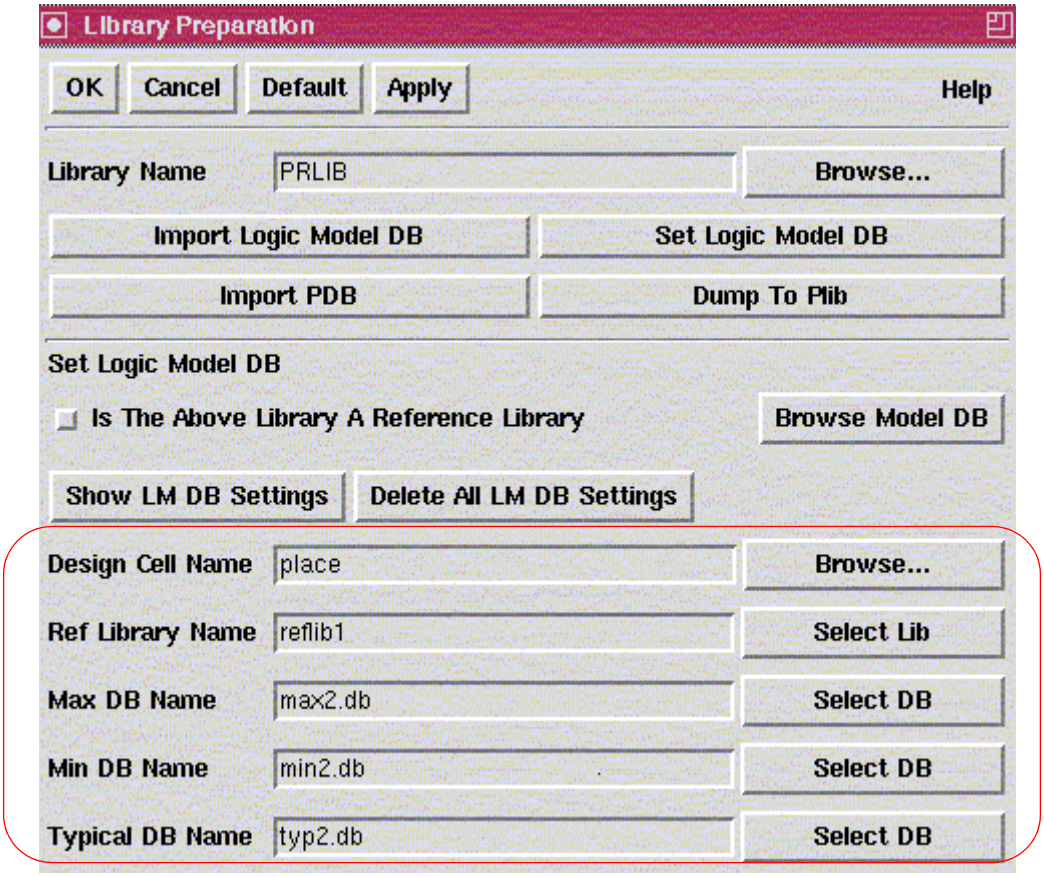
- Specify the UNIX directory path names for each .db file representing libraries for maximum, minimum, and typical operating conditions. To do this, type the path names in the Max DB File, Min DB File, and Typical DB File text boxes. If a library is not available, leave that text box empty.
- Click Apply.

Note:

To delete a specified timing library from Milkyway, click Delete DB and select the file you want to delete.

6. (Optional) You can modify or change on a case-by-case basis the current logic models loaded in the Milkyway library. You can assign different process corners for every design cell in every library and decide which model is loaded in to Astro. To do this, click Set Logic Model DB.

   The Library Preparation dialog box changes, providing text boxes for typing in the design cell name, reference library name, and process corners you want.



