IC Compiler
Data Preparation Using Milkyway™
User Guide

Version C-2009.06-SP4, December 2009

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, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, 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, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, 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, StarRC, 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

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:

   https://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 User Guide

The *IC Compiler Data Preparation Using Milkyway User Guide* describes how to use the Milkyway tool to create libraries in the Milkyway Environment for IC Compiler. A related manual is the *Milkyway Environment Data Preparation User Guide*, which focuses on data preparation using the Scheme extension language for the older physical implementation tools Astro and JupiterXT. Because the focus of this user guide is on IC Compiler, it does not describe Scheme commands or Milkyway views and data formats not supported by IC Compiler.

Audience

Users of this manual must be familiar with Tcl scripting and physical IC implementation concepts, including standard-cell-based design.

This guide provides information about preparing for the place and route process, preparing libraries, translating physical design data, processing standard cells, and setting up required technology data for use with IC Compiler.

Related Publications

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

https://solvnet.synopsys.com/DocsOnWeb

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

- IC Compiler
- Design Compiler

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:

https://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/GlobalSupportCenters/Pages
- 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/GlobalSupportCenters/Pages

1

Starting the Data Preparation Process

The data preparation tasks described in this guide are performed with the Milkyway Environment data preparation tool.

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 Milkyway Environment tool, not the database.

This chapter contains the following sections:

Milkyway Compatibility with Synopsys Tools

Every Milkyway database release has a schema version. This means that each Milkyway database release supports specific versions of a Synopsys tool. A Milkyway database has to be loaded into a compatible release of a Synopsys tool.

The following table indicates the schema versions of various releases of Milkyway Environment.

Milkyway Environment Version	Schema Version
C-2009.06	4.0
B-2008.09	3.2
A-2007.12	2.0

With the C-2009.06 Milkyway database schema version 4.0, physical-only cells are clearly defined and stored. Interface Logic Models (ILMs) created under version B-2008.09 can be automatically converted to the latest schema. The database supports wire master index values greater than 255, up to 8192. Explicit power and ground net description data are also better supported. The enhanced database schema allows IC Compiler to consistently handle design information, providing a foundation for future improvements.

IC Compiler version C-2009.06 reads a Milkyway database of schema version 3.2 and earlier. However, note that IC Compiler version B-2008.09 or earlier cannot read Milkyway database schema version 4.0.

You can convert a database of schema version 4.0 to schema version 3.2 using IC Compiler version C-2009.06 using the `save_mw_cel -previous` command or the `convert_mw_lib -previous` command. You can convert databases of schema version 3.2 and earlier to schema version 4.0 using the `convert_mw_lib` and the `open_mw_lib` commands. When you convert a database from an earlier version to the latest version, the CEL and the ILM views of the design database are converted. FRAM views are not converted. For more information about how to convert the database schema versions, see “Converting a Milkyway Database Model” on page 2-8.

Specific changes regarding Synopsys tools, including IC Compiler, for the C-2009.06 Milkyway database are as follows:

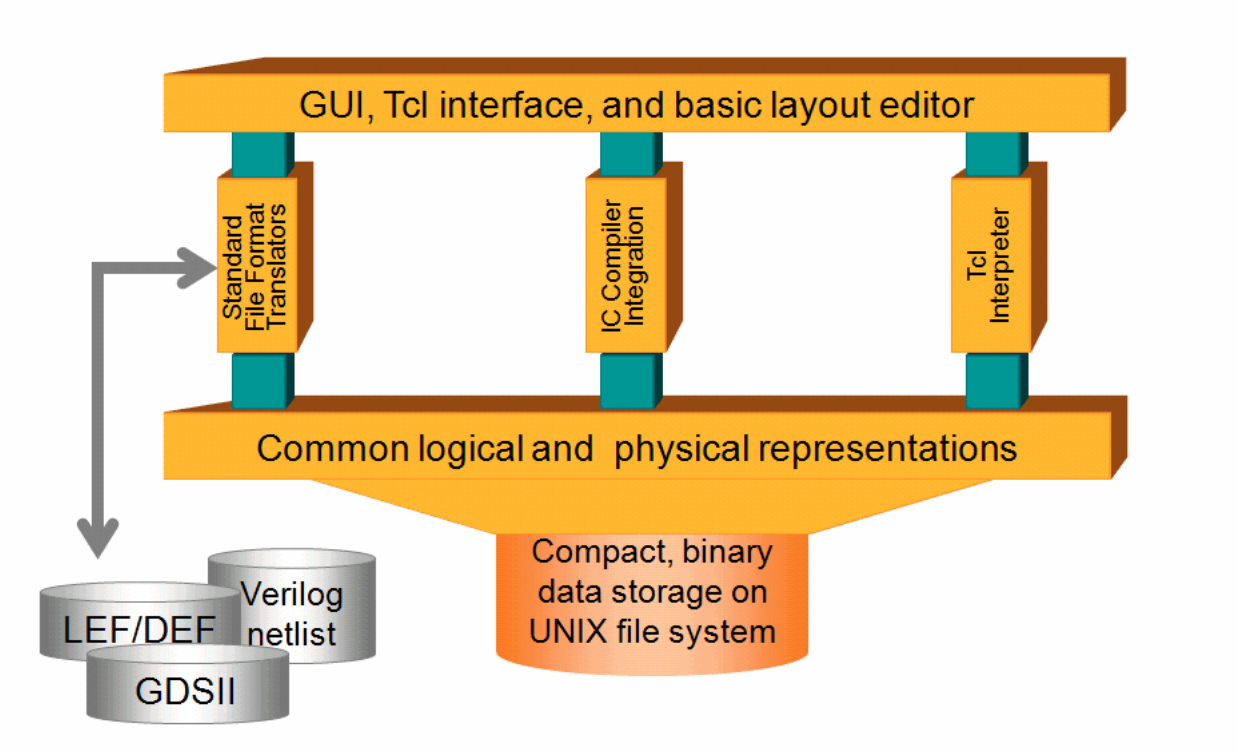
- When necessary, the Milkyway design database is automatically converted to ensure integrity of the data.
- After conversion, you must save the Milkyway design database to the disk to retain the latest version of the Milkyway database.

- Physical Libraries are not affected for the Milkyway C-2009.06 database.
- All commands of IC Compiler including GUI and collections are updated to ensure that they work seamlessly with the new Milkyway database schema.

The Milkyway Environment

Figure 1-1 describes the basic components of the Milkyway Environment, which provides file translators for common file formats such as GDSII and DEF. You access these translators through the Milkyway GUI or through Tcl commands. Synopsys recommends that you use Tcl rather than Scheme commands for long term script support. Also, Tcl supports the latest technologies such as 45 nm design rules.

Figure 1-1 The Milkyway Environment



You use the Milkyway Environment for library and data preparation. The Milkyway Environment is integrated with IC Compiler, and IC Compiler reads and writes directly to the Milkyway database for place and route operations. Some commands exist in both Milkyway and IC Compiler. If a command is available in both Milkyway and IC Compiler, such as `check_library`, use the version of the command in IC Compiler.

Milkyway stores data in binary format in a UNIX file system. Although you can access design data from a UNIX shell, it is recommend that you do not change any data directly. This is because Milkyway runs background processes to maintain the integrity of your design data. Changing data manually at the UNIX file system level can compromise your design data's integrity.

Before you exit the Milkyway Environment, you need to close your design and library. By following this procedure, Milkyway can run a database check to clean out temporary working files. Exiting before closing a design and library can result in data corruption and unnecessary temporary files in the working directory.

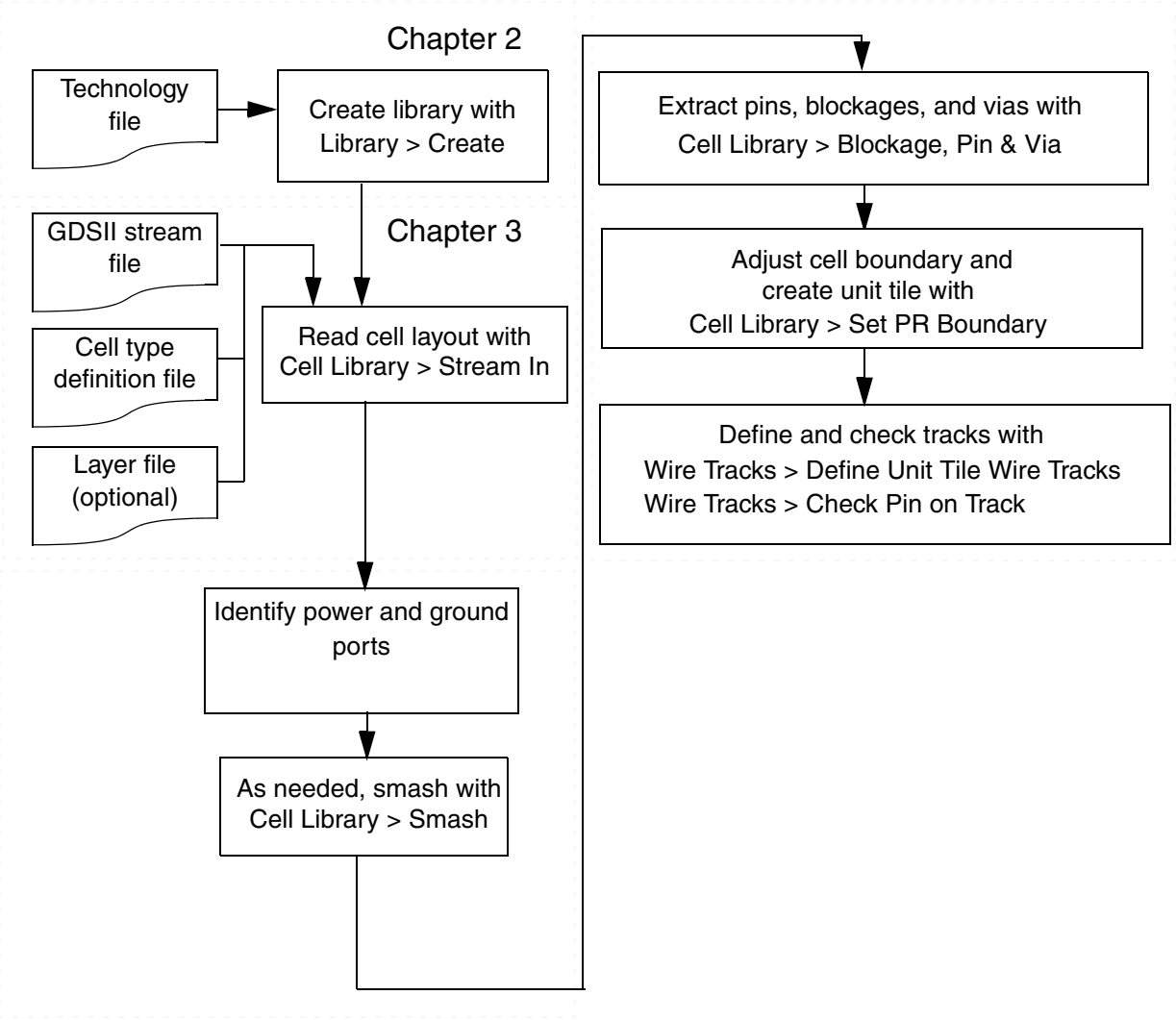
The Milkyway tool and the data preparation functions can perform several tasks essential for cell library and data preparation, including the following:

- Creating cell libraries
- Importing cell 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

Data Preparation Flow

Figure 1-2 describes a basic data preparation flow using the Milkyway Environment tool to create and load a library for placement and routing in IC Compiler. These steps are documented in the subsequent chapters and appendixes.

Figure 1-2 A Basic Data Preparation Flow



Starting Milkyway

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

```
% Milkyway -galaxy &
```

Note:

Although the Milkyway Environment can be opened in any directory, it is better to open it in the directory containing your design data files. That way you will be able to more easily access libraries and design cells and the created .cmd and .log files.

Exiting Milkyway

Before you exit the Milkyway Environment, close and save your edited design and library.

To exit Milkyway, Choose Tools > Quit. A dialog box asks if you really want to quit. Click OK.

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

AServer and 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 this 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 of the `Milkyway` command. 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 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 -galaxy -tcl -file tcl_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 -galaxy -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 (.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 Milkyway 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, you must create a Milkyway library in which to store the design data. The Milkyway Environment tool enables you to create and load libraries for this design data. A Milkyway library can contain both top-level designs and the cells used in those designs.

This chapter contains the following sections:

Milkyway Libraries

The Milkyway database consists of libraries that contain information about your design. Libraries contain information about design cells, standard cells, macro cells, and so on. They contain physical descriptions, such as metal, diffusion, and polysilicon geometries. Libraries also contain logical information (functionality and timing characteristics) for every cell in the library. Finally, libraries contain technology information required for design and fabrication.

Milkyway provides two types of libraries: reference libraries and design libraries. Reference libraries contain standard cells and hard or soft macro cells, which are typically created by vendors. Reference libraries contain physical information necessary for design implementation. Physical information includes the routing directions and the placement unit tile dimensions (the width and height of the smallest instance that can be placed).

A design library contains a design cell. The design cell might contain references to multiple reference libraries containing standard cells and macro cells. Also, a design library can be a reference library for another design library.

The Milkyway library is stored as a UNIX directory with subdirectories, and every library is managed by the Milkyway Environment. The top-level directory name corresponds to the name of the Milkyway library. Library subdirectories are classified into different views containing the appropriate information relevant to the library cells or the designs.

In a Milkyway library there are different views for each cell, for example, NOR1.CEL and NOR1.FRAM. This is unlike a .db formatted library where all the cells are in a single binary file. With a .db library, the entire library has to be read into memory. In the Milkyway Environment, the Synopsys tool loads the library data relevant to the design as needed, reducing memory usage.

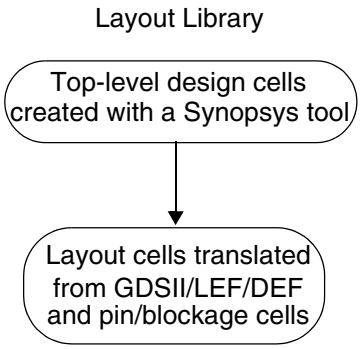
The following are commonly used Milkyway views:

- CEL – The full layout view
- FRAM – The abstract view for place and route operations

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 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.

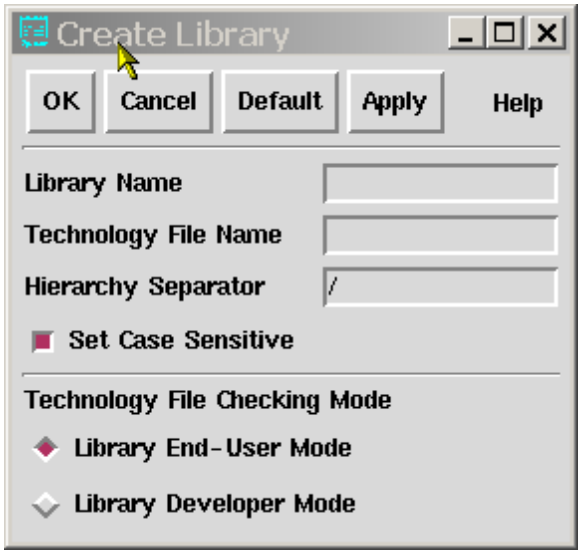
Creating a Library

You can create a library with Milkyway. Before you can create a library, you need a technology file.

To create a new library,

1. 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 2-5.

   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. Toggle on the Set Case Sensitive option. (IC Compiler supports only case-sensitive Milkyway libraries.)
6. Click OK.

Naming Conventions for Milkyway Library Cells

The Milkyway database 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 checks the technology file for errors. You have two checking levels to choose from:

- Library End-User Mode

  This is the default level. 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, this mode is 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.

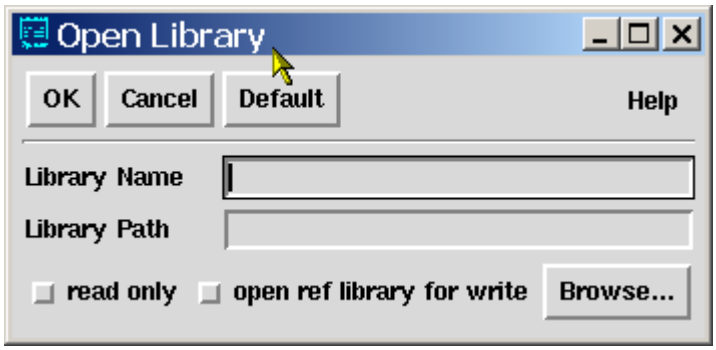
Opening a Library

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

To open an existing library,

1. Choose Library > Open.

   The Open Library dialog box appears.



2. Fill in the Open Library dialog box.
3. Click OK.

Referencing Cells in Another Library

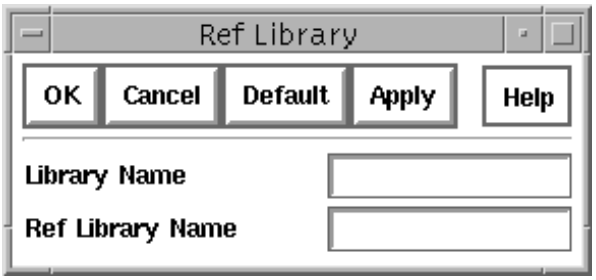
You can work in only one library at a time. However, you can access cells in another library by making a reference library of the library you are working in. When working with reference libraries, you should understand the following:

- 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. Choose Library > Add Ref.

The Ref Library dialog box appears.



2. Fill in the Ref Library dialog box.
3. Click OK.

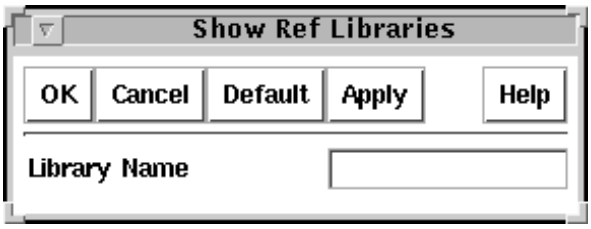
   The libraries become associated so that anytime you open the library specified by Library Name, you can access cells in the library specified by Ref Library Name.

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

To list the reference libraries associated with a library,

1. Choose Library > Show Refs.

   The Show Ref Libraries dialog box appears.



2. Enter the library name.

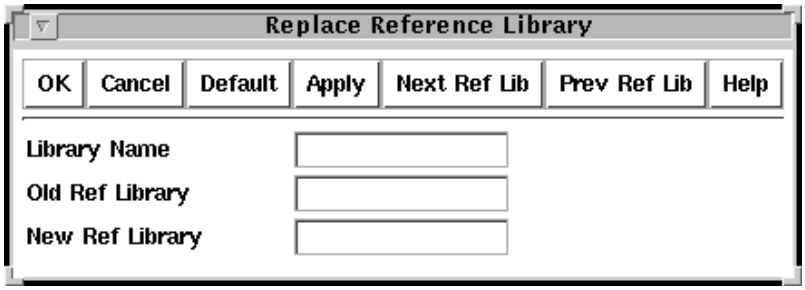
   This is the name of the library for which you want to list reference libraries.
3. Click OK.

   The reference libraries are listed in the message area.

To change a reference library associated with a library,

1. Choose Library > Replace Ref.

   The Replace Reference Library dialog box appears.



2. Fill in the Replace Reference Library dialog box.
3. Click OK.

Converting a Milkyway Database Model

To work with Synopsys tool version C-2009.06, existing Milkyway design libraries must be converted to the new database model. Before converting a Milkyway design library, you should create a backup copy of the design library with the previous database model.