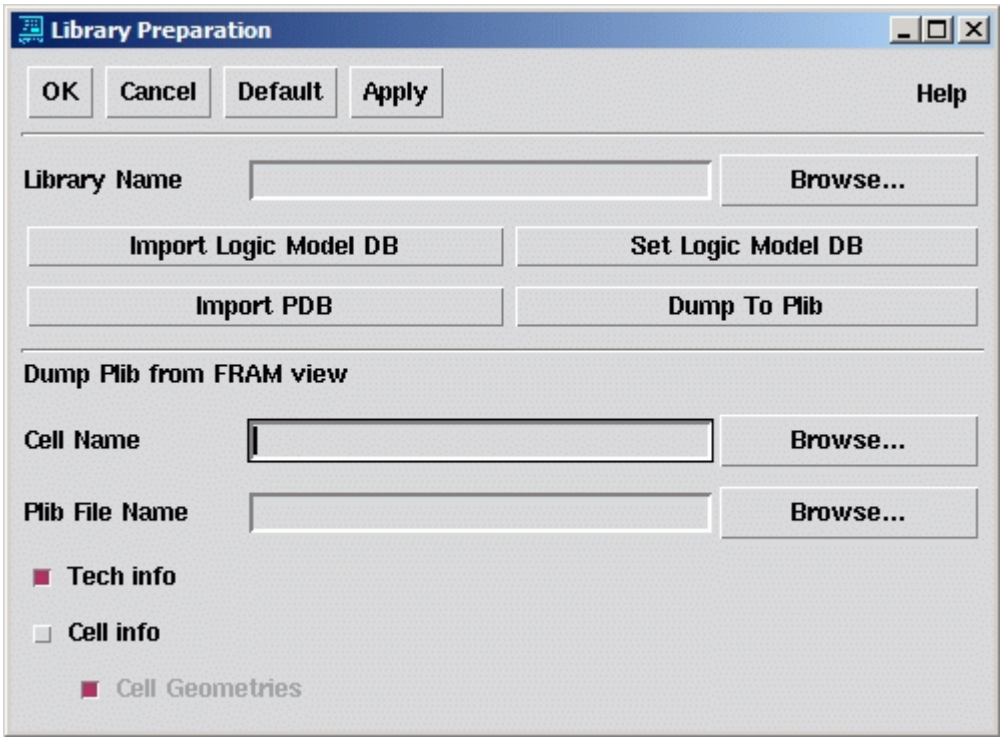
Provide the information for each design cell you want to change and click Apply.

Note:

Steps 7 through 12 are optional steps for Physical Compiler in XG mode. It is not required that you create or import a PLIB/PDB file if you have a Milkyway reference library because Physical Compiler in XG mode reads information directly from the Milkyway reference library.

7. (Optional) If you want to write a PLIB file from your Milkyway reference library, click Dump To Plib.

   The Library Preparation dialog box expands (as shown below). You can also write to a PLIB file by invoking the Dump PLIB dialog box by choosing Library > Dump Plib in the Astro or Milkyway menu.



Complete the following steps to write a physical library in .plib format:

a. Type the library name in the Library Name text box or select it from the browser.

b. Type the cell name in the Cell Name text box or select it from the browser.

c. Type the UNIX path for the .plib files in the Plib File Name text box or select it from the browser.

d. Select from the following options:

Tech info

This option is selected by default. When Tech info is selected, Milkyway writes a technology .plib file. You can also

- write a complete .plib file by selecting the Tech info option, the Cell info option, and the Cell Geometries option.
- write a .plib file with technology and cell information, but without geometry information, by selecting the Tech info option and the Cell info option and deselecting the Cell Geometries option.

Cell info

This option is not selected by default. If you select this option, Milkyway writes a .plib file with cell information. You can

- write a .plib file with cell information and geometries, but without technology information, by selecting the Cell info option and deselecting the Tech info option.
- write a .plib file with technology and cell information, but without geometry information, by selecting the Tech info option and the Cell info option and deselecting the Cell Geometries option.
- write a .plib file with cell information but without technology information or cell geometries by selecting the Cell info option and deselecting the Tech info option and the Cell Geometries option.

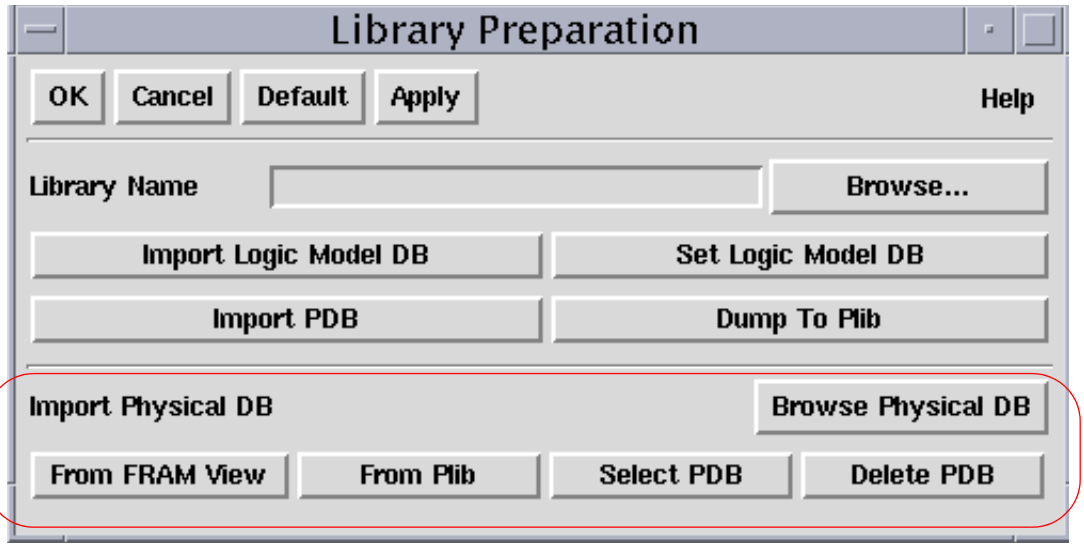
  Cell Geometries

  This option is selected by default when you select the Cell info option. When Cell Geometries is selected, Milkyway writes a .plib file with cell info and geometries.

e. Click Apply.

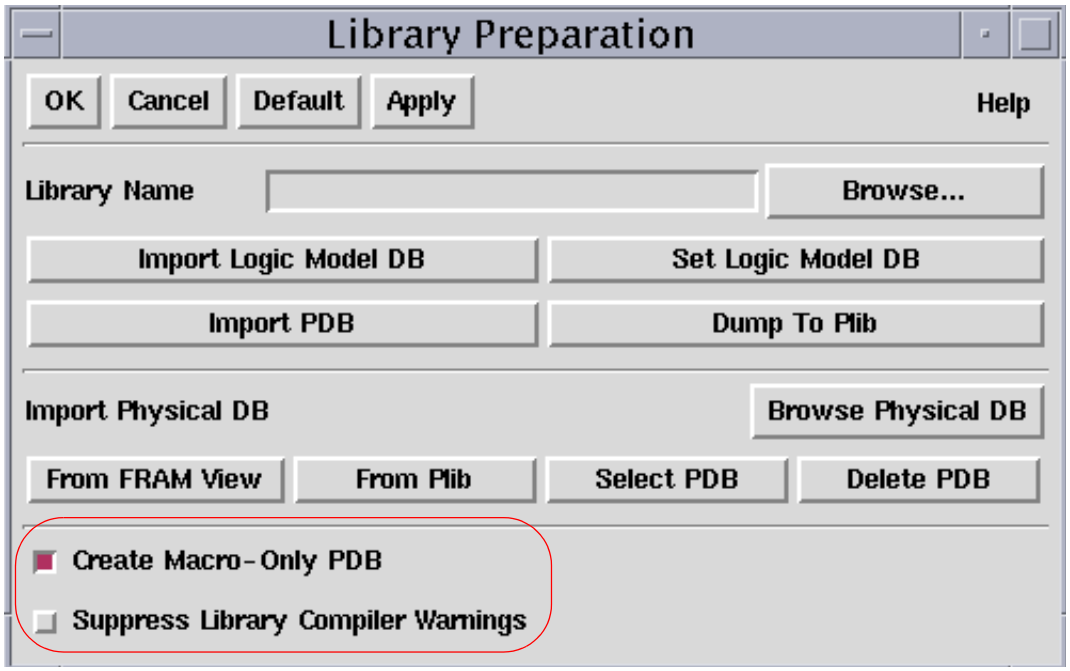
8. (Optional) Click Import PDB to bring in the physical libraries.

   The Library Preparation dialog box for importing the physical database appears.



9. (Optional) If you do not have .plib or .pdb files or if you want to create the physical library from the Milkyway reference library, use this step.
   - Click From FRAM View.

   The Library Preparation dialog box expands to enable importing physical library data.



If you want to create a .pdb file without technology information, click Create Macro – Only PDB on, then click Apply.