Caution:

You cannot use designs saved with the new Milkyway database model in Synopsys tools with a version prior to B-2008.09, such as Astro, Jupiter, or older versions of IC Compiler.

There are two ways to convert a Milkyway design to the new database model:

- Manual conversion by using the `convert_mw_lib` command
- Automatic conversion by using the `open_mw_cel` command

These commands are available in both IC Compiler and the Milkyway Environment.

Note:

Both the `convert_mw_lib` and the `open_mw_cel` commands convert the specified design but not the designs in the Milkyway reference libraries, unless they are used in the Milkyway design library. The `convert_mw_lib` command does not convert the child macro CELs that are in the reference library, even if they are referenced by the top level design. The CEL views and the Interface Logic Model (ILM) views of the design are converted. Database model conversion does not affect FRAM view cells.

If you need to convert a Milkyway database model from version C-2009.06 to version B-2008.09, when the design has a wire master index greater than 255, use the `remove_route_by_type` command to delete all wires before converting the design. You must perform routing on the design again after the database model has been converted.

Manual conversion

To minimize the runtime and memory consumption due to automatic database model conversion when opening a design, you should manually convert designs to the new database model. To manually convert a design to the new database model, use the `convert_mw_lib` command in either IC Compiler or the Milkyway Environment.

You can either convert all designs in the Milkyway design library, using the `-all` option, or you can specify a design to convert, using the `-cell_name` option. The `convert_mw_lib` command also converts all of the opened design's child macro cells that are in the same design library. It does not convert the child macro cells that are located in a reference library. The `convert_mw_lib` command opens the design, converts its database model, and saves it with an incremental version number.

If the top-level design refers to CEL or ILM views of the design in the reference library:

- The views that are loaded are converted in memory
- The converted views of the design are not saved to disk
- The ILM views generated using Synopsys tool version B-2008.09 and its service packs need not be re-created in tool version C-2009.06

Note:

ILMs that are created using versions earlier than Synopsys tool version B-2008.09 are not converted. CEL views created using Synopsys tool version A-2007.12 or earlier can be converted in IC Compiler version C-2009.06.

To save time when converting a Milkyway design library to the new database model, use the `remove_mw_cel` command to remove cells in your design library that are no longer required before you run the `convert_mw_lib` command.

The syntax of the `convert_mw_lib` command is

```
convert_mw_lib mw_lib -all | -cell_name mw_cell
```

where

`mw_lib`

Specifies the Milkyway design library containing the cells to be converted.

`-cell_name` *`mw_cell`*

This option converts the specified cell and saves it. Note that the name of the view must not be specified in this option. To convert a specific view, such as *`cell_name`*`.ilm` or *`cell_name`*`.cel`, use the `open_mw_cel` command.

`-all`

This option converts all the cells in the design library.

You must specify either the `-cell_name` or the `-all` option when you run the `convert_mw_lib` command.

For example, to convert all designs in the `my_mw_lib` Milkyway design library to the new database model, enter the following command:

```
prompt> convert_mw_lib my_mw_lib -all
```

Automatic conversion

When you use the `open_mw_cel` command to open a Milkyway design, it checks the database model version and automatically updates the database model, if necessary.

When opening a hierarchical design using the `open_mw_cel` command, if the current cell contains references of soft macro cells in the design library or reference libraries, the MW-365 warning message is displayed. The message shows the current cell model and its child macro cells, and recommends using the `convert_mw_lib` command to convert the design.

The `open_mw_cel` command does not save the converted design. To save the converted design, you must use the `save_mw_cel` command. To keep a copy of an older data model when using the `open_mw_cel` command, use the `save_mw_cel -as` *`name`* command. If you do not save the design, the next time you use the `open_mw_cel` command to open the design, it automatically converts the design again. Use the `-as` *`name`* option or the `-increase_version` option to prevent overwriting the current design.

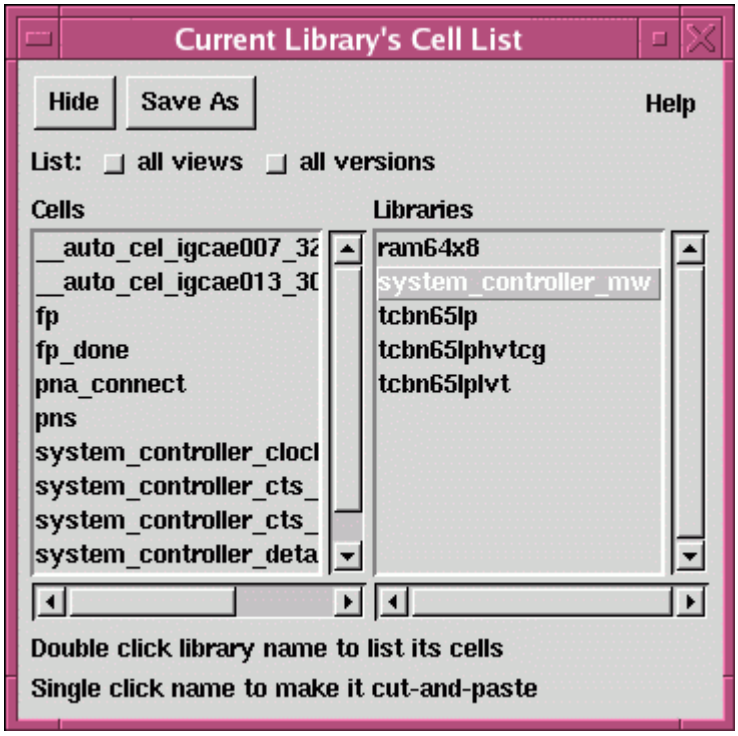
Listing the Cells in a Library

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. Choose Library > Show Cells.

   This opens the Current Library's Cell List.



2. In the Libraries list, double-click the name of the library of interest. The cells in that library are shown in the Cells list.
3. Use the scroll bars to view the list.

   You can optionally save the list of cells to a file by using the Save As button.
4. Click the Hide button to close the Current Library's Cell List.

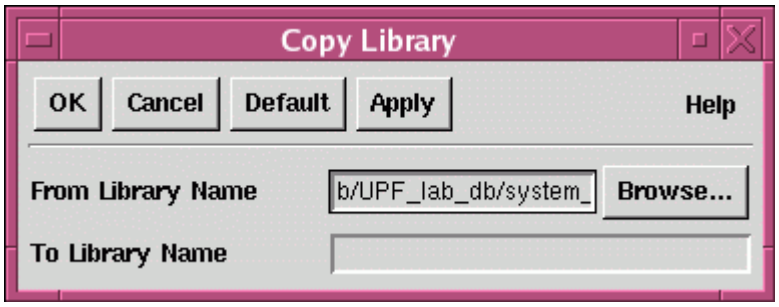
Copying a Library

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. Choose Library > Copy.

   The Copy Library dialog box appears.



2. Fill the From Library Name with the name of the library to copy. If necessary, use the Browse button to find the desired library.
3. Fill the To Library Name with the name of new library to create by copying.
4. 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. 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.

Protecting Library Technology Information

You can encrypt the library technology using the `encrypt_lib` command. This is useful when you need to hide the technology file when providing a Milkyway library to another party.

If the library technology is encrypted by the `encrypt_lib -key` *my_key_string* command, you can unlock the library using the `decrypt_lib -key` *my_key_string* command.

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.

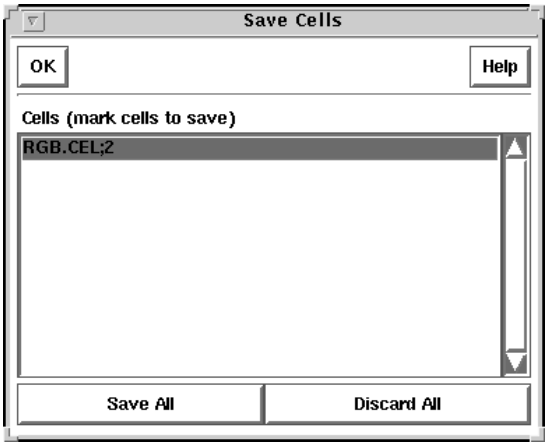
To close a library,

1. Choose Library > Close.

   A confirmation dialog box appears.

2. Click OK to confirm.

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



3. Select the cells you want to save from the list of cells. Click Save All to highlight all the cells in the list or Discard All to un-highlight all of them.
4. Click OK to save the selected cells and close the library.

3

GDSII and OASIS Data Preparation

To prepare your data, you must first import the physical cells into one or more libraries by translating GDSII stream data. The steps for importing the physical cells are covered in the following sections:

- GDSII Stream Data Translation
- Cell Type Specification
- Layer Map Specification
- The Milkyway OASIS Interface

GDSII Stream Data Translation

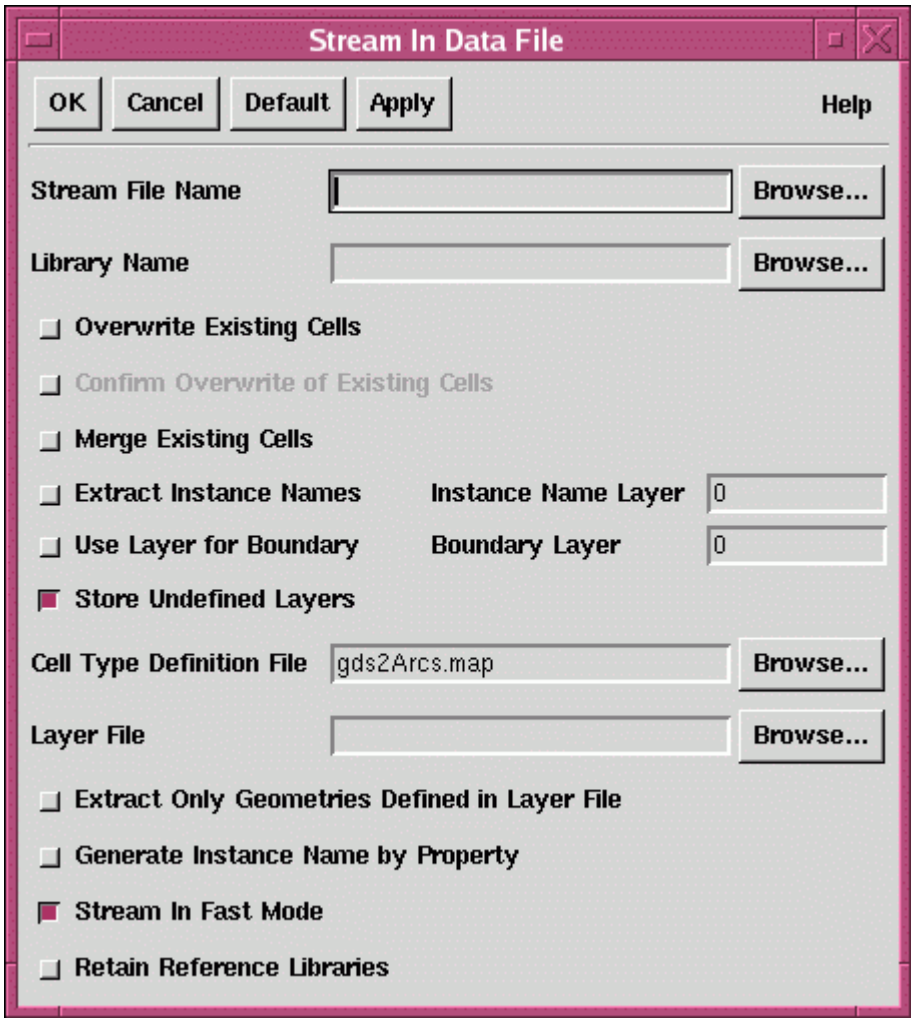
After you have the input files you need, you are ready to translate your GDSII Stream files. Using Layout > Stream In, you can translate all or part of the cells in the layout.

To import a library in GDSII Stream format,

1. 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 a Cell Type Definition File a Layer File in the fields provided. These files define the cell types and layer mapping as described in the following sections, "Cell Type Specification" and "Layer Map Specification." For detailed descriptions of the command options, see the Online Help.
3. Click OK.

Milkyway translates the cells, creates a Synopsys layout cell for each cell, and adds them to the specified library.

If a cell type is marked as "other," Milkyway displays a message to let you know that a cell type is not set for the cell. Milkyway ignores a cell with no cell type in further processing.

Cell Type Specification

Milkyway allows you to map cell types to cells in the layout by creating a cell type definition file. GDSII Stream files do not designate the type of a cell and this information might be needed for specific tools. After creating this file, you specify it in the Cell Type Definition File box as described in the next section, "GDSII Stream Data Translation" on page 3-2."

This text file contains cell type statements in the following format:

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

where

cellType is a keyword 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.
- `gdsTapCell` or `tapCell` marks the specified cells as tap cells, which are physical-only cells that have power and ground pins only and do not have signal pins.

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.

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 Cell Library > 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.

Layer Map Specification

The Stream In Data File dialog box (Cell Library > Stream In) and the Stream Out Data File dialog box (Output > Stream Out) provide options for specifying a layer mapping file to manipulate the stream data. If you are using the `read_gds` or `write_gds` Tcl command, you can specify the layer file in the command line by using the `-layer_mapping` option.

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 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 “GDSII Stream Data Translation” on page 3-2). To control the mapping from GDSII layers to Milkyway database layers, 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 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 in the mapping file 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. 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 specify 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, `auStreamIn` will, by default, convert both data types to the same layer. To translate layers differently, 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 geometry from being streamed in by selecting the layer mapping file and deselecting Store Undefined Layers. 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, deselect Store Undefined Layers in the Stream In Data File dialog box and click OK. 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][PinCode]
  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
`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. A for all net types. 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.

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

Variable	Description
	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.
`PinCode`	Optional. Can be used when only the Pin geometry, or Terminal (top level pin) geometry in the Milkyway database needs to be translated. Valid types: T for terminal geometries of the current design P for terminal geometries of the current design, as well as the physical pin geometries of the child cells depends on the `-child_depth` option. Note: PinCode P and T only work when MilkywayNetType is set to A. If MilkywayNetType is not A, the PinCode rule will be ignored.
`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
`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. The optional third character specifies the pin code. 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
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 using the menu option Cell Output > Stream Out, the contact or contact array is 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.

The Milkyway OASIS Interface

OASIS stands for Open Artwork System Interchange Standard. It defines an interchange and encapsulation format for hierarchical integrated circuit mask layout information. The OASIS reader and writer allow you to annotate the Milkyway database with physical information. As a replacement for GDSII, OASIS removes all 16- and 32-bit integer width restrictions.

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*.

Writing GDSII and OASIS From IC Compiler

You can write a design in GDSII or OASIS format from the Milkyway database by using commands in IC Compiler. You set the options for layout file generation by using the `set_write_stream_options` command and generate the GDSII or OASIS file with the `write_stream` command in IC Compiler.

The `set_write_stream_options` command lets you set many file generation options such as naming conventions, name mapping, hierarchical depth, via flattening, and file compression. You can use the command multiple times to incrementally set individual options. For full details, see the man page for the `set_write_stream_options` command.

After you set the options, you use the `write_stream` command to write out the design in GDSII or OASIS format. This is the syntax of the command:

```
write_stream
 [-lib_name lib_name]
 [-format gds | oasis]
 [-cells list_of_cells]
 [-cell_file cell_name_file]
 stream_file_name
```

You must specify at least the output file name. The default format is GDSII, so to write the design in OASIS format, you must use the `-format oasis` option. For example,

```
icc_shell> write_stream -format oasis my_design.oasis
```

By default, the command writes out all cells in the currently opened library. To restrict the output to a specified library or specified cells, use the `-lib_name`, `-cells` or `-cell_file` option. For more information, see the man page for the `write_stream` command.

IC Compiler supports only the writing, not the reading, of GDSII and OASIS files.

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 a Milkyway database. LEF defines the elements of an IC process technology and the associated library of cell models, and it 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 5.7.

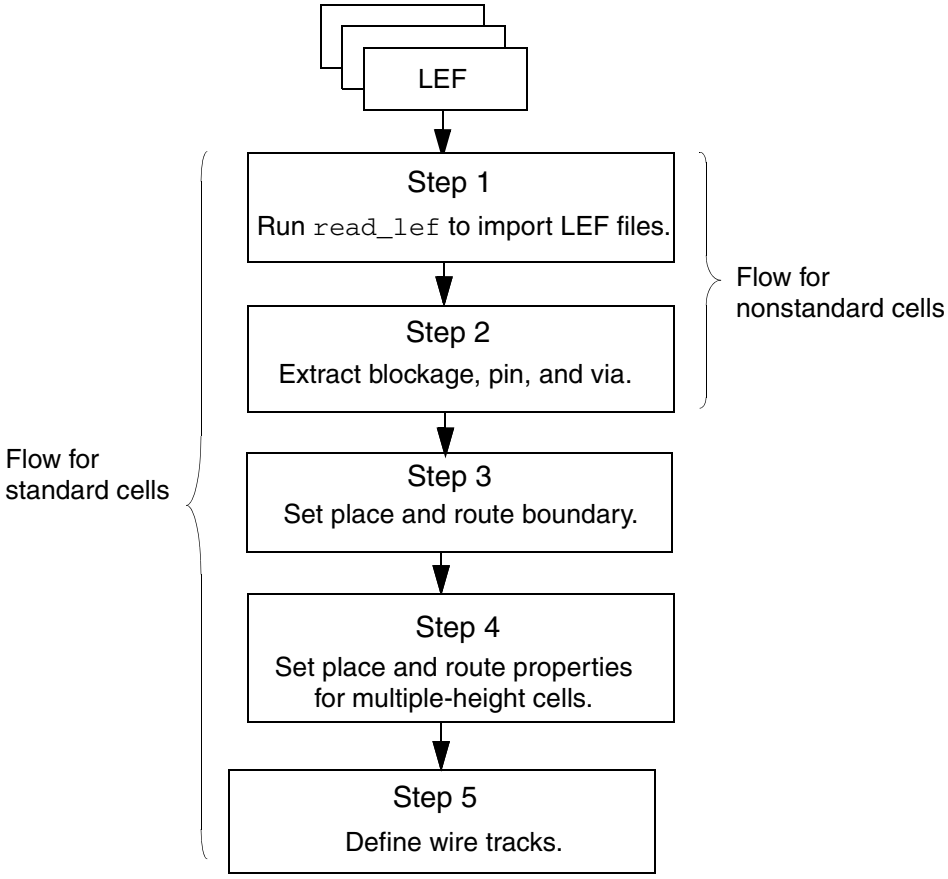
The commands for importing LEF data are `read_lef` (automated flow) and `read_lib` (advanced flow) for standard and nonstandard cells. Both commands provide complete translation for LEF syntax.

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
- Exporting LEF Data From Milkyway Using write_lef

Figure 4-1 shows an overview of the Milkyway library preparation flow using the `read_lef` and `read_lib` commands. 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 Milkyway reference library, ready for place and route, 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.