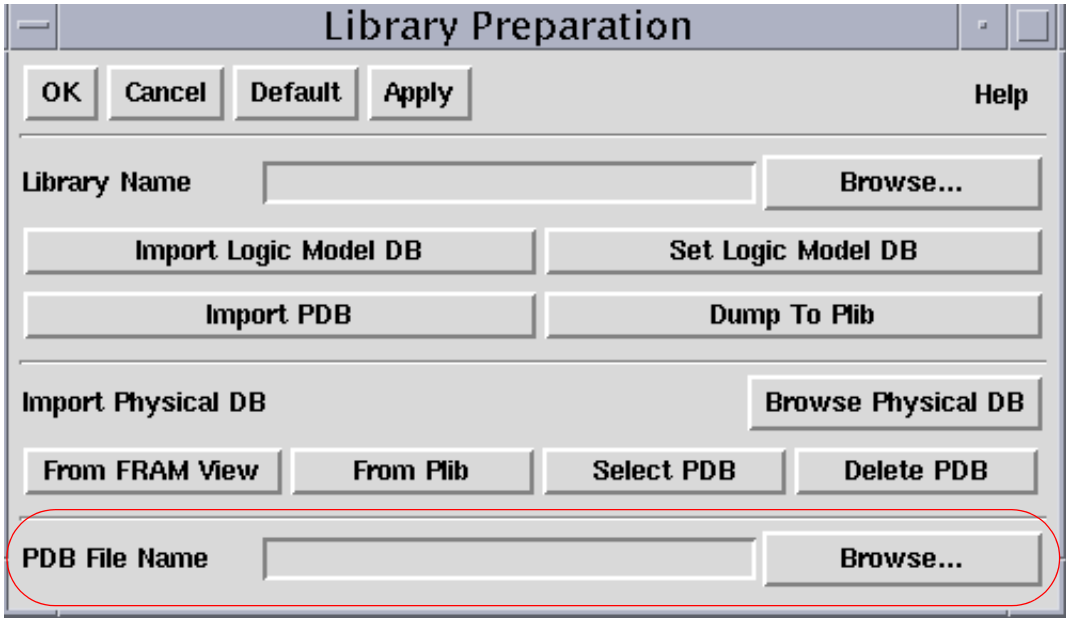
For a standard cell or macro library, generating a macro-only .pdb file ensures that all technology information comes from the tf file.

10. (Optional) If you already have a .plib file and want to use it, click From Plib.

    The Library Preparation dialog box expands, providing a text box for typing in the name of the PLIB file, or you can browse to choose the PLIB file. Type the .plib file name in the text box, then click Apply.

11. (Optional) If you already have a .pdb file and want to use it, click Select PDB.

    The Library Preparation dialog box expands, providing a text box for typing in the name of the physical library, or you can browse to choose the physical library.



Type the .pdb file name in the PDB File Name text box, then click Apply.

12. (Optional) If you have multiple .pdb files for the same Milkyway reference library, repeat step 11 for each .pdb file.

    Note:

    To delete a specified .pdb library from Milkyway, click Delete PDB, then select the name of the .pdb file you want to delete.

Library Preparation for a Library With Multi-Height Cells

Preparing a library with multi-height cells uses these major steps shown below.

1. Generate CEL views by streaming in a GDS file (`auStreamIn`) or a LEF file.
2. Extract blockage, pin, and via information (`auExtractBlockagePinVia`).
3. Mark the cell type (described following).
4. Set the PR boundary (described following).
5. Run `gePrepLibs` to create and attach the LOGIC view and the .pdb file.

Marking the Cell Type

To mark the cell type, follow these steps:

1. Choose Tools > DataPrep > Cell Library > Mark Cell Type
   or
   Type `cmMarkCellType`
2. Specify the Library Name.
3. Specify the Cell Name of the multi-height cell.
4. Select the Cell Type as “double height” or “triple height” or “triple+ height” based on the height of the cell with respect to the site height.
5. Click OK.

Setting the PR Boundary

To set the PR boundary, follow these steps:

1. Choose Tools > DataPrep > Cell Library > Set PR Boundary
   or
   Type `auSetPRBdry`.
2. Specify the Library Name.
3. Specify the Cell Name as .*, which means all cells in the specified library.
4. Select “based on marked cell type.”
5. Click OK.

Library Preparation for a Library With Multi-Height Sites

Preparing a library with multi-height sites uses these major steps:

1. Generate CEL views by streaming in a GDS file (`auStreamIn`) or a LEF file (`auNLIApi`).
2. Extract blockage, pin, and via information (`auExtractBlockagePinVia`).
3. Set PR Boundary for single-height cells (described following).
4. Set PR Boundary for multi-height cells (described following). You do this for each multi-height cell height you need.
5. Run `gePrepLibs` to create and attach the LOGIC view and the .pdb file.

Setting the PR Boundary for Single-Height Cells

To set the PR boundary for single-height cells, follow these steps:

1. Choose Tools > DataPrep > Cell Library > Mark Cell Type
   or
   Type `auSetPRBdry`.

2. Specify the Library Name.
3. Specify the Cell Name as .*, which means all cells in the specified library.
4. Select “all cells are single height.”
5. Click OK.

Setting the PR Boundary for Multi-Height Cells

To set the PR boundary for multi-height cells, follow these steps. You do these steps for each multi-height cell height you need.

1. Choose Tools > DataPrep > Cell Library > Mark Cell Type
   or
   Type `auSetPRBdry`.
2. Specify the Library Name.
3. Specify the Height.
   - Select “specify.”
   - Enter the cell height as the value.
4. Specify the Tile Name.
   - Select “specify.”
   - Enter the name of the tile.
5. Select “all cells are single height.”
6. Click OK.

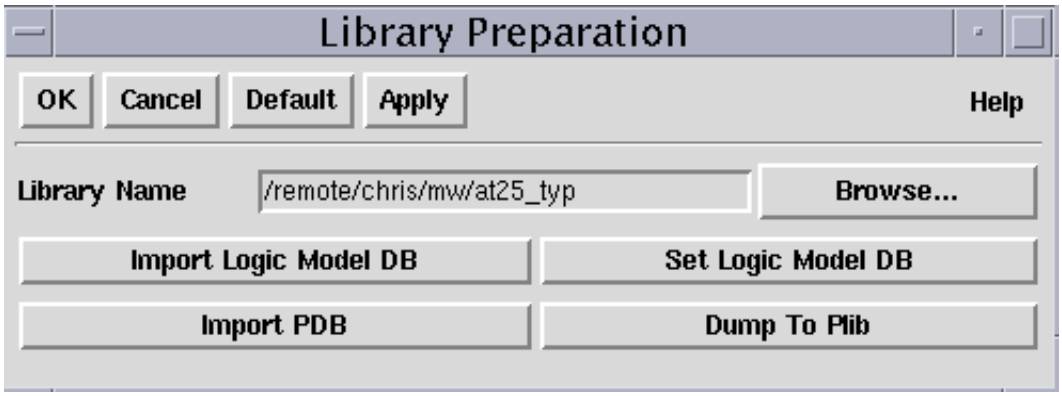
Setting Up Design Libraries

Before running any Physical Compiler operations, you need to set the `search_path`, `target_library`, `link_library`, `mw_reference_library`, and `mw_design_libraryvariables`.

Example

This example shows how, after completing the library preparation steps, you use this library in a shell such as psyn_shell.

If your reference library is in /remote/chris/mw/at25_typ, the name in the Library Name text box is /remote/chris/mw/at25_typ:



1. After completing the library preparation steps, set the `search_path, link_library`, and the `target_library`. For example, enter

```
psyn_shell-t> set search_path (. )
psyn_shell-t> set link_library (mwlib_max.db \
      mwlib_min.db *}
psyn_shell-t> set target_library (mwlib_max.db \
      mwlib_min.db}
psyn_shell-t> set_operating_conditions \
      -min mwlib_min -min_library mwlib_min \
      -max mwlib_max -max_library mwlib_max
```

2. Set the Milkyway library path. You need to provide absolute path names for your reference libraries. For example, enter

```
psyn_shell-t> set mw_reference_library \
      directory_pointer_where_reference_milkyway_library_is
```

   If you have multiple reference libraries, determine which reference library contains technology information and list this library as the first reference library in `mw_reference_library`. For example, if you have ref_lib1, ref_lib2, and ref_lib3, and ref_lib3 contains the technology information, ref_lib3 must be first in the list. For example, enter

```
psyn_shell-t> set mw_reference_library [list /remote/ \
mw/at25_typ/ref_lib3 /remote/mw/at25_typ/ref_lib1 \
/remote/mw/at25_typ/ref_lib2]
```

   Important:

   You must use absolute path names.

3. Set up the output directory in which to write the design in Milkyway format. For example, enter

```
psyn_shell-t> set mw_design_library directory_name
```

4. Create a Milkyway design library directory.

   To create a Milkyway design library, enter

   ```
   psyn_shell-t> create_mw_design -tech tech_file
   ```

   The `create_mw_design` command creates a Milkyway library structure in the specified library directory for `write_milkyway` or `read_milkyway` to use.

   The `create_mw_design` command creates a Milkyway design library and requires that you specify `-tech_file` *`tech_file`*.

5. If you have a DesignWare component that does not have a corresponding FRAM library,

   - Include the DesignWare library directory location in your search path.
   - Include the DesignWare library in your `link_library`.

   The following example is for a scenario where the dw_lib.sldb library is present in the .lib/db directory.

   Setup.tcl

   ```
   # Set search_path
   set search_path "../lib/db $synopsys_root/libraries/syn"

   # Set libraries
   set target_library "lib_max.db"
   set link_library "* $target_library dw_lib.sldb \
        dw01.sldb dw02.sldb dw_foundation.sldb"
   set synthetic_library "dw_lib.sldb dw01.sldb dw02.sldb \
        dw_foundation.sldb"
   ```

   Run.tcl

   ```
   source setup.tcl
   set mw_design_library design_fr_pc
   # Provide the full path name for mw_reference_library
   because
   # mw_reference_library does not honor the search_path
   variable.
   set mw_reference_library [list lib/mw/phylib]
   create_mw_design -tech_file astro.tf
   read_db design.db
   link
   physopt -mpc
   ```

Generating the Technology File from LEF Information

To generate Milkyway reference library, you need a technology (.tf) file. Usually your vender prepares this file but you can generate a simple file from LEF information with some manual effort. Use these steps to generate a .tf file.

1. Dump out the technology file.

   This file contains VIARULE, which is generated from LEF information. Check the following options' settings:

   - Make sure that the "skip techfile" option in Tech Options is turned off. This allows the Milkyway Library Technology information to be modified according to the statements in the LEF file.
   - Make sure that if the LEF file contains a poly layer definition and poly contact definition, turn on the "poly layer" option in Layer Options. Otherwise, layer mapping mismatch errors occur in the later stages.

2. Dump out the Technology file from this Library for editing. Select Library > Dump Tech File.

For syntax checking purposes, use this technology file to create another Library by using the Library > Create option.

Creating the Milkyway Design Library Directory

You must create the Milkyway design library directory before you can use the `write_milkyway` command. This design library directory is used by tools such as Milkyway. To create the Milkyway design library directory, use the `create_mw_design` command.

The `create_mw_design` command creates a Milkyway design library using the technology file.

When you use `create_mw_design`, keep the following points in mind:

- Library preparation must be done for `create_mw_design` to work correctly.
- Before you run `create_mw_design`, specify the `mw_reference_library` and `mw_design_library` variables. The `mw_reference_library` variable is used to set the Milkyway reference libraries for the design and to enhance the `search_path` variable.
- You must run `create_mw_design` before reading any design.db file and before running `write_milkyway` the first time.

Because `search_path` is set by `create_mw_design`, if you do not run the command, libraries are not in `search_path` and the design fails during the link process.

- The `-tech_file` option is required when a Milkyway design is created for the first time.

To use the `create_mw_design` command, enter

```
psyn_shell-t> create_mw_design [options]
```

To do this	Use this
Specify the Milkyway technology file used to create the design library; required when a Milkyway design is created for the first time. This file is not required if a Milkyway design directory for the same technology file exists.	-tech_file
Specify the name of the min capacitance table file used for TLUPlus models.	-min_tluplus
Specify name of the max capacitance table file used for TLUPlus models.	-max_tluplus
Specify name of the nominal capacitance table file used for TLUPlus models.	-nom_tluplus
Specify the name of the ITF-to-Milkyway layer map file used for TLUPlus models. Use this option when minimum, maximum, or nominal values are present.	-tf2itf_map
Specify the design library directory to create in which to write the Milkyway library. If you omit this option, make sure the variable `mw_design_library` is set to the correct design library.	*design_dir*

For more information, see the man page.

Example

To create a design library named testmdb and prepare the design for processing, use the following command sequence:

```
psyn_shell-t> set mw_reference_library \{'this', 'that'}
psyn_shell-t> set mw_design_library testmdbpsyn_shell-t>

create_mw_design -tech_file  testmdb.tf
```

Setting Up TLUPlus Libraries for Physical Compiler Flow

You can set up TLUPlus libraries in either of two ways, depending on whether you plan to run a Physical Compiler or Physical Compiler and Astro flow. The Physical Compiler commands to accomplish the set up are

- `create_mw_design`

  Use this command to set up for a Physical Compiler flow or Physical Compiler and Astro flow. See “Setting Up a TLUPlus Library” on page E-21.

- `set_tlu_plus_files`

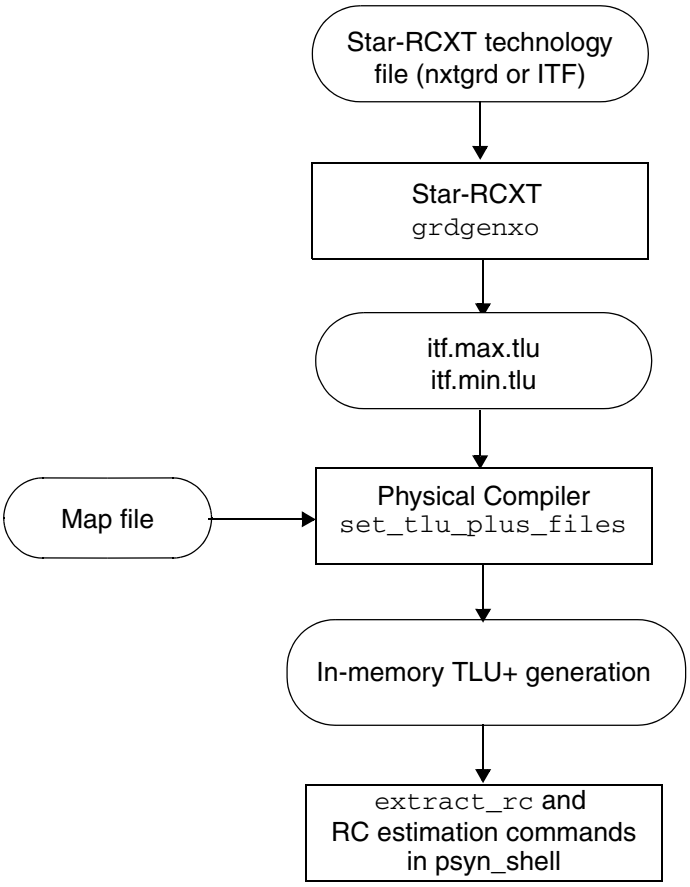
  Use this command to set up for a Physical Compiler flow. See “Setting Up a TLUPlus Library for a Physical Compiler and Astro Flow” on page E-22.

  If you are using a Physical Compiler and Astro flow, use `create_mw_design`. See “Creating the Milkyway Design Library Directory” on page E-19.

Setting Up a TLUPlus Library

The flow to set up the TLUPlus library to use in a Physical Compiler flow either preroute or postroute is shown in Figure E-4. Use this setup when you are not using Astro or Physical Compiler Expert.

Figure E-4 Flow to Set Up and Use TLUPlus for Physical Compiler



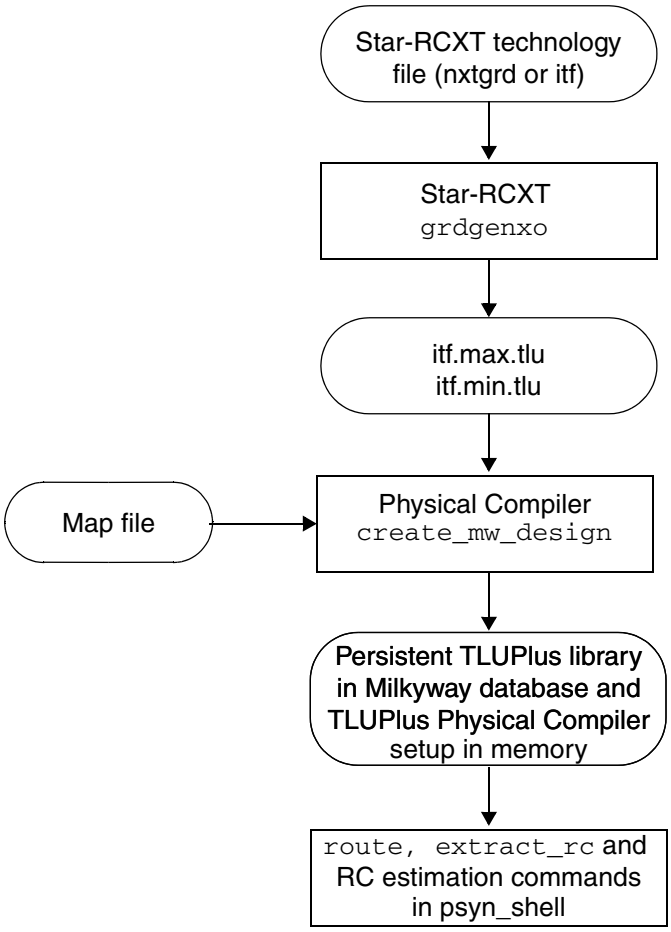
As shown in Figure E-4, this flow uses the `set_tlu_plus_files` command to specify the itf.tlu file. You use this flow for preroute estimation (`run_router` or `physopt`) or postroute extraction (`extract_rc`).

See “TLUPlus-Based Extraction for a Physical Compiler or Third-Party Flow” in Physical Compiler documentation. For information about Astro, see the Astro documentation. For information about Star-RCXT, see the Star-RCXT documentation.

Setting Up a TLUPlus Library for a Physical Compiler and Astro Flow

The flow to set up the TLUPlus library for a Physical Compiler and Astro flow either preroute or postroute is shown in Figure E-5. Use this setup when you are using Astro or Physical Compiler Expert.

Figure E-5 Setting Up and Using TLUPlus in a Physical Compiler and Astro Flow



As shown in Figure E-5, this flow uses the `create_mw_design` command to create a persistent TLUPlus library in the Milkyway database. (See "Creating the Milkyway Design Library Directory" on page E-19.) This library is persistent on disk and can be read by Astro. The command also creates a map file for Physical Compiler that maps the .tf file and ITF file that you can use with the `set_tlu_plus_files` command.

A mapping file is required to map between Milkyway mask names and ITF layer names. (See "Layer Mapping Between Technology File and ITF File" on page E-24.) You use this flow for postroute with the `route`, `extract_rc`, and RC estimation commands.

For information about Astro, see the Astro documentation. For information about Star-RCXT, see the Star-RCXT documentation.

Running the Library Preparation Flow for TLUPLus Extraction

The Milkyway library preparation flow for Physical Compiler includes the following steps. Before you do the steps, make sure that tf and TLU units match. For information, see the Star-RCXT documentation.

1. Make sure you have the interconnect technology format (ITF) file, which contains technology information such as oxide thickness, metal thickness, and spacing. The ITF file and the nxtgrd file are both technology files for Star-RCXT.

   If you do not already have the ITF file, create it from the nxtgrd file, which contains a copy of the ITF file as comments. The ITF portion of the nxtgrd file is between "$TECHNOLOGY=" and "end of itf file". Remove the $ at the beginning of each line. Copy the ITF portion from the nxtgrd file and paste it in a new file. Usually you save this ITF file as itf.

2. Run the `grdgenxo` command (a Star-RCXT command, which requires a Star-RCXT license) for each operating condition you specified, using the ITF file as input.

   Physical Compiler accepts two operating conditions—min and max. (Astro can accept up to three operating conditions.)

   Enter

   ```
   grdgenxo -itf2TLUPlus -i itf
   ```

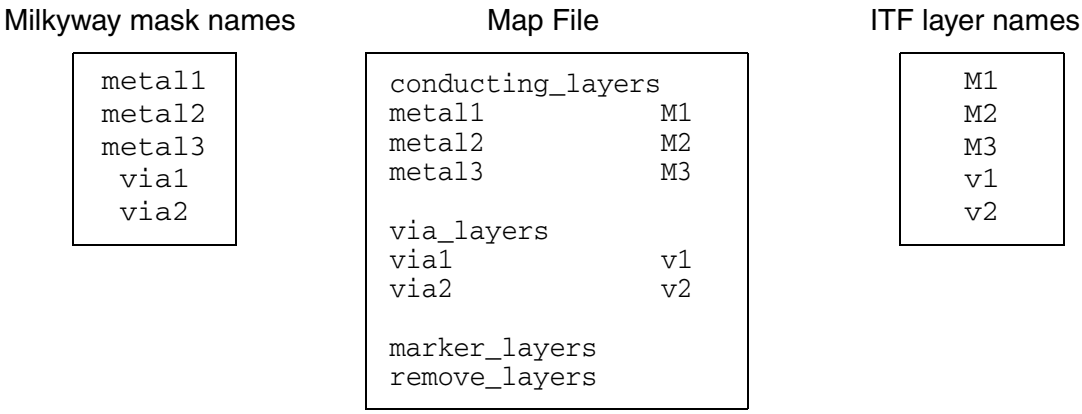
   The generated file is named *itf_file_name*.tlu by default.

Layer Mapping Between Technology File and ITF File

You may need a map file for `create_mw_design`. This map file is a layer name mapping file between the Milkyway technology file and the Star-RCXT technology file (nxtgrd file or ITF file).

Figure E-6 shows layer mapping between the Milkyway technology file and the ITF file.

Figure E-6 Layer Mapping Between Milkyway Technology File and ITF File



For example, the Milkyway technology file contains

```
Layer      "M1" {
                  layerNumber             = 1
                  maskName                = "metal1"
```

The mask name in the Milkyway technology file is metal1 and the layer name in the ITF file is M1. Therefore, your map file should contain the following, as shown in Figure E-6.

```
conducting_layers
metal1                M1
```

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W