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

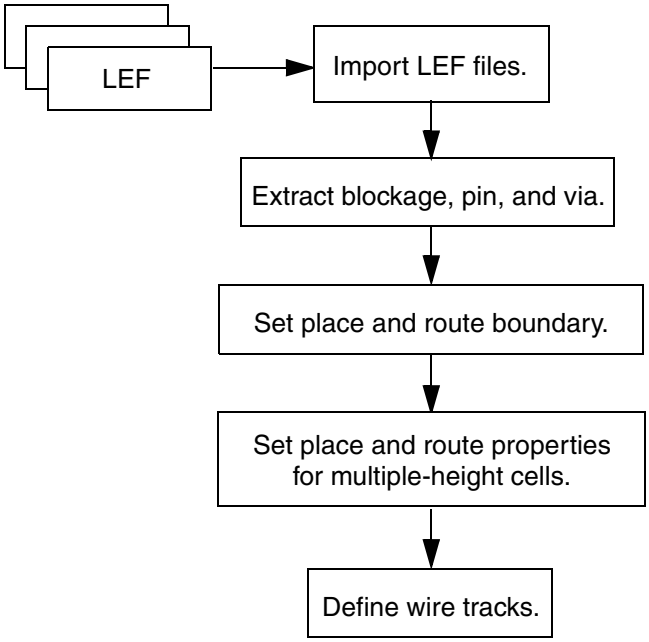
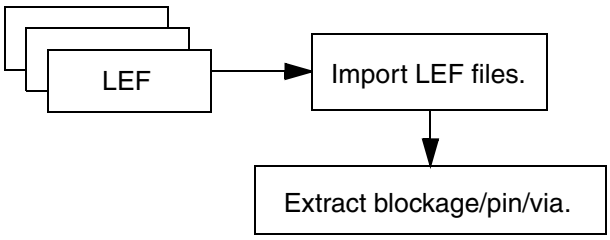


Figure 4-3 shows the `read_lef` flow for nonstandard cells using a two-step automated process. Note that the `read_lef` flow runs through *all* steps shown in Figure 4-2. Using the automated `read_lef` flow for nonstandard cells, you can ignore any error messages that might be reported in the last three 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

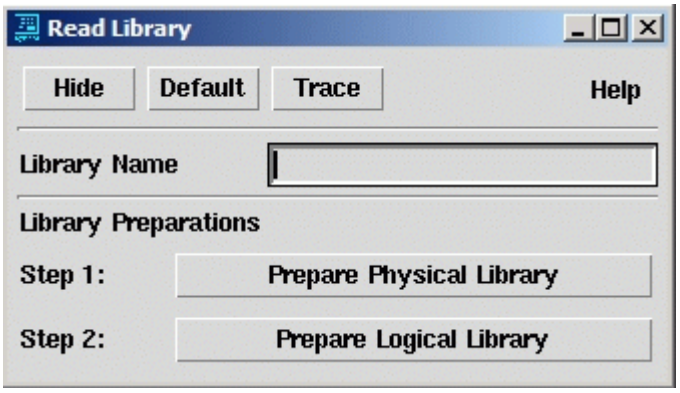


Overview of read_lib

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.

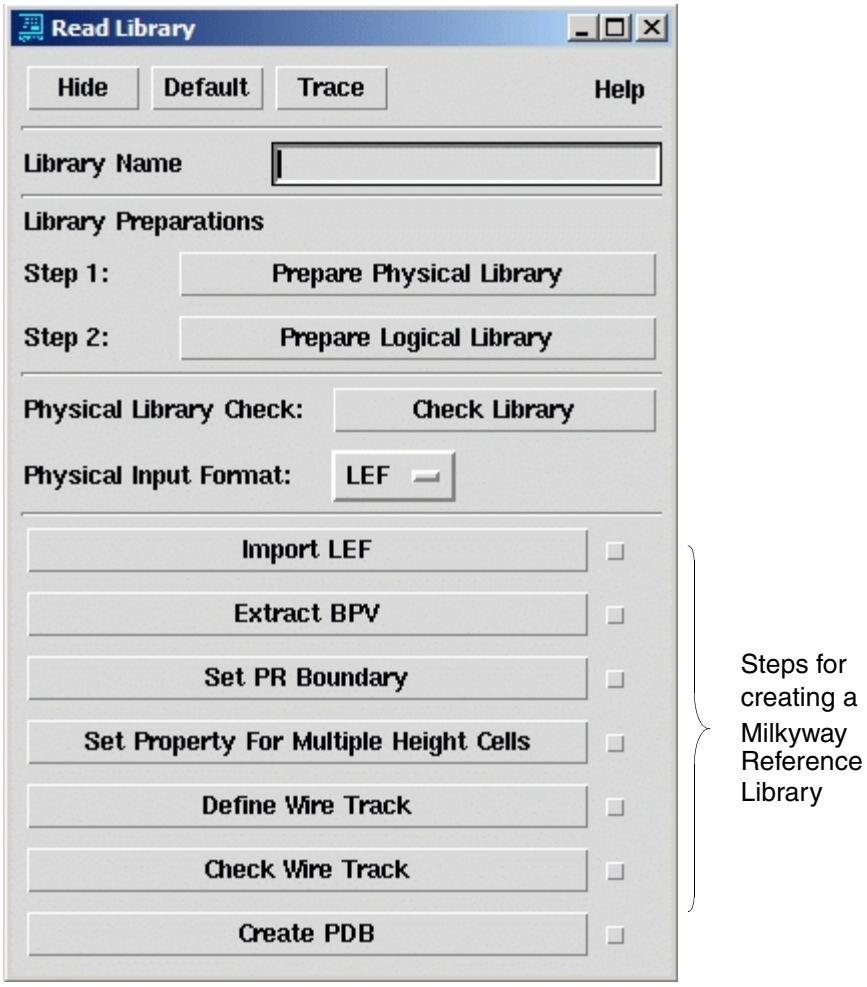
Figure 4-4 Initial Read Library Dialog Box



Alternatively, you can select the Advanced Library Prep Mode option in the Read LEF dialog box, shown in Figure 4-6.

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 4-5. The steps for using `read_lib` are covered in "Creating a Milkyway Library Using read_lib" on page 4-12.

Figure 4-5 Read Library Dialog Box



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 4-12.)

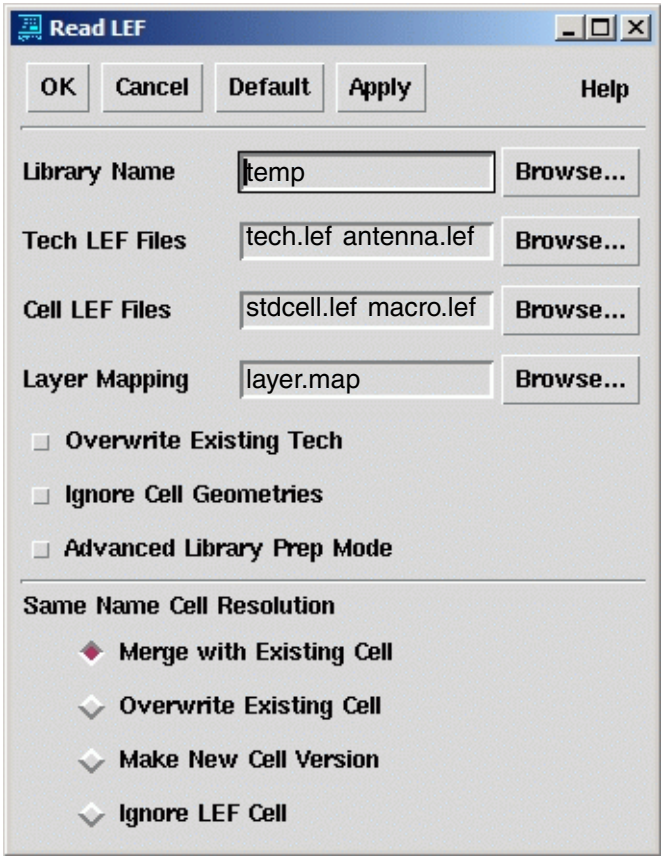
Step 1: Importing LEF Files

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 4-6 shows the Read LEF dialog box with a library named temp.

Figure 4-6 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, which adds new cells to the existing library. 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:

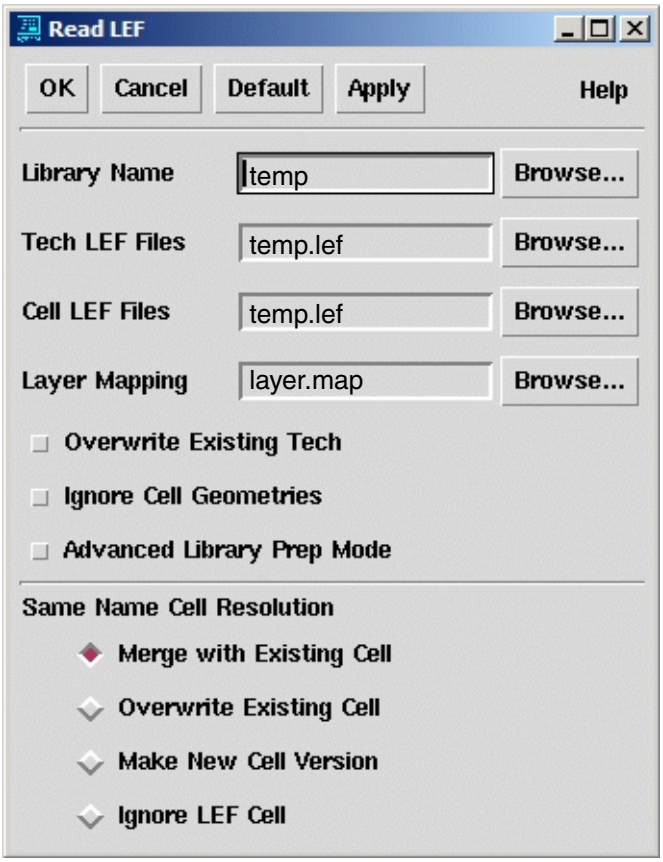
If you select Advanced Library Prep Mode in the Read LEF dialog box and click Apply, Milkyway opens the Read Library dialog box. When you click OK in the Read LEF dialog box, Milkyway completes the first step (Import LEF) and the status check box next to Import LEF in the Read Library dialog box is selected. 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 4-12.

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. For example, 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 4-7.

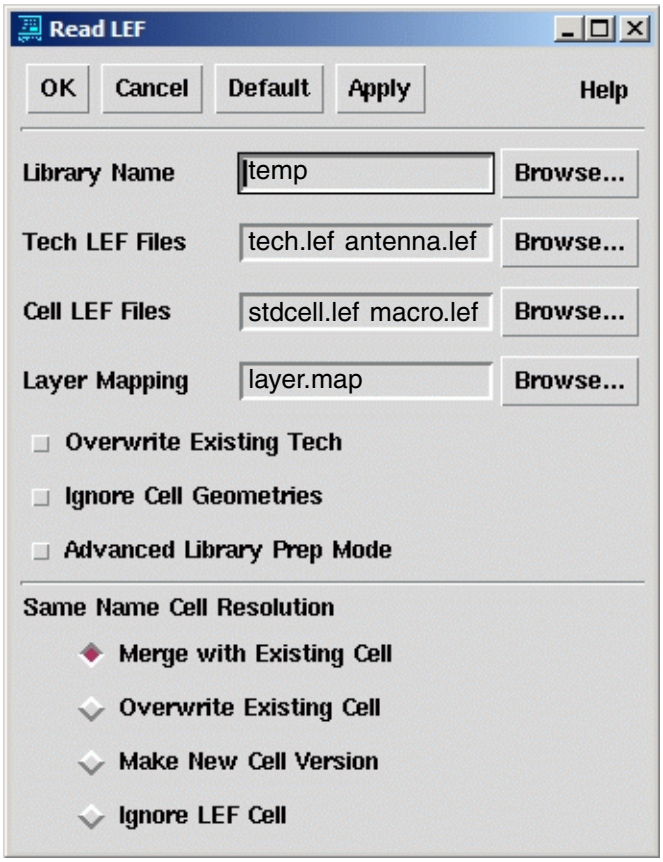
Figure 4-7 Read LEF for Single Tech and Cell LEF Files



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 4-8. Use a blank space or comma as a separator when specifying multiple files.

Figure 4-8 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, not the cell-level information. Similarly, only the cell-level information from the files specified in the Cell LEF Files box is used, not the technology information.

3. Use the Layer Mapping box to specify a layer mapping file that contains the LEF layer names and corresponding Milkyway layer numbers.

   Although specifying a layer mapping file is optional, it is strongly recommended. If no file is specified, Milkyway follows its internal default order, with the first layer in LEF mapping to layer 1 in Milkyway, 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 to create a layer mapping file. This is the syntax:

   ```
   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.

For a current version of the Perl script, see the SolvNet article “Using the lef_layer_tf_number_mapper.pl script.”

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 numbers. Without this mapping, Milkyway assigns default layer numbers for you.

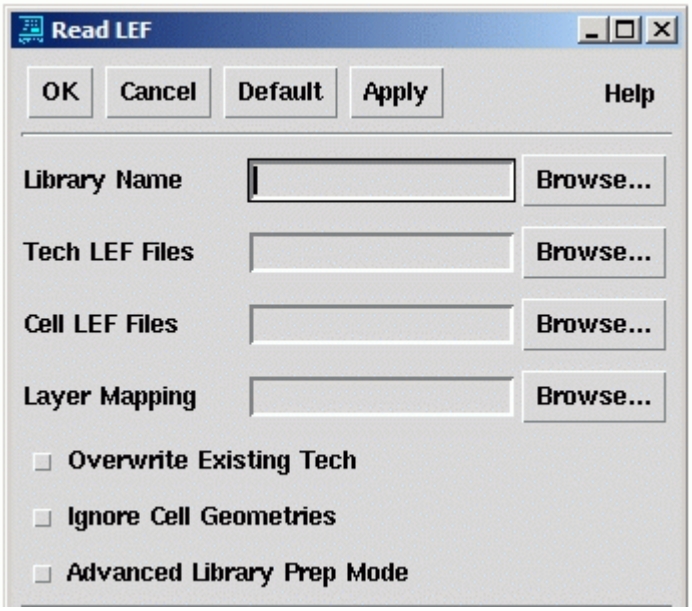
4. Click OK.

   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 4-9 shows the main options, followed by a brief description of each option.

Figure 4-9 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. If the library does not already exist, the `read_lef` command creates a new reference library based on LEF information.

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 names and corresponding Milkyway layer numbers. 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, the second layer in LEF mapping to layer 2 in Milkyway, and so on.

Overwrite Existing Tech (Optional)

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)

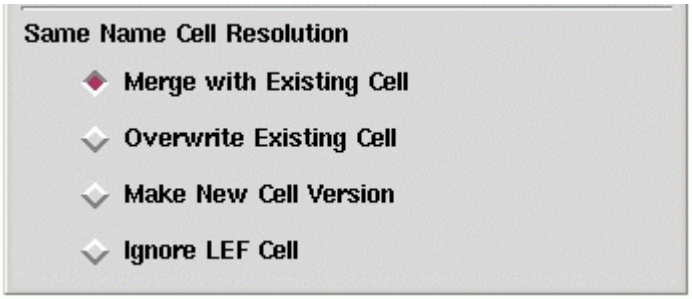
When you select this option, the pin, rectangle, polygon, and via objects are not created. After LEF input, you must input the GDSII data to create the physical information. This option is typically used for the Tech LEF + Physical Cell GDS flow.

Advanced Library Prep Mode (Optional)

By default, this option is not selected and the `read_lef` command automatically completes all library preparation steps and creates a Milkyway library ready for place and route. If this option is selected, you manually run these steps, including 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 4-10 shows the Same Name Cell Resolution options, followed by a brief description of each option.

Figure 4-10 Read LEF Dialog Box: Same Name Cell Resolution Options



Merge with Existing Cell

When selected, the latest cell version will include both 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

If this option is selected, the latest cell versions will be overwritten by the LEF macro cells.

Make New Cell Version

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

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.

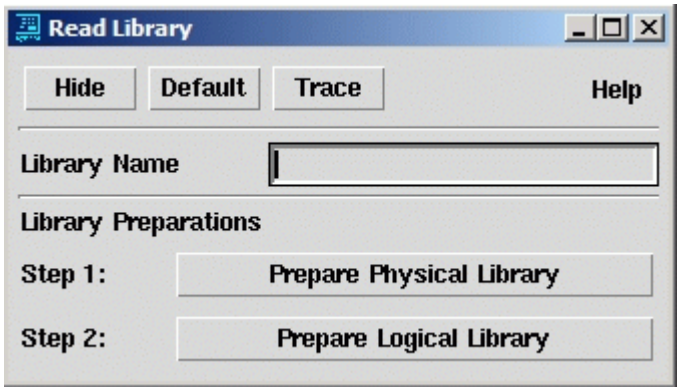
Note that if you select the Advanced Library Prep Mode option in the Read LEF dialog box and click Apply, Milkyway opens the Read Library dialog box. When you click OK in the Read LEF dialog box, Milkyway completes the first step (Import LEF) and the status button next to Import LEF in the Read Library 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:

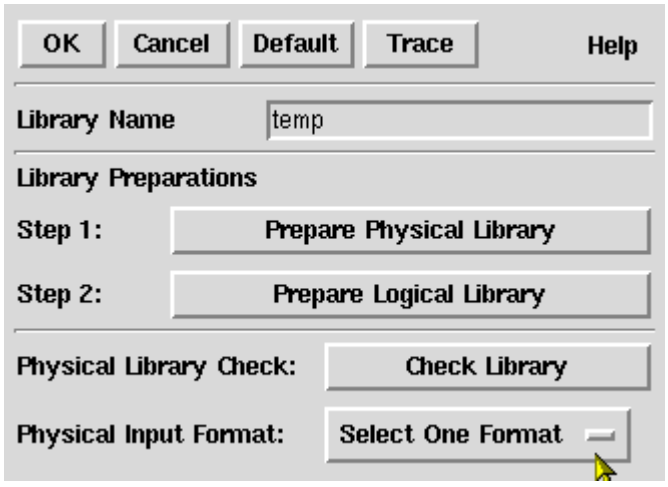
Step 1 (importing LEF files) has the same default options for `read_lib` as it does for `read_lef`.

Step 1: Importing LEF Files

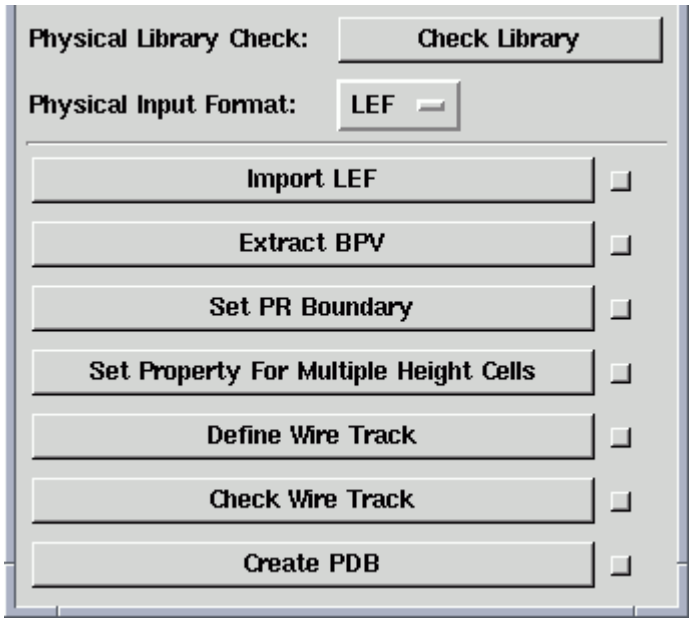
1. Invoke the Read Library dialog box, by entering `read_lib` on the Milkyway command line. This command guides you through the steps required to create a Milkyway library ready for place and route.



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.



5. Click Import LEF. In the Read LEF dialog box, fill in the options as described previously. Click OK.

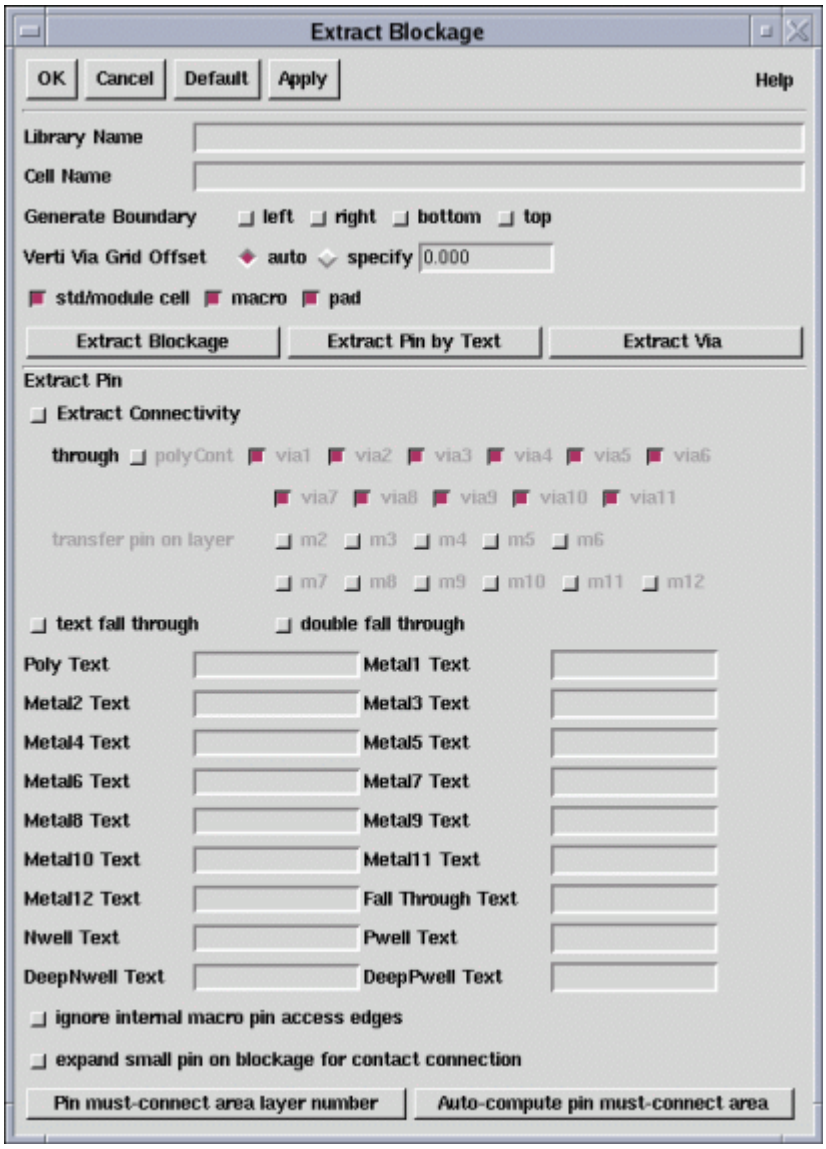
Step 2: Extracting Blockage, Pin, and Via and Generating FRAM View

The `read_lef` command automates the steps required to create a Milkyway library that is ready for place and route. 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. 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 4-11. Electrical equivalence for pins (EEQ) information is extracted, based on the EEQ information provided from the LEF file.

Figure 4-11 Extract Blockage Dialog Box



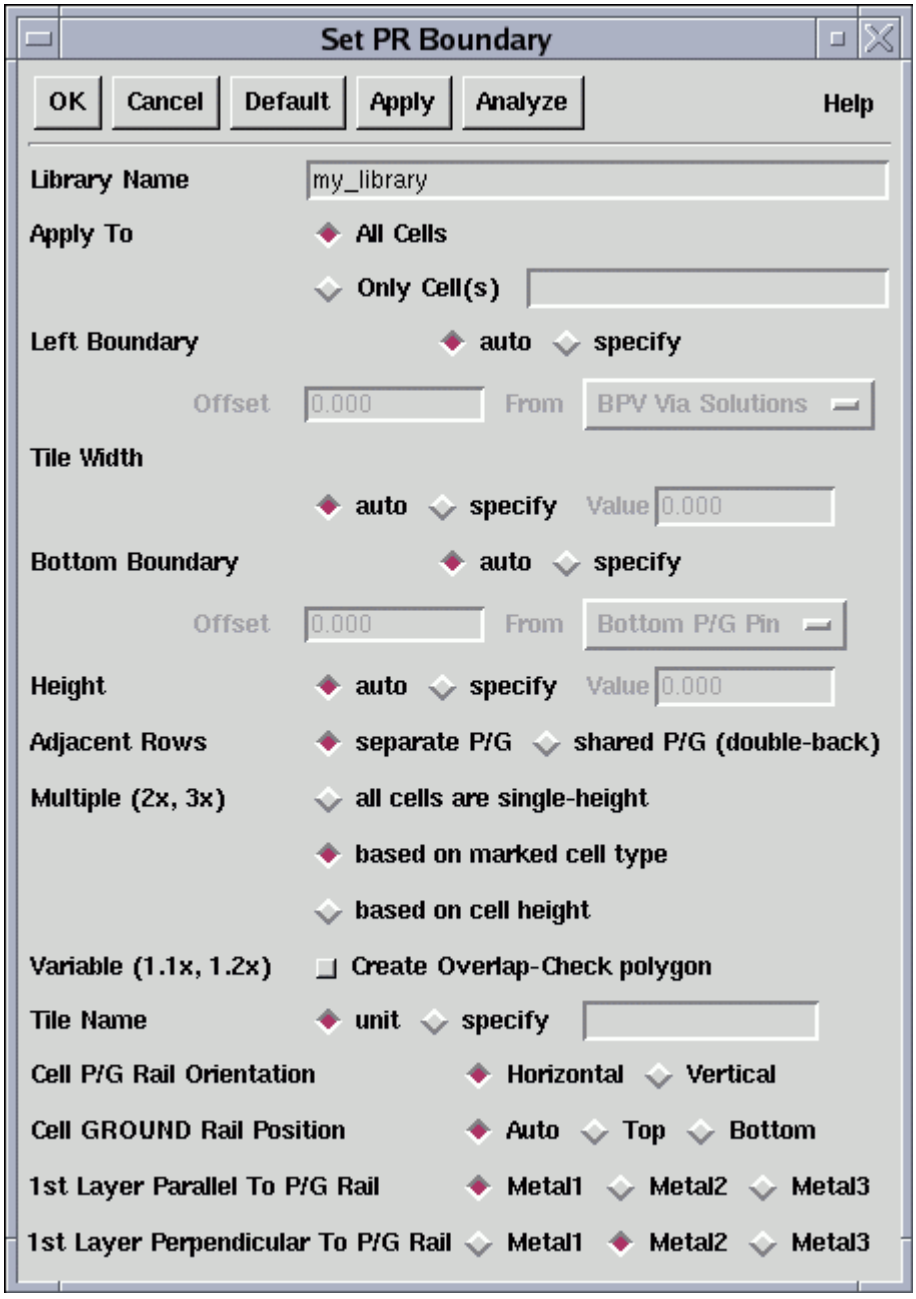
2. Specify the library name if it is not provided.
3. Click OK.

Step 3: Setting the Place and Route Boundary

1. Choose 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 4-12.

Figure 4-12 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.

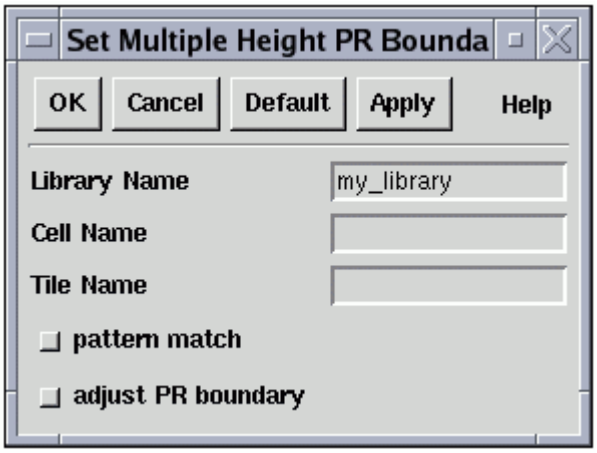
2. Select the options you want.
3. Click OK.

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

1. Choose 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 4-13.

Figure 4-13 Set Multiple Height PR Boundary Dialog Box



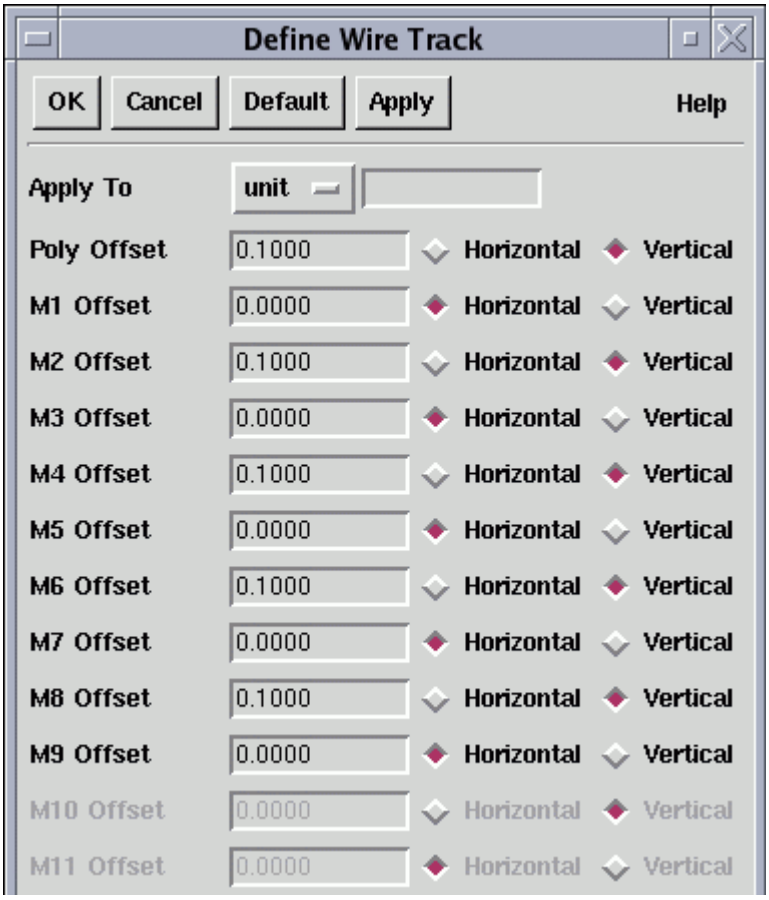
2. Click OK.

Step 5: Define Wire Tracks

1. Choose 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 4-14.

Figure 4-14 Define Wire Track Dialog Box



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

2. Click OK.

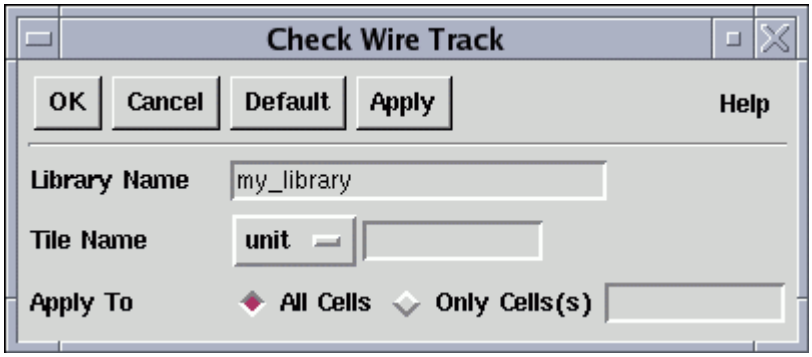
Step 6: (Optional) Check Wire Tracks

1. Choose 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 4-15 shows the Check Wire Track dialog box.

Figure 4-15 Check Wire Track Option



If you see pins without a desirable access point on the Grid, redo Define Wire Track (modify the routing layer's offset). It may be 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 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."

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 the Read LEF dialog box 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.

Exporting LEF Data From Milkyway Using write_lef

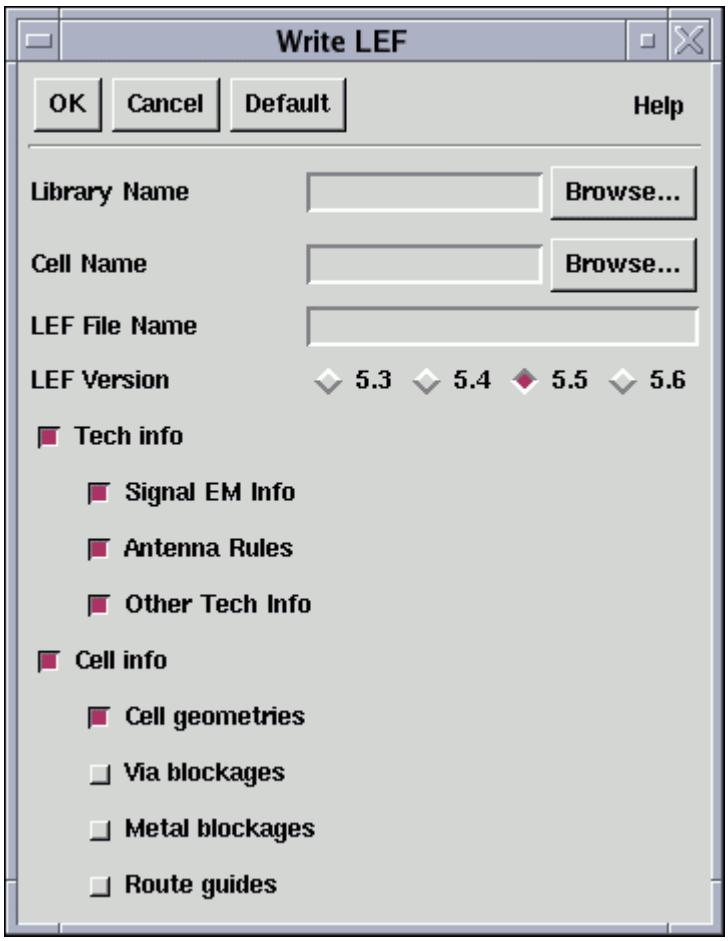
This section discusses the Write LEF dialog box and the options used to export LEF data from the Milkyway environment. (For information about the Read LEF dialog box and the options used to import LEF data into the Milkyway environment, see "Library Preparation Using LEF" on page 4-2.)

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.

To open the Write LEF dialog box, choose Output > LEF Out.

The Write LEF dialog box opens, as shown in Figure 4-16.

Figure 4-16 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 LEF interface supports versions 5.3, 5.4, 5.5, and 5.6. The default is 5.5.

Tech info (Optional)

In the default mode, Milkyway writes a technology LEF file. You can also specify whether to write out signal electromigration information, antenna rules, and other technology information in the library.

Cell info (Optional)

In the default mode, Milkyway writes a LEF file with cell information. You can also specify whether to write out cell geometries, via blockages, metal blockages, and route guides.

The Verilog Interface

When you are ready to read in a design in Verilog format, open the Milkyway design library and then read in the Verilog netlist file for the design.

- Read the Verilog netlist file for the design by using the `read_verilog` command (or by choosing Netlist In > Verilog To CEL).

  The Verilog netlist file is a structural or gate-level design in one file.

  ```
  Milkyway> read_verilog -dirty_netlist design.v
  ```

- Write Verilog data for the design by using the `write_verilog` command (or by choosing Cell > Hierarchical Verilog Out).

  ```
  Milkyway> write_verilog design_name.v
  ```

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 reader and writer is consistent with those used in other Synopsys tools.

The DEF interface supports versions 5.3, 5.4, 5.5, 5.6, and 5.7. 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 Versions 5.6 and 5.7 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 gzip 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 4-32.

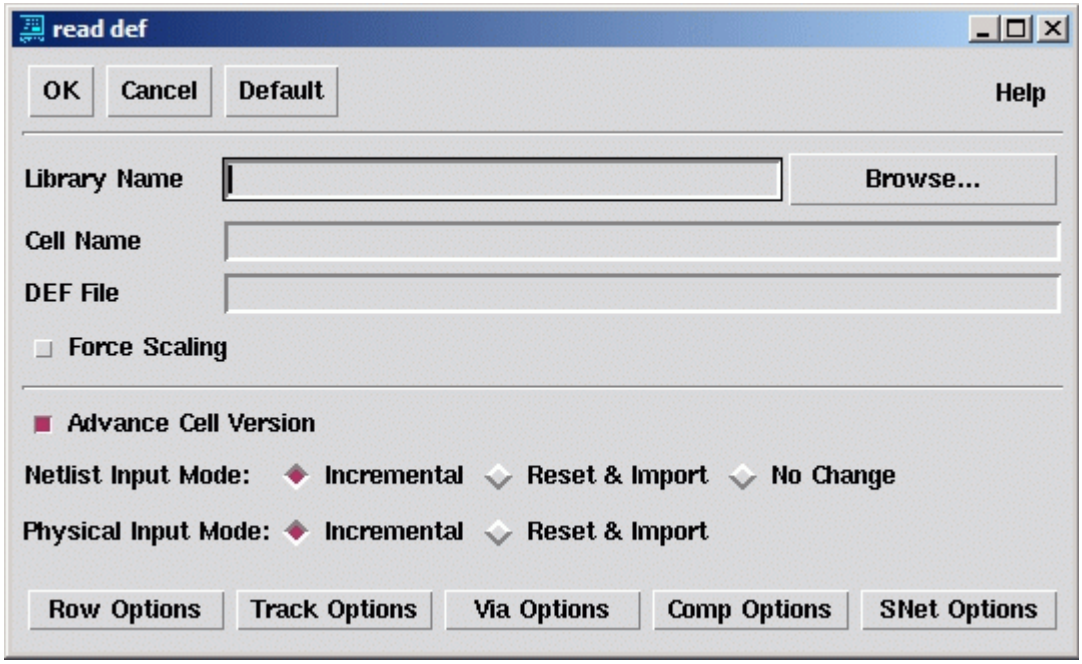
The `read_def` command performs semantic checks according to the LEF/DEF language documentation. As the DEF file is processed, syntax and semantic issues are reported with information messages, warnings, and error messages, depending on the severity of the DEF statement violation. The DEF reader then processes the DEF file until it either finishes processing the input file or cannot proceed further due to a lack of required information.

To open the Read def dialog box, enter the following Scheme command:

```
read_def
```

The Read def dialog box opens, as shown in Figure 4-17.

Figure 4-17 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 between multiple file names. 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.

Note:

The `read_def` command supports the gzip format (*.gz). There is no explicit option in the Read def dialog box for gzip support. However, the command recognizes the *.gz file name extension, such as my_gizpd_input.def.gz, as indicating that the input DEF file is gzipped.

Force Scaling (Optional)

If the Milkyway database precision is not a whole multiple of the input DEF unit, and if this option is selected, `read_def` performs scaling of the data to match the Milkyway database precision. Otherwise, `read_def` quits with an error message about the precision mismatch.

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 the Verilog-plus-DEF input flow as well as the 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 an information message 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 issues an 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. It does not issue any warnings or information messages.

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

Mode 4: `-netl_phys nochange_incr`

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

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

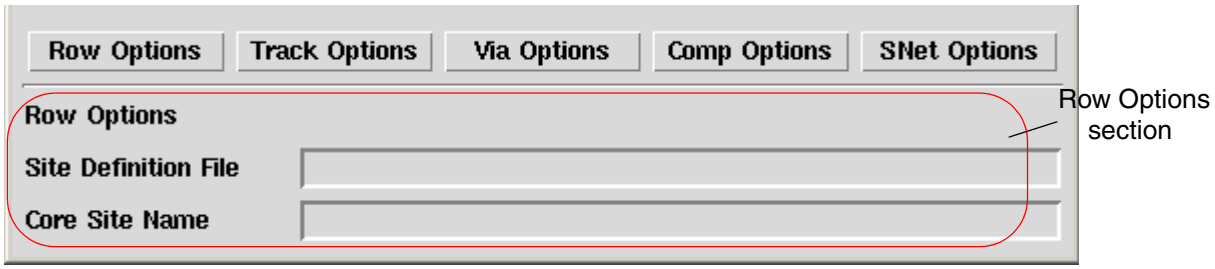
Mode 5: `-netl_phys nochange_rimport`

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

Row Options

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

Figure 4-18 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.

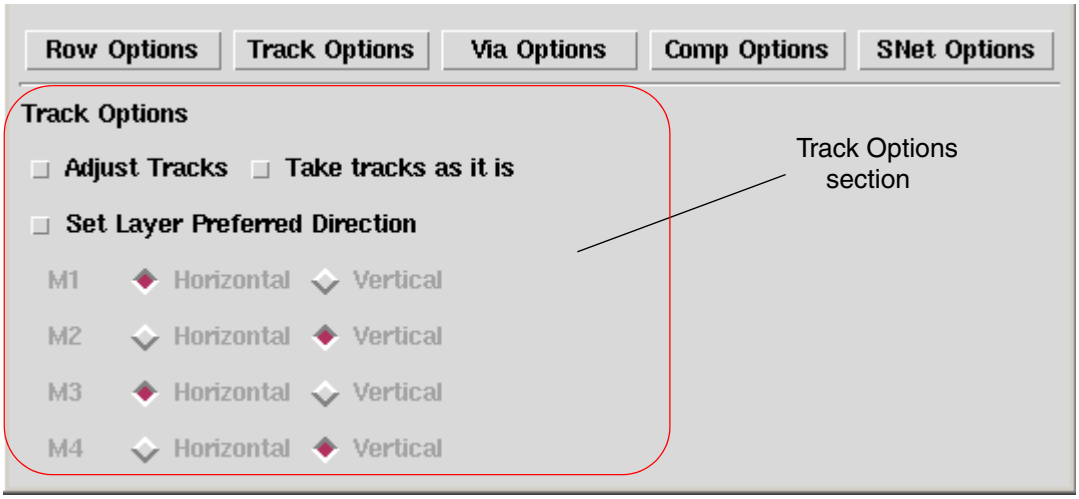
If the Site Definition File box is empty and the name in the Core Site Name box is “core” (which implies that the design library or the reference library has a site named “core”), Milkyway creates rows according to these specified options.

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 does not create rows.

Track Options

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

Figure 4-19 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

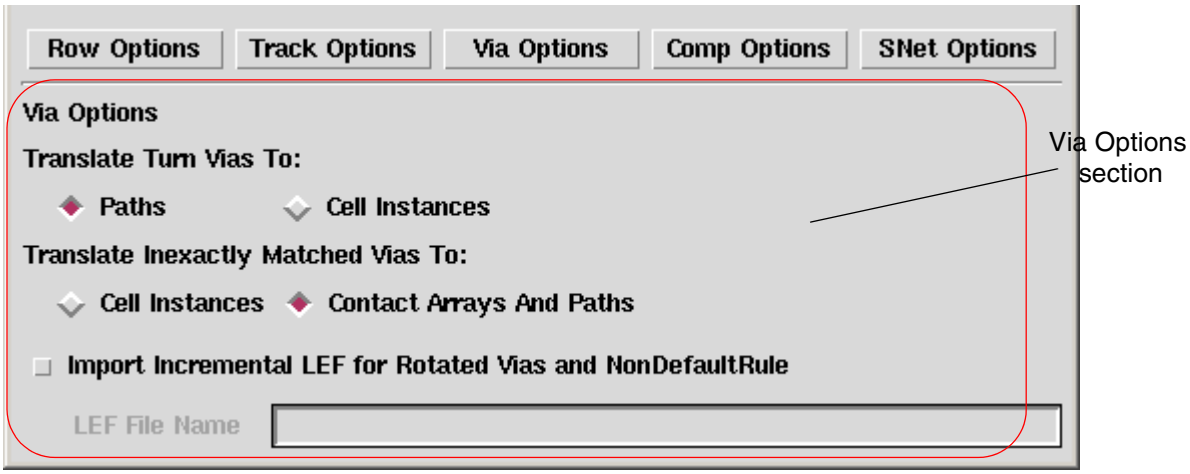
If this option is selected, the wire track direction is set for metal layers M1, M2, M3, and M4.

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 4-20.

Figure 4-20 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

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.

- Cell Instances

  Turn vias are created as via cell instances. The vias remain marked as turn vias.

Translate Inexactly Matched Vias To

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.

- 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.

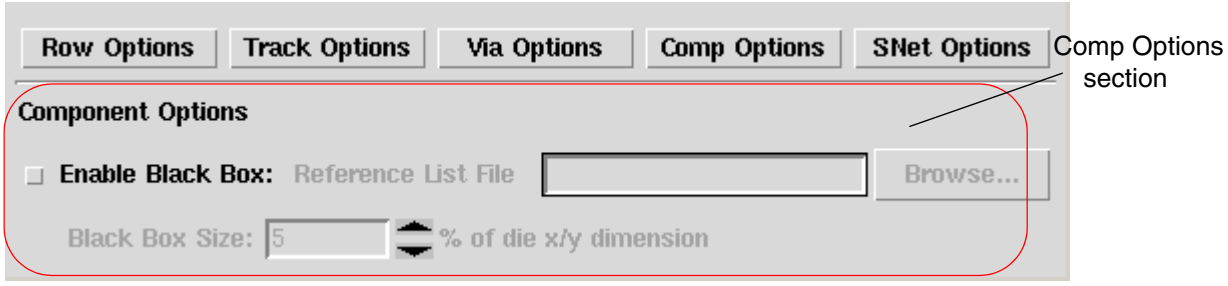
Import Incremental LEF for Rotated Vias and NonDefaultRule

If this option is selected, enter the LEF file name containing the rotated vias and design-specific nondefault rule vias.

Component Options

Clicking the Comp Options button opens the Component Options section, shown in Figure 4-21.

Figure 4-21 read_def Component Options



The Component Options section contains the following options:

Enable Black Box

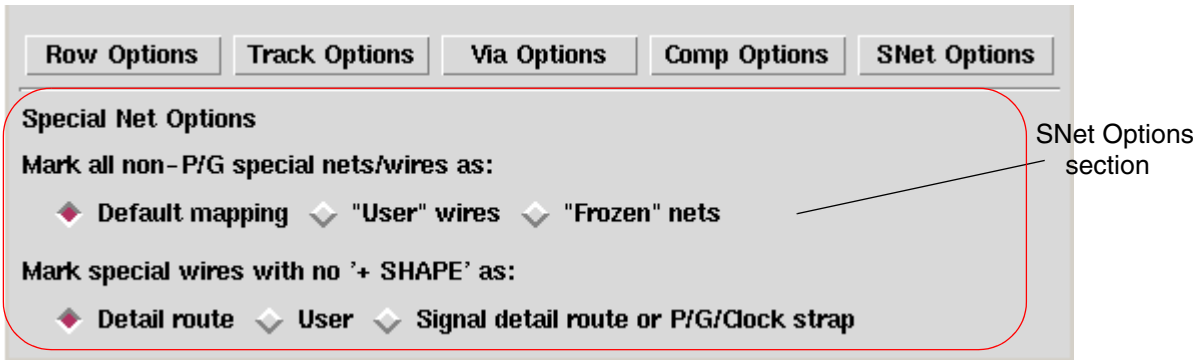
If you select the Enable Black Box option, you must enter the name of the reference list file (which contains the cell instance master names) in the Reference List File text box. Alternatively, you can browse to the reference list file by using the Browse button.

You can also specify the percentage of die x/y dimension in the Black Box Size text box. The black box size is applied to each instance master. The default is 5. For example, if the die size is 100 by 100, the default black box size it 5 by 5.

SNet Options

Clicking the SNet Options button opens the Special Net Options section, shown in Figure 4-22.

Figure 4-22 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 setting has priority.

Exporting DEF From Milkyway Using write_def

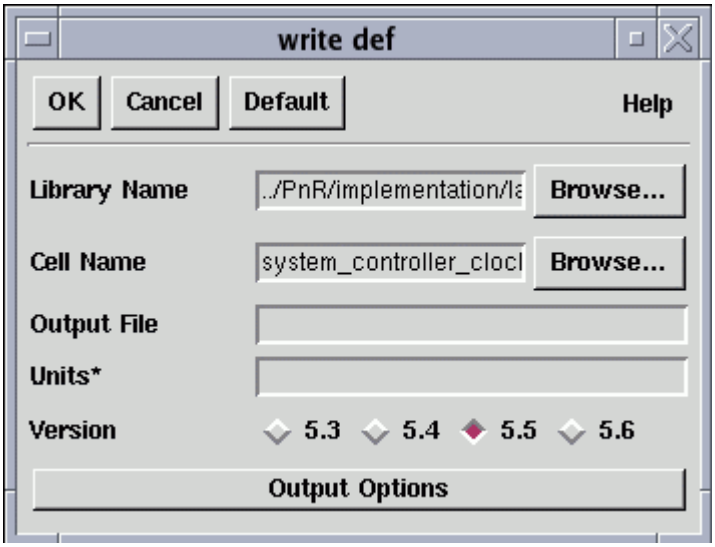
This section discusses the `write_def` dialog box and options used to export DEF from the Milkyway environment. (For information about the `read_def` dialog box and options used to import DEF into the Milkyway environment, see “Importing DEF Data Into Milkyway Using read_def” on page 4-24.)

To open the write_def dialog box, enter the following Scheme command:

```
Milkyway > write_def
```

The Write def dialog box opens, as shown in Figure 4-23.

Figure 4-23 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. 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. To write DEF 5.7, use the `write_def` Tcl command in the Milkyway Environment or the `write_def` command in IC Compiler.

Version Limitations

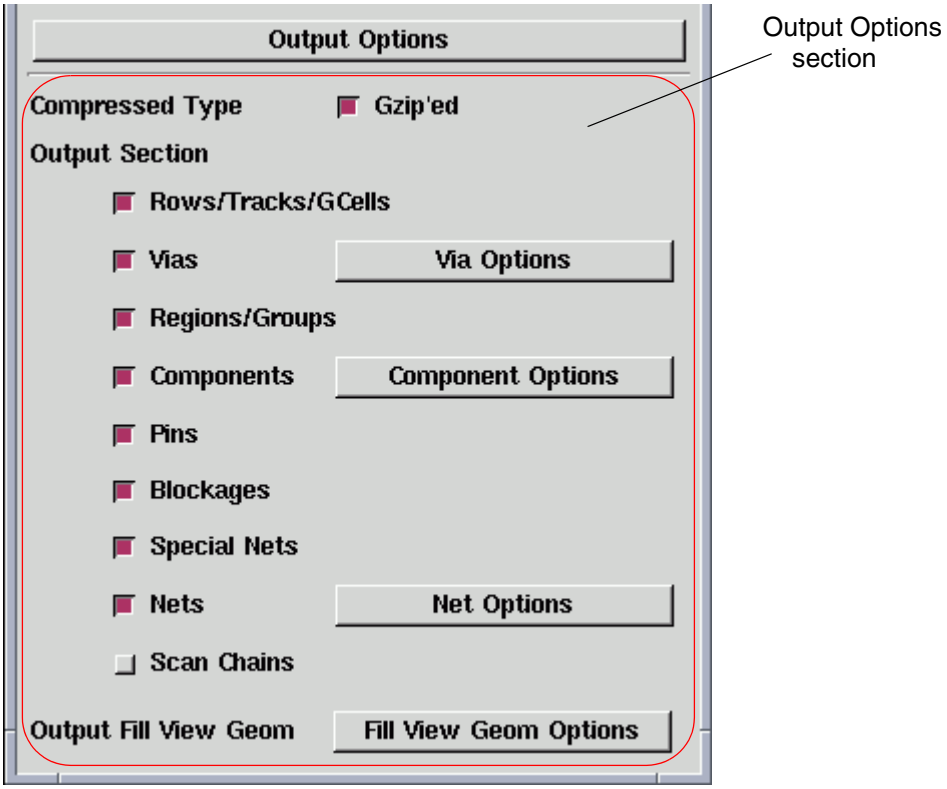
If you read a DEF file version 5.7 into your design and write out a DEF file of a previous version (5.3, 5.4, 5.5, or 5.6),

- The output file might contain incomplete design data due to version incompatibilities.
- The generated vias change to fixed vias. When this happens, the `read_def` command issues a warning message.
- Since the router does not support fill wire OPC, `UnsupportedRouteType` is used for all shapes that the router cannot recognize.
- When used with any DEF version other than 5.7, the `write_def` command will display:
  - The DDEFW-021 message indicating that wiring will be ignored because the `FILLWIREOPC` route type is supported only in DEF 5.7.
  - The DDEFW-022 message indicating that the partial placement blockage will be ignored because partial placement blockage is supported only in DEF 5.7.
- The `read_def` command does not support DEF pin vias and will display the MWDEFR-154 message, indicating that the via will be ignored if the VIA statement is specified in the DEF PINS section.
- The `read_def` command annotates the port statement only if one port statement defines one layer. If one port statement defines multiple layers, the `read_def` command will display the MWDEF-155 error message and exit.

Output Options

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

Figure 4-24 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.

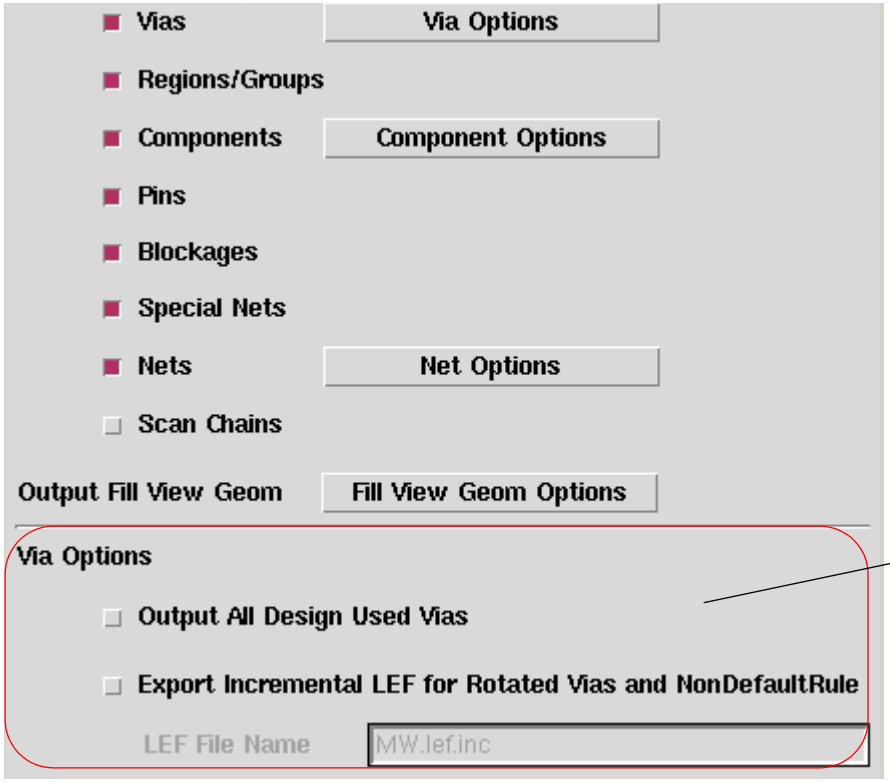
Note:

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 writes out 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 4-25.

Figure 4-25 write_def Via Options



The Via Options section has the following options:

Output All Design Used Vias

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 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.

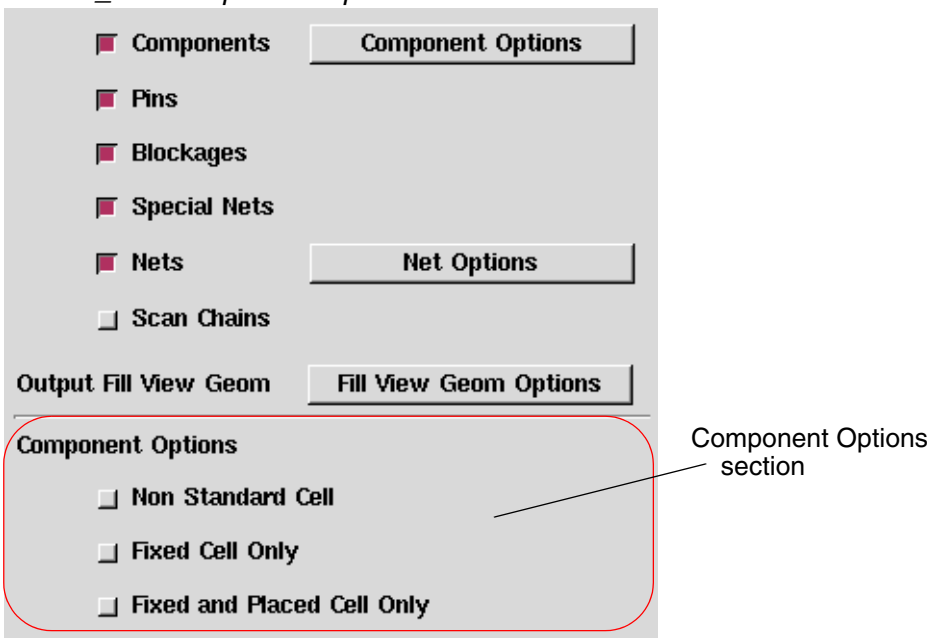
Export Incremental LEF for Rotated Vias and NonDefaultRule

If this option is selected, the command writes 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 4-26.

Figure 4-26 write_def Component Options



The Component Options section contains the following options:

Non Standard Cell

If this option is selected, nonstandard cell information is written.

Fixed Cell Only

If this option is selected, only fixed cell information is written.

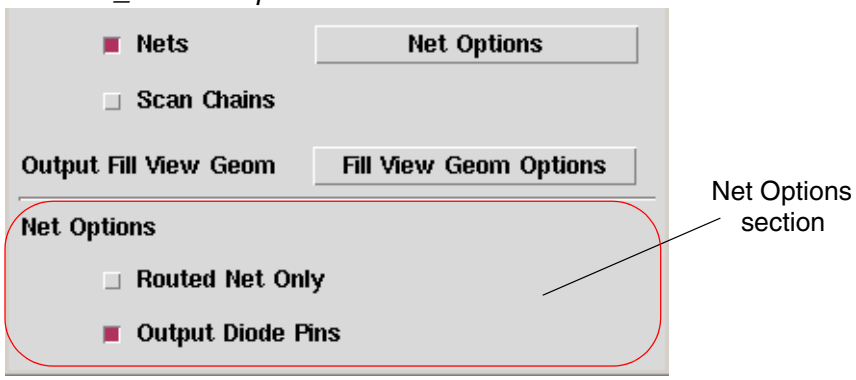
Fixed and Placed Cell Only

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 4-27.

Figure 4-27 write_def Net Options



The Net Options section contains the following options:

Routed Net Only

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.

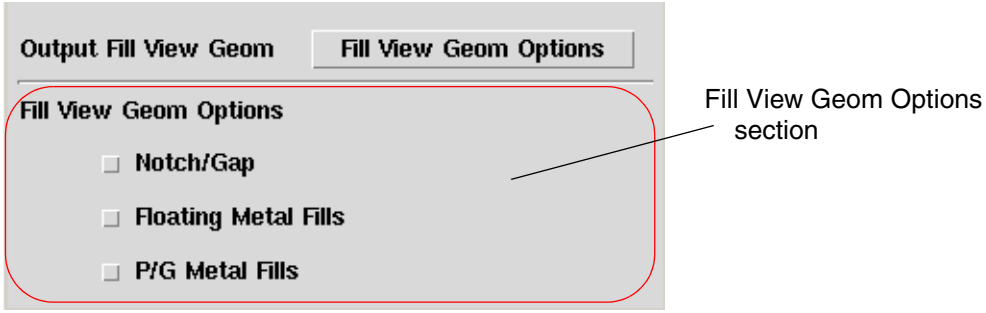
Output Diode Pins

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 4-28.

Figure 4-28 write_def Fill View Geometry Options



The Fill View Geom Options section contains the following options:

Notch/Gap

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

If this option is selected, floating metal fills in Fill View are written to the Fills section.

P/G Metal Fills

If this option is selected, power and ground metal fills in Fill View are written to the special net section with shape `FILLWIRE`.

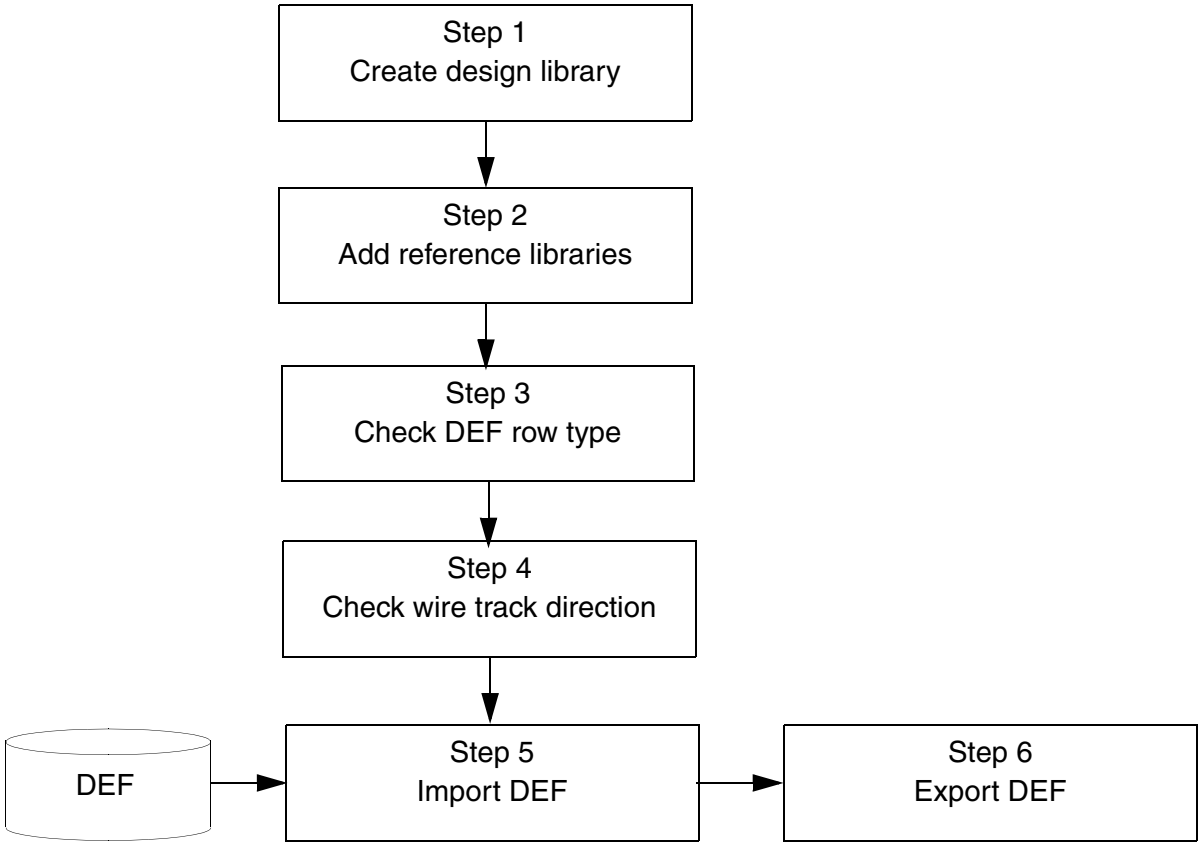
Recommended DEF Flows

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 4-29 is an overview of the input and output flow.

Figure 4-29 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.

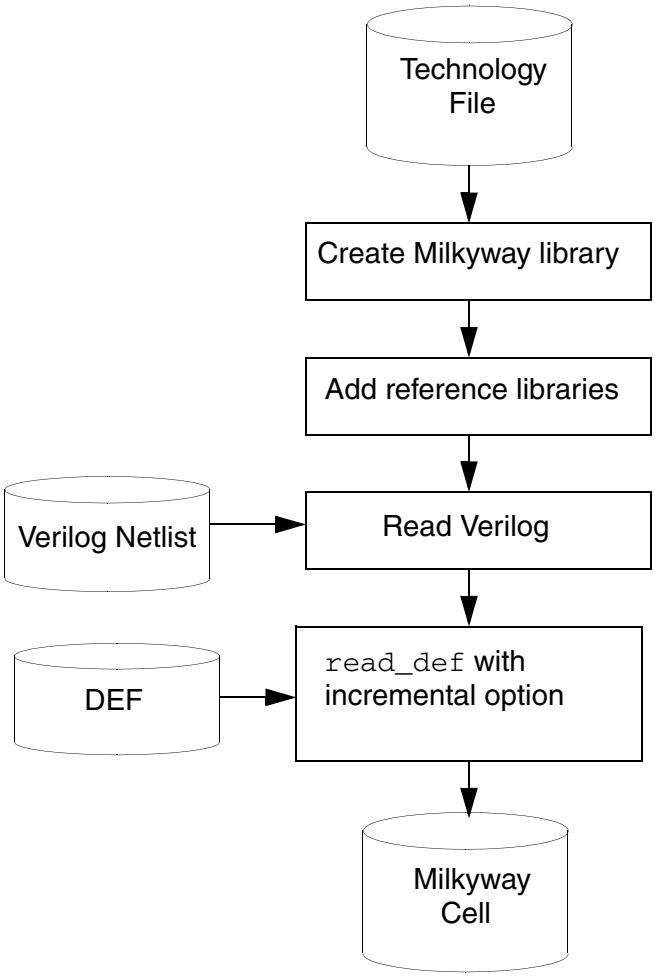
For each via referenced in the net section wiring, if a match to any of the Milkyway Technology contact codes is not found, the via is translated as a Milkyway via cell instance. The vias in the DEF vias section define all generated vias in the design. Therefore, a via cell is created to store all physical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

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 Incremental option in the `read_def` dialog box.

Figure 4-30 shows the incremental DEF flow of reading Verilog followed by DEF.

Figure 4-30 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 `read_verilog` 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 the Incremental option in the “read def” dialog box.

Supported DEF Versions 5.6 and 5.7 Syntax

Table 4-1 contains the DEF syntax, versions 5.6 and 5.7, that the `write_def` command 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 because the older version might be missing constructs that are supported in the newer version.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
1) VERSION `VERSION versionNumber ;`	Yes	No	Yes	The parser is a super set 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 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	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	This section is used to list all properties used in the design. You must define properties in the *PROPERTYDEFINITIONS* statement before you can refer to them in other sections of the DEF file.
`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 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	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 subblocks. 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 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	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, the `write_def` command writes it out from the input DEF file. If the information is not included in the input DEF file, the `write_def` command 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	The `ViaNameMapFile` command 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 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	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. 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. If one of the following conditions occur, the `read_def` command exits: • T • he `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 the enclosure value is smaller than the definition in contact code. If this occurs, no via instance object will be created.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`+ CUTSIZE xSize ySize`	Yes	Yes	Yes	If the cut size is not equal to the definition in the contact code, the `read_def` command quits with an error.
`+ LAYERS botmetalLayer cutLayer topMetalLayer`	Yes	Yes	Yes	If the layers are not equal to the definition in the contact code, the `read_def` command quits with an error.
`+ CUTSPACING xCutSpacing yCutSpacing`	Yes	Yes	Yes	If the cut spacing is smaller than the definition in the contact code, the `read_def` command 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, the `read_def` command 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 `[Styles`	Yes	No	No	This section is currently not supported.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`numStyles ;`	Yes	No	No	
`{- STYLE styleNum pt pt pt ... ;} ...`	Yes	No	No	
`END STYLES]`	Yes	No	No	
15) NONDEFAULTRULES				
`NONDEFAULTRULES numRules ;`	Yes	No	Yes	
`{- ruleName`	Yes	Yes	Yes	
`[+ HARDSPACING]`	Yes	Yes	No	DEF 5.6 syntax. Specifies that the routing spacing requires hard rules, which implies that routers treat extra spacing requirements as violations.
`{+ LAYER layerName`	Yes	Yes	Yes	
`WIDTH minWidth`	Yes	Yes	Yes	
`[DIAGWIDTH diagWidth]`	Yes	No	No	If used in DEF nondefault rule definitions, the `read_def` command produces a warning message and the `write_def` command cannot write it out.
`[SPACING minSpacing]`	Yes	Yes	Yes	
`[WIREEXT wireExt]} ...`	Yes	Yes	Yes	
`[+ VIA viaName] ...`	Yes	Yes	Yes	
`[+ VIARULE viaRuleName] ...`	Yes	Yes	Yes	
`[+ MINCUTS cutLayerName numCuts] ...`	Yes	No	No	If used in DEF nondefault rule definitions, the `read_def` command produces a warning message and the `write_def` command cannot write it out.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`[+ PROPERTY {propName propVal} ...] ...`	Yes	No	No	
`END NONDEFAULTRULES`	Yes	No	Yes	
16) REGIONS `REGIONS numRegions ;`	Yes	No	Yes	A DEF region is a physical area to which you can assign a component or group. A DEF group contains a set of components, which can be either leaf cells or hierarchical cells. The `read_def` command adds the leaf cells into the move bound, but not the hierarchical cells. The `write_def` command uses the move bound type to determine the appropriate DEF region type to write into the REGION section of the DEF file. The `create_bounds` command is used to create the move bounds without a DEF file, and the `report_bounds -all` command is used to report all move bounds.
`[ - regionName pt pt`	Yes	Yes	Yes	
`[ pt pt ]?`	Yes	Yes	Yes	The `read_def` and `write_def` commands support multiple rectangles.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`+ TYPE { FENCE | GUIDE }`	Yes	Yes	Yes	Default - All instances assigned to the region are placed inside the region boundaries, and other cells are also allowed inside the region. It's a hard move bound. FENCE - All instances assigned to this type of region must be exclusively placed inside the region boundaries. No other instances are allowed inside this region. It's an exclusive move bound. GUIDE - All instances assigned to this type of region should be placed inside this region. It's a soft move bound.
`[ + 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	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`[+ HALO [SOFT] left bottom right top]`	Yes	Yes	No	DEF 5.6 syntax. Specifies a keepout margin around the component. SOFT, DEF 5.7 syntax specifies the type of Halo. The default is HARD.
`[ + 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	The PINS section supports multi-layer pins with vias for external DEF pins. The PORT syntax supports multiple ports of a pin, which are not handled as separate PINS statements using a common name.
`[ [ - 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	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`[ + USE [SIGNAL | POWER | GROUND | CLOCK | TIEOFF | ANALOG | SCAN | RESET ] ]`	Yes	Yes	Yes	Milkyway does not support `ANALOG, SCAN, RESET` and regards them as unknown.
`<antenna rules>`	Yes	Yes	Yes	
`[[+PORT]`	Yes	Yes	Yes	
`[ + LAYER layerName pt pt | + POLYGON layerName pt pt pt | + VIA viaName pt ...] ... [ + { FIXED | PLACED | COVER } pt orient ]`	Yes	Yes	Yes	`pinName` is used to set `nameId`. Polygon-shaped pins are supported in DEF 5.6 and 5.7.
`[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	
`[ + PROPERTY { propName propVal }... ]... ; ]…`	Yes	Yes	Yes	
`END PINPROPERTIES`	Yes	No	Yes	
20) BLOCKAGES				
`Blockage numblockages ;`	Yes	No	Yes	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`- LAYER LayerName`	Yes	Yes	Yes	Used for route guide. Due to DEF syntax limitation, DEF files cannot distinguish pre-route route guides from signal route guides. Therefore, the `write_def` command doesn't output pre-route route guides.
`+ COMPONENT CompName | + SLOT | + FILLS`	Yes	No	No	
`+ PUSHDOWN`	Yes	Yes	Yes	In flip-chip design, some top-level objects, such as bump cells, should be considered when the soft macro is being designed, in case the top-level objects overlap with the soft macro. By using PUSHDOWN blockage, overlapping can be avoided before the block is merged with the top level.
`[+ SPACING minSpacing | + DESIGNRULEWIDTH effectiveWidth]`	Yes	No	No	
`{ Rect pt pt`	Yes	Yes	Yes	
`| POLYGON pt pt pt ...} ...`	Yes	No	No	
`- PLACEMENT`	Yes	Yes	Yes	
`[+SOFT]`	Yes	Yes	Yes	DEF 5.7 syntax. Specifies the type of placement blockage. Implies that the initial placement should not use the area, but later timing optimization phases can use the blockage area.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Annotated	Written out	Comments
|+PARTIAL maxDensity	Yes	Yes	Yes	DEF 5.7 syntax. For a partial blockage, the amount of area used to place a cell will be limited to the specified percentage of the blockage area. This reduces the density in a locally congested area and preserves it for timing optimization and buffer insertion.
+ COMPONENT CompName |	Yes	No	No	
+ PUSHDOWN	Yes	Yes	No	Currently, the write_def command 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	
{ 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 layerName [+OPC]	Yes	Yes	Yes	DEF 5.7 syntax. A new route type, UnsupportedRouteType, is defined and used for OPC shapes.
{ Rect pt pt	Yes	Yes	Yes	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`-VIA viaName [+OPC]pt ...]`				DEF 5.7 syntax. Indicates adding via metal fill. Contact or ContactArray without any net connection in fill view cell is used for VIA. If it is an OPC via, the contact or contactArray's route type is `UnsupportedRouteType`. If it is not OPC, the route type is `MiscFillTrackRouteType`.
`| 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	
`+ USE {SIGNAL | POWER | GROUND | CLOCK | TIEOFF | ANALOG | SCAN | RESET}`	Yes	Yes	Yes	`ANALOG`, `SCAN`, and `RESET` are translated into an unknown net type. The `TIEOFF` net must be defined in the netlist so that the DEF reader can read it.

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`+ PATTERN { STEINER | BALANCED | WIREDLOGIC | TRUNK }`	Yes	Yes	Yes	
`+ ESTCAP wireCapacitance`	Yes	Yes	Yes	
`+ WEIGHT weight`	Yes	Yes	Yes	
`+ PROPERTY: { propName propVal }... ]`	Yes	Yes	Yes	
`SPECIALWIRING`				Using the `read_def -preserve_wire_ends` command, all special net power and ground routes are created as paths with square ends; FILLWIRE routes are created as rectangles; all other routes are created as wires with half-width extension ends. You should use this option when the DEF file is from a third party tool after the `read_def` command. Then, the `read_def` command can keep the wiring description exactly as described in the DEF. Also, there will not be any incorrect DRC violations and the flow will not be broken.
`+ POLYGON layerName pt pt pt ...`	Yes	No	No	Milkyway does not support `POLYGON`. In DEF 5.6, 45-degree routing is described using the `routingPoints` option. 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	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
{+ 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.
viaName [DO numX BY numY STEP stepX stepY]	Yes	Yes	Yes	
+ SHAPE { RING | PADRING | BLOCKRING | STRIPE | FOLLOWPIN | IOWIRE | COREWIRE | BLOCKWIRE | BLOCKAGEWIRE | FILLWIRE | FILLWIREOPC | DRCFILL }	Yes	Yes	Yes	routeType is determined by SHAPE and USE. DRCFILL maps to notch/gap in the FILL view. FILLWIREOPC is a DEF 5.7 syntax. OPC indicates that the FILL shapes require OPC correction during mask generation. The router does not support fill wire OPC.
+ 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	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`+ 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	
`+ NONDEFAULTRULE ruleName`	Yes	Yes	Yes	
`+ SOURCE { DIST | NETLIST | TEST | TIMING | USER }`	Yes	No	Yes	
`+ FIXEDBUMP`	Yes	No	Yes	
`+ FREQUENCY frequency`	Yes	No	Yes	
`+ ORIGINAL netName`	Yes	No	Yes	
`+ USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL | TIEOFF }`	Yes	Yes	Yes	`ANALOG`, `SCAN`, and `RESET` are translated into an unknown net type. The `TIEOFF` net must be defined in the netlist so that the DEF reader can read it.
`+ PATTERN {BALANCE | STEINER | TRUNK | WIREDLOGIC }`	Yes	Yes	Yes	
`+ ESTCAP wireCapacitance`	Yes	Yes	Yes	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`+ 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	
`( 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	Yes	The scan in and out port information is written out to the `FLOATING`/`ORDERED` statement.
`[ + START { fixedInConp | PIN } [ outPin ] ]`	Yes	Yes	Yes	

Table 4-1 DEF 5.6 and 5.7 Syntax Supported in Milkyway (Continued)

DEF 5.6 and 5.7 Syntax	Parsed	Anno-tated	Written out	Comments
`[ + 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	
`[- 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

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 Blockage, Pin, and Via Information
- Specifying Port Information for Advanced Operations
- 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. 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. At this point in the design flow, you use the `set_attribute` command to identify power and ground ports.

For example, in Tcl mode,

```
Milkyway> set_attribute [get_ports VDD] port_type "Power"
Information: Setting attribute 'port_type' on port VDD. (MWUI-031)
{"VDD"}
Milkyway> set_attribute [get_ports GND] port_type "Ground"
Information: Setting attribute 'port_type' on port GND. (MWUI-031)
{"GND"}
```

In Scheme mode, you can similarly use the dbSetCellPortTypes command. For example,

```
dbSetCellPortTypes "my_library" "my_cell" '(
        ("VDD" "Inout" "Power")
        ("IOVDD" "Inout" "Power")
        ("VSS" "Inout" "Ground")
        ("IOVSS" "Inout" "Ground")
 ) #f
```

In this example, the scheme command defines the cell ports and their types for VDD, IOVDD, VSS, and IOVSS ports of the cell my_cell in the library named my_library. The `#f` indicates that it will replace any existing port information for the cell.

You can check of the port type information to verify whether this information has been loaded. You can generate the port type information using the following command:

```
dbDumpGPortTable "my_library" "port_info.txt"
```

Before you extract back-bias pins, ensure that the n-well, p-well, deep p-well, and deep n-well mask names are added to the technology file. Also ensure that the bias pins port type is set to BiasPG.

Smashing Hierarchical Cells

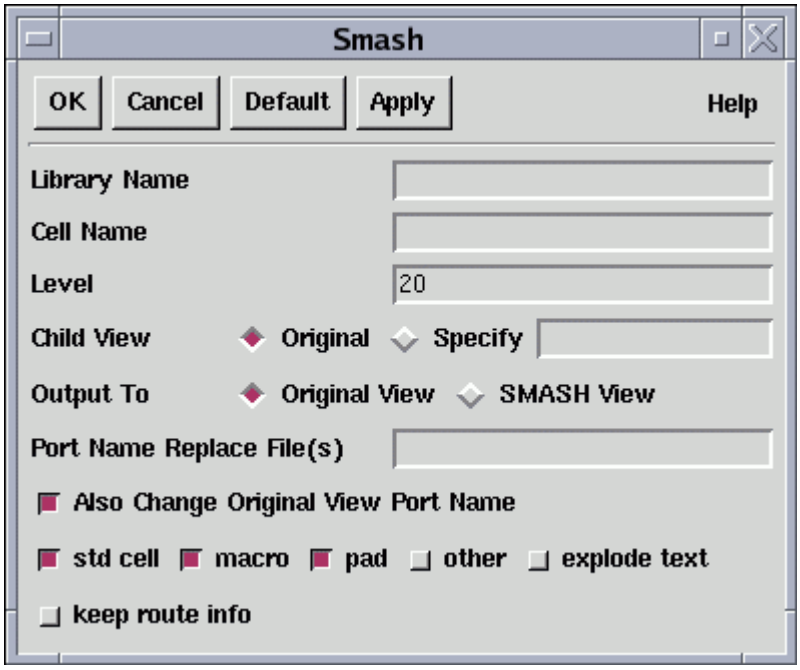
To perform processing that involves geometries inside the hierarchy of a layout cell, you must first smash the layout cell to a level sufficient to provide the access required to perform the processing. For example, to remove unnecessary metal2 from metal1 pins inside a cell instance that is inside a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

During the creation of pin or 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. Choose Cell Library > Smash.

   The Smash dialog box appears.



2. Fill in the Smash dialog box options.
3. Click OK.

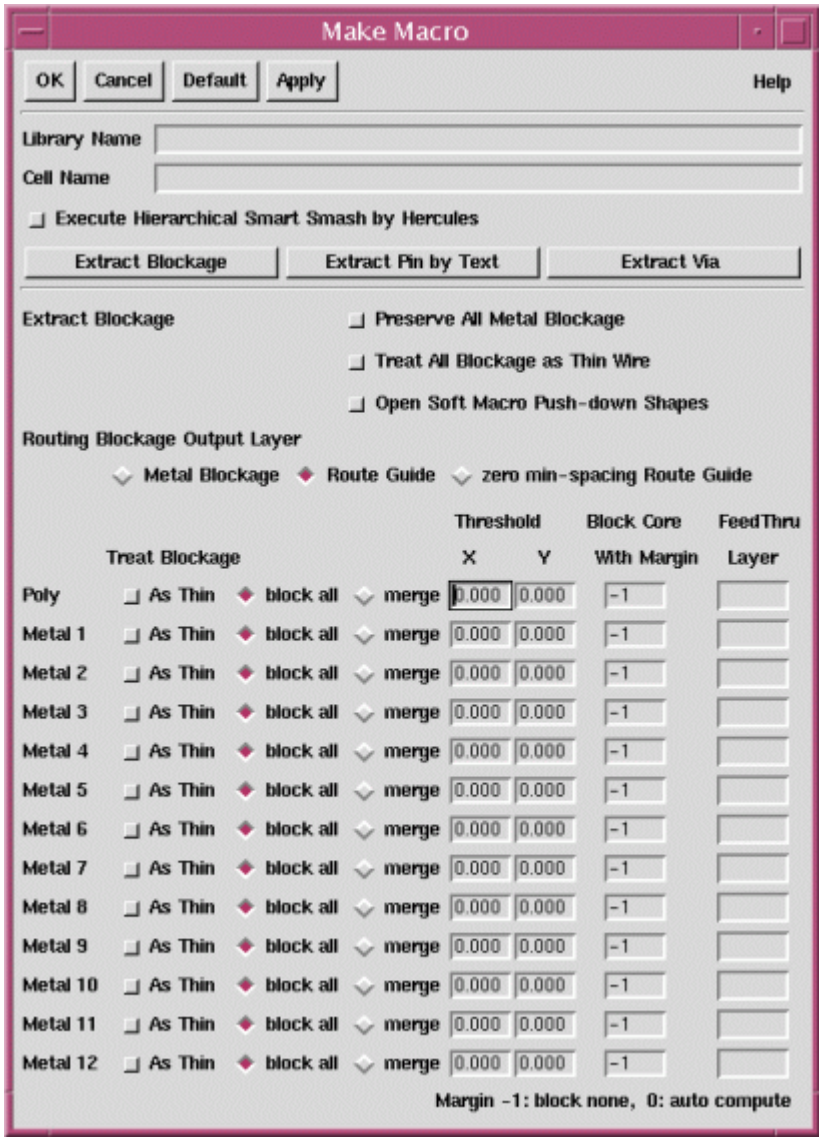
Extracting Blockage, Pin, and Via Information

After you create and process the layout cells, you need to create a FRAM (routing) view for each layout cell in your library. The FRAM view for a cell contains the following:

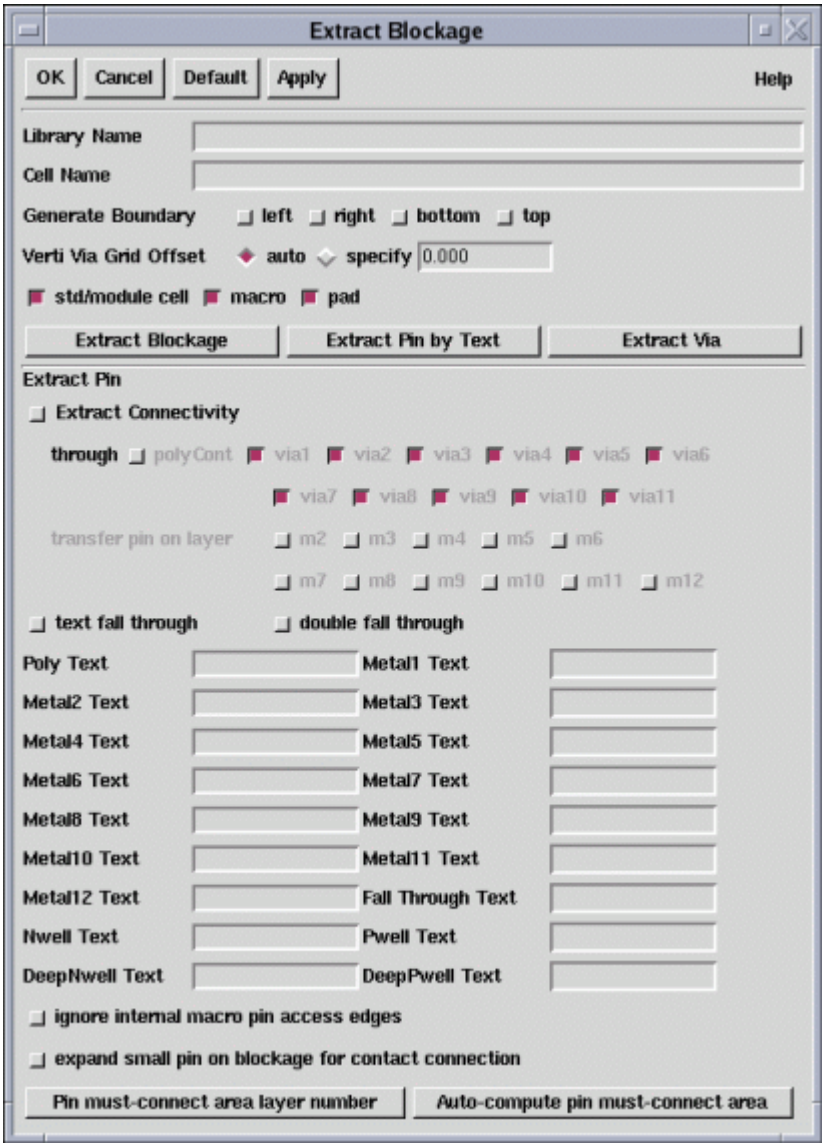
- Blockage areas (areas where no routing on a particular layer can occur)

- Pins (cell connection points during routing)
- Via regions (areas where vias can be placed during routing)
- Pin solutions (solutions for routing to pins)

For cells created in the Milkyway database, you can perform pin and blockage extraction by choosing Cell > Make Macro Abstract.



You create pin or blockage cells by choosing from the menu Cell Library > Blockage, Pin & Via, which displays the Extract Blockage dialog box.



After entering values and selections for the options in this dialog box, click OK to perform the extraction, which creates the FRAM views of the cells.

You can run this process with different options for standard cells, module cells, macro cells, and pads. To do so, select the types of cells you want to process with the standard, module, macro, and pad options. After you perform extraction, you choose Library > Check to check your library.

Creating Via Regions and Pin Solutions

During blockage, pin, and via 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 from 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 From 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 list of layers used for the text identifying poly geometries.
- Metal# Text is the list of 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

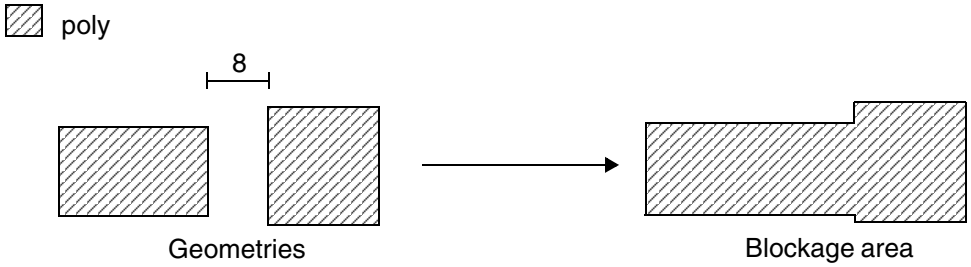
Milkyway creates blockage areas by merging any geometries on the same layer that are within merging distance of each other except for pins. By default, the merging distance equals

$$(2 \times \text{minSpacing}) + \text{minWidth}$$

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.) Therefore, Milkyway merges the geometries on the left to create the blockage area on the right, as shown in Figure 5-1.

Figure 5-1 Merging Same-Layer Geometries



In the Extract Blockage dialog box (Cell Library > Blockage, Pin & Via), when you click the Extract Blockage button and enable Merge Blockage, you can set the merging thresholds for Poly, Metal1 through Metal12, PolyCont, and Via1.

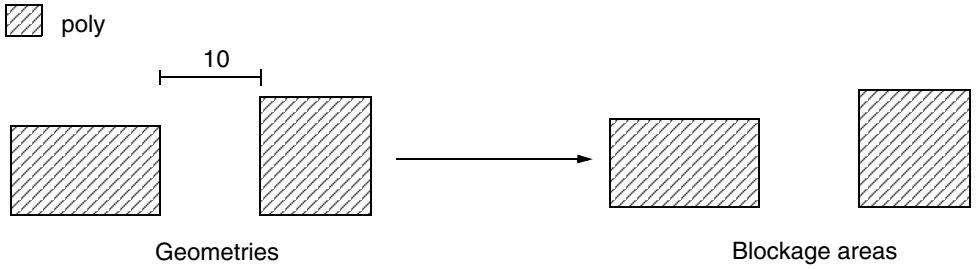
Typing a threshold value for one of these layers changes the merging distance to the following:

$$(2 \times \text{minSpacing}) + \text{threshold}$$

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.

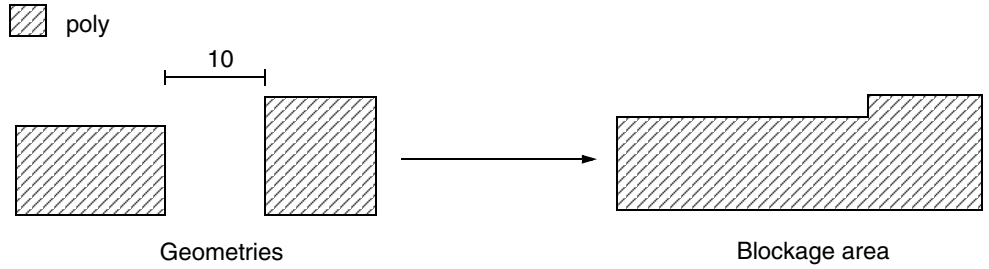
For example, if the technology file specifies a poly *minSpacing* of 2 and a poly *minWidth* of 4 and the poly threshold is 0, the merging distance equals 4. Therefore, no merging occurs, as shown in Figure 5-2.

Figure 5-2 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 5-3.

Figure 5-3 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 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, make sure that the Generate Boundary options are not selected 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.

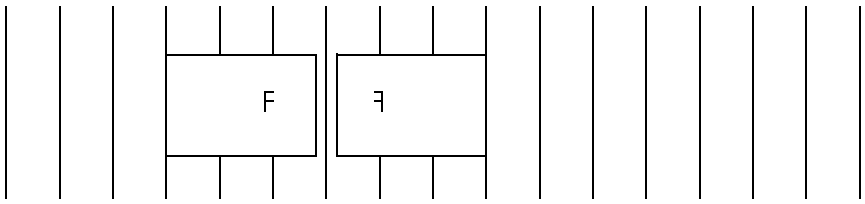
Verti Via Grid Offset

Indicate how you want to place standard cells to achieve optimum horizontal via placement.

Option	Instructs how 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

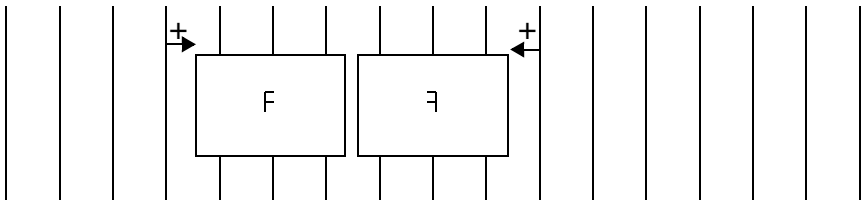
To place an unflipped cell so that its left side is on a vertical wire track (or a flipped cell so that its right side is on a vertical wire track), as shown in Figure 5-4, select “specify” and type a zero (0).

Figure 5-4 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 5-5, select “specify” and type a positive number (less than the pitch of the vertical routing layer) to indicate the distance.

Figure 5-5 Specifying Placement of Cell at Specified Distance From Vertical Wire Track



Specifying Port Information for Advanced Operations

Before you create the FRAM view for a cell, you must identify the power and ground ports by using the `set_attribute` command. If you plan to perform advanced operations, you can specify additional port information after you create the FRAM view.

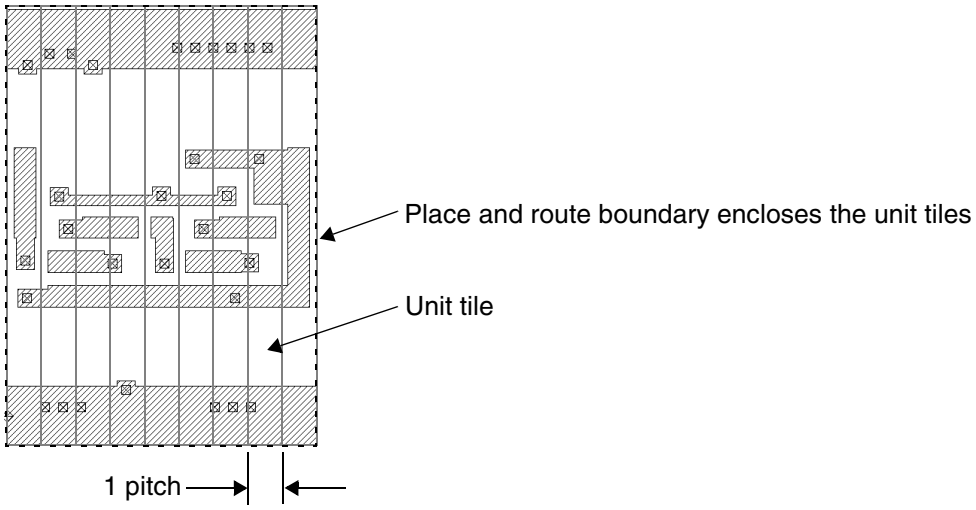
Creating a Port Information File Manually

Create a Tcl script to set the desired attributes. Open the library and cell. Then run the script to set the attributes.

Setting the Place and Route Boundary

Synopsys tools can use 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 5-6.

Figure 5-6 Place and Route Boundary Example

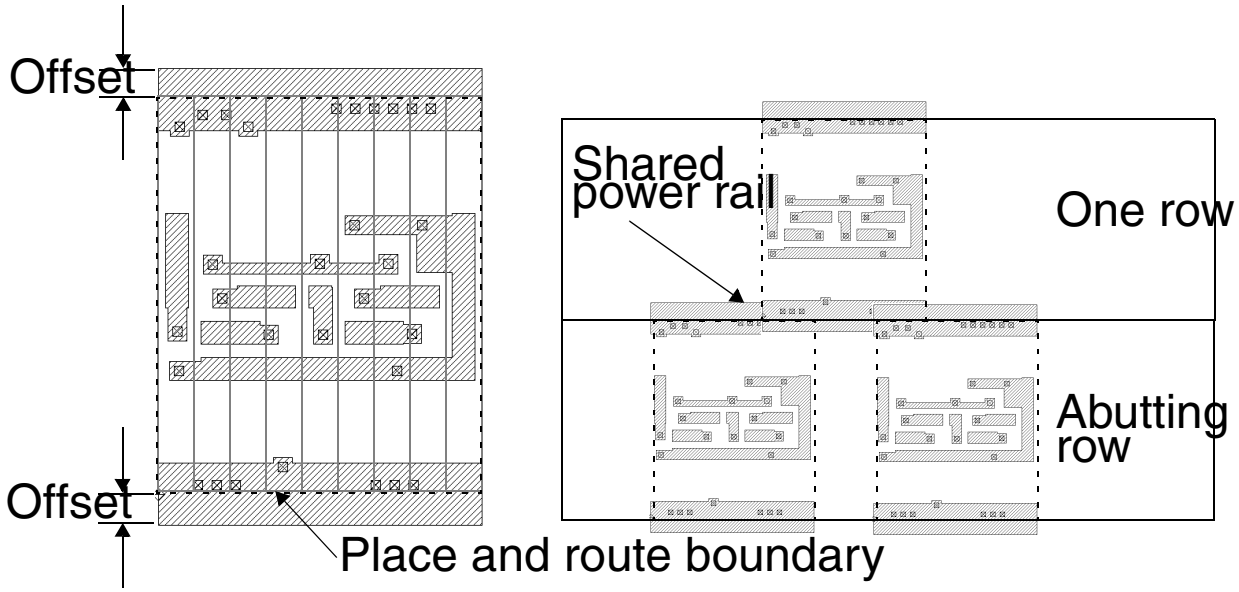


You generate place and route boundaries for cells and create the unit tile in your library by selecting Cell Library > Set PR Boundary from the menu. Synopsys tools can observe the following specifications when creating the place and route boundaries and unit tile for each library:

- 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 5-7.

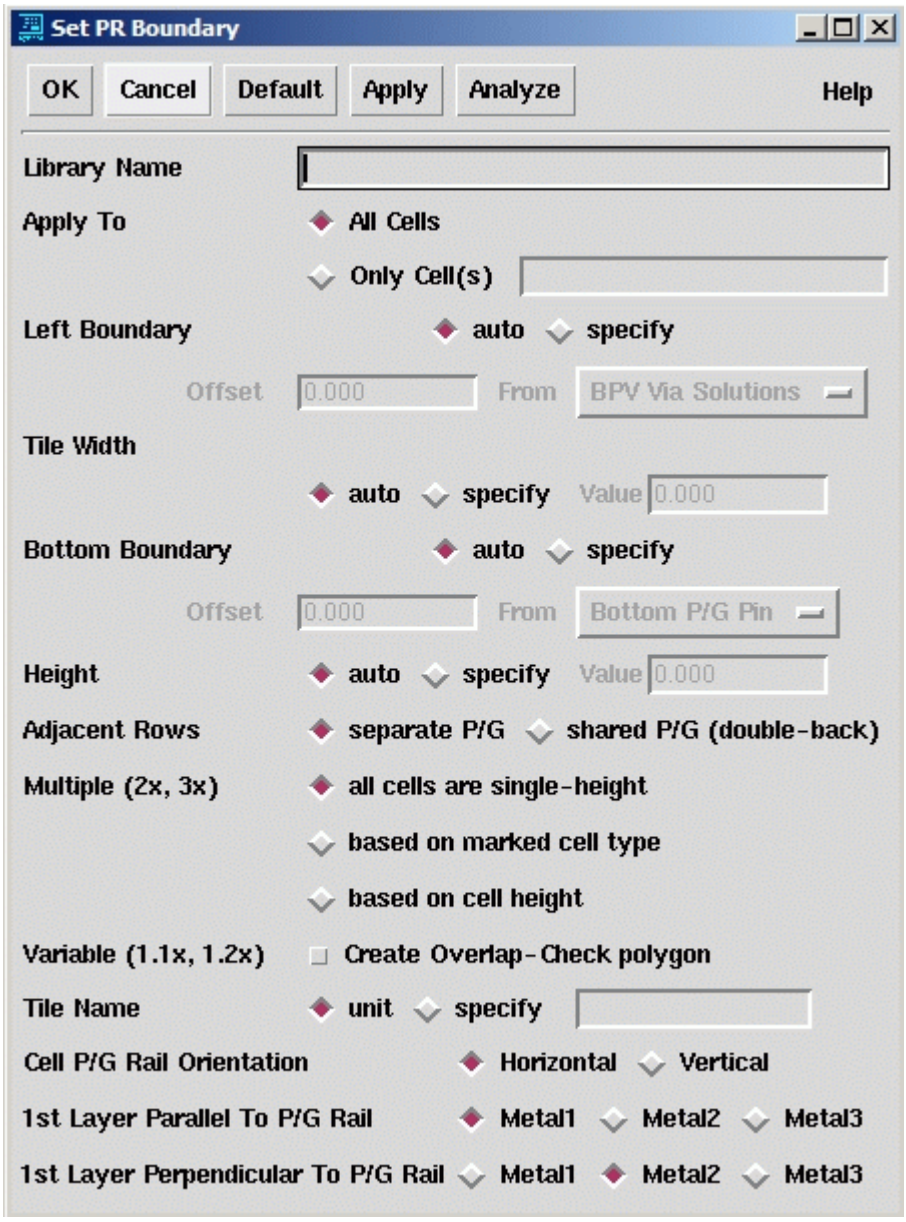
Figure 5-7 Overlapping Abutting Rows Example



To set the place and route boundary,

1. Enter Cell Library > Set PR Boundary.

   The Set PR Boundary dialog box appears.



2. Enter the required information in the Set PR Boundary dialog box.

   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.

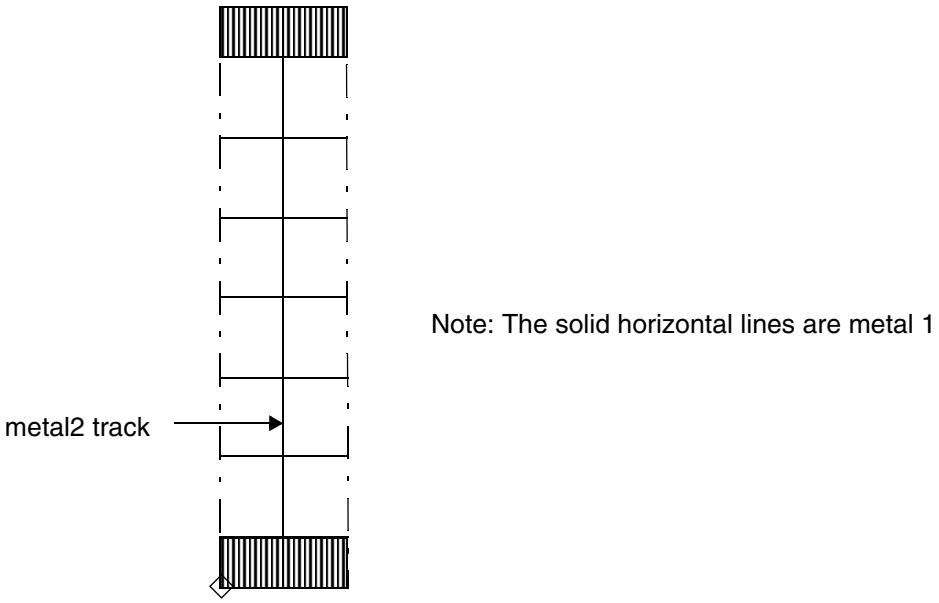
The Cell Library> Set PR Boundary command creates the place and route boundaries for the specified cells and creates a unit tile.

Defining Wire Tracks

After you adjust the cell boundary and create the unit tiles using Cell Library> Set PR Boundary, 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 5-8 shows an example of a unit tile with the wire tracks defined.

Figure 5-8 Unit Tile With Wire Tracks Defined

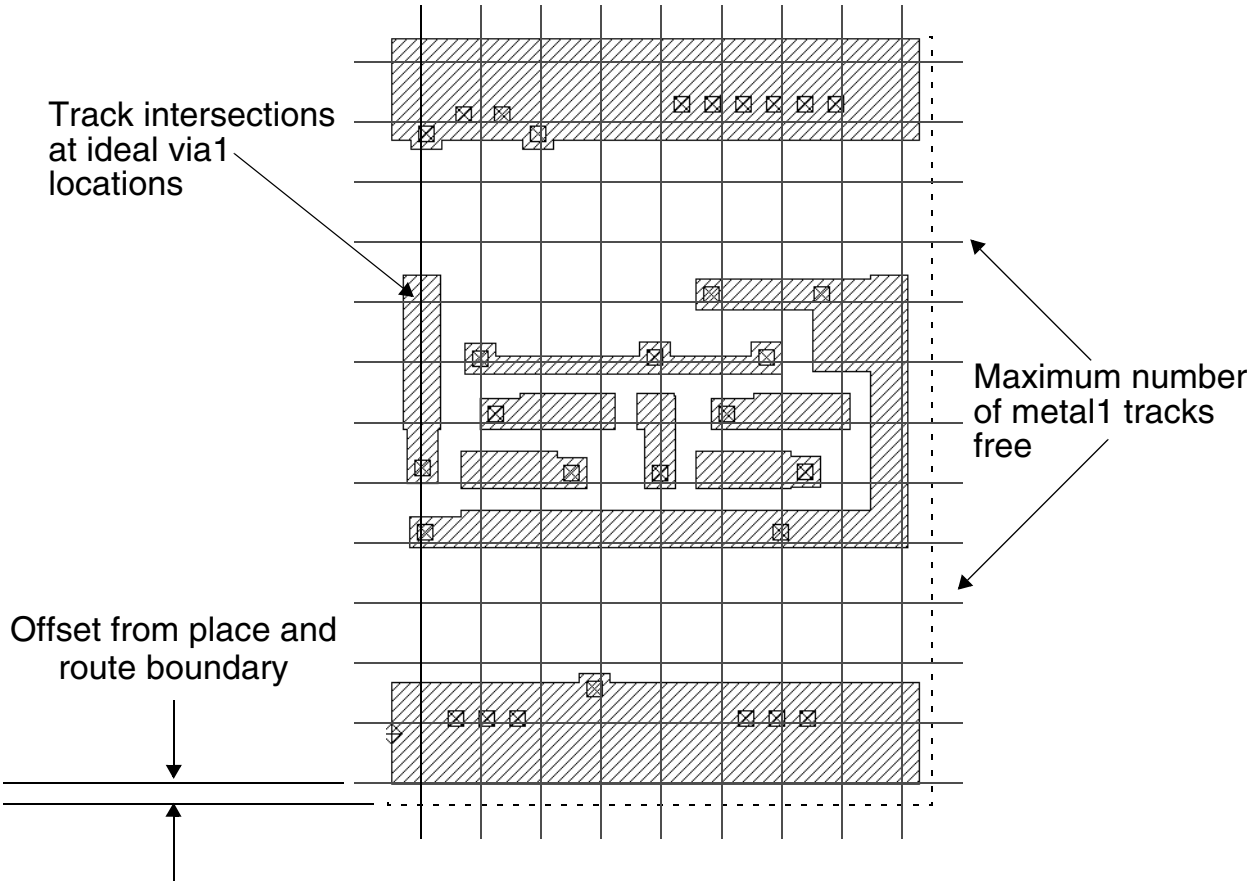


Use the following guidelines to ensure that the wire tracks are created in a way that optimizes routing capability:

- 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. 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 5-9).

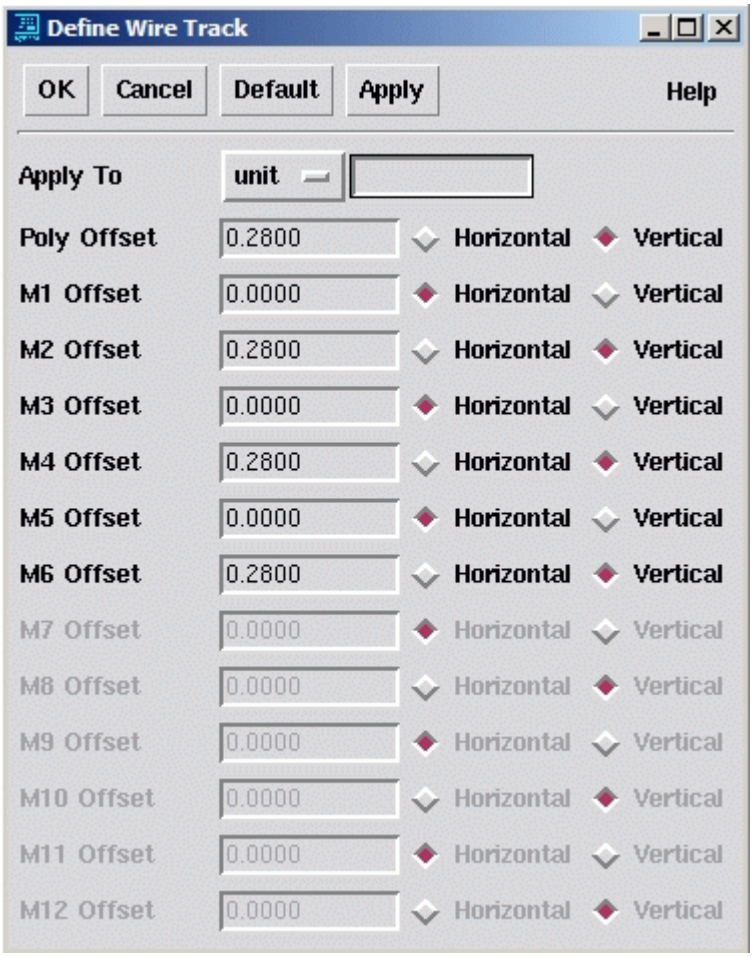
Figure 5-9 Standard Cell Placed on Array of Unit Tiles



To define wire tracks,

1. Enter 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.
3. Click OK.

6

Library Checking

Advanced IC design relies on high-quality libraries. Ensuring that library data is correct and consistent is an essential step before design creation. Milkyway provides a checking capability for logical and physical libraries to facilitate successful design and verification. Although Milkyway provides consistency checks, you need to ensure consistency between your physical and timing libraries.

This chapter includes the following sections:

Library Checking Overview

Consistency between logic libraries and physical libraries is critical to achieving good results. You can perform library checking for logical and physical libraries with the `check_library` command in IC Compiler.

Note:

The Milkyway Environment also has a `check_library` command that can perform some of the same types of library checking as the `check_library` command in IC Compiler. However, the command options, command option syntax, default behavior, and setup options are different between the two tools. This chapter describes the `check_library` command usage in IC Compiler.

To make sure that your libraries are consistent, run the `check_library` command:

```
icc_shell> check_library
```

By default, the `check_library` command performs consistency checks between the logic libraries specified in the `link_library` variable and the physical libraries that are referenced in the current Milkyway design library. You can also physical libraries for self-consistency or consistency between logic libraries and other logic libraries. This is the full command syntax:

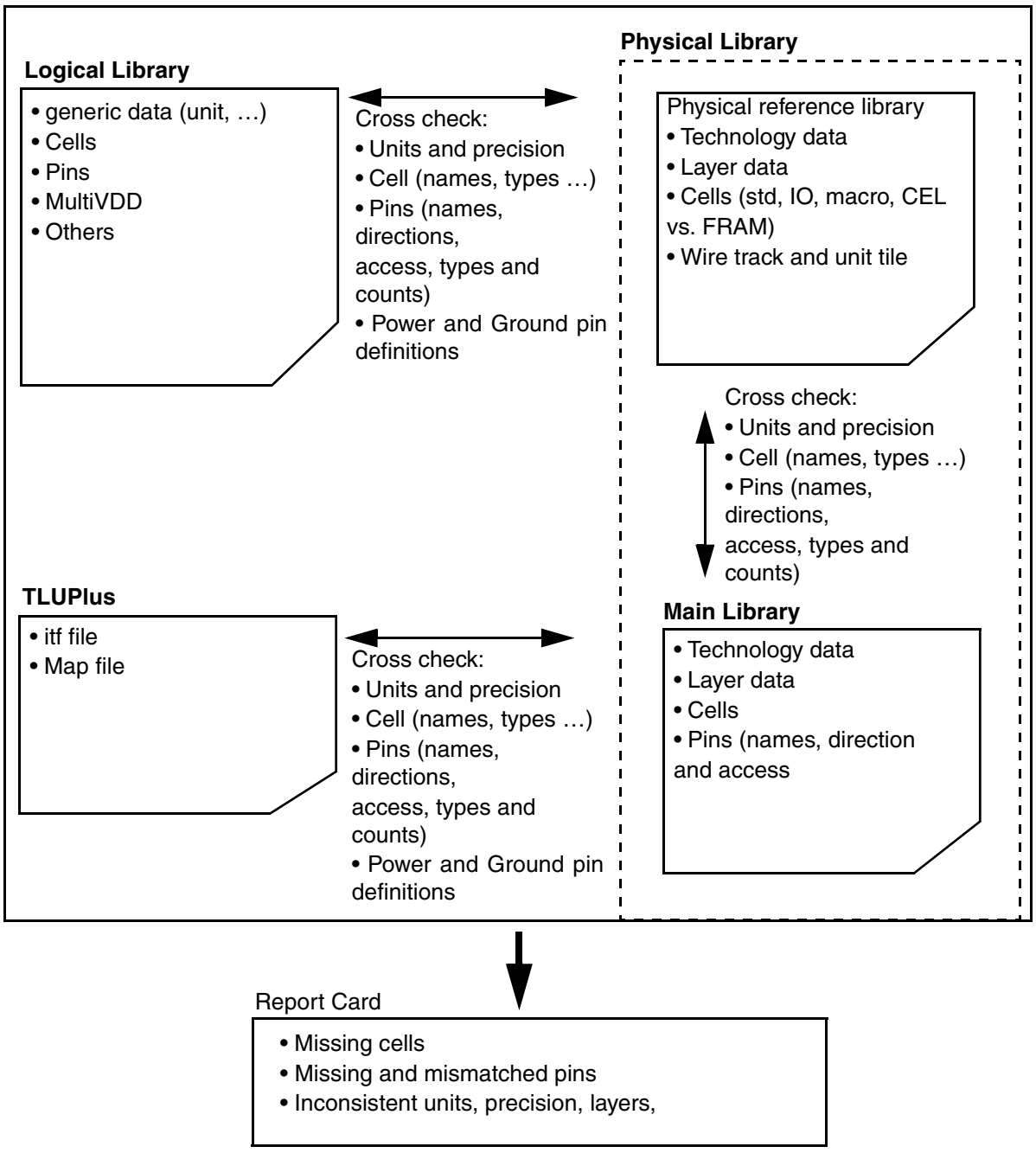
```
check_library
  [-logic_library_name logic_library_names]
  [-mw_library_name physical_library_names]
  [-cells cell_list]
```

The libraries that you explicitly specify in the command override the default libraries. The `-logic_library_name` option specifies a list of logic libraries to be checked for consistency. If the `-logic_library_name` option is not explicitly specified but the `set_min_library max_library` command option is specified, the command checks the minimum and maximum libraries as a pair. If neither the `-logic_library_name` option nor the minimum and maximum libraries are specified but the `link_library` and `search_path` variables are set, the command groups the libraries in the list by cell set, and within each group, it checks the consistencies across all the libraries. Libraries that cannot be paired are not checked.

Use the `-cells` option to specify a list of cell names to check. If the cell names are not specified, all the cells in the library are checked. The `check_library` command returns a status message indicating whether the check was successful.

Figure 6-1 provides an overview of the Milkyway library checks that are performed by `check_library`.

Figure 6-1 Library Checking Overview



By default, the `check_library` command verifies that all cells that exist in the logic library and the physical library also exist in the other library, including any physical-only cells, and that the pins on each cell are consistent between the logic library and the physical library. This validation includes consistency in naming, direction, and type, including power and ground pins.

Note:

If the logic library does not contain `pg_pin` definitions, the command uses the power and ground pins as defined in the physical library.

You can control the types of library checking performed by the `check_library` command and the format of the checking report by using the `set_check_library_options` command. This is the full syntax of the command:

```
set_check_library_options
  [-mcmm]
  [-upf]
  [-validate {timing }]
  [-scaling {scaling_types}]
  [-compare {construct | attribute | value}]
  [-tolerance {type relative_tolerance absolute_tolerance}]
  [-group_attribute {group_name/attribute_name}]
  [-analyze {nominal_vs_sigma table_trend}]
  [-criteria
    {std_error=val_err slope=val_sl trend=['/' | '' | '^' | 'V']}]
  [-report_format
    {csv[=csv_dir] [nosplit] sort_by_cell | sort_by_group_type}]
  [-leq]
  [-cell_area]
  [-cell_footprint]
  [-bus_delimiter]
  [-routeability]
  [-view_comparison]
  [-antenna]
  [-signal_em]
  [-same_name_cell]
  [-rectilinear_cell]
  [-physical_only_cell]
  [-tech_consistency]
  [-tech]
  [-drc]
  [-phys_property phys_property_list]
  [-reset]
  [-logic]
  [-physical]
  [-logic_vs_physical]
  [-all]
```

For example, you can explicitly specify the following types of library consistency checking:

- Verify that the area for each cell (except pad cells) is consistent between the logic library and the physical library.

  ```
  icc_shell> set_check_library_options -cell_area
  ```

- Verify that the cell footprints are consistent between the logic library and the physical library.

  ```
  icc_shell> set_check_library_options -cell_footprint
  ```

- Verify that the same bus naming style is used in the logic library and the physical library.

  ```
  icc_shell> set_check_library_options -bus_delimiter
  ```

Specify the option for each check that you would like to enable, or to enable all consistency checks, use the `-logic_vs_physical` option.

To reset the library checks to the default settings (cell name and pin consistency checks), run the `set_check_library_options -reset` command.

To see the enabled library consistency library checks, run the `report_check_library_options -logic_vs_physical` command.

If `check_library` reports any inconsistencies, you must fix these inconsistencies before you process your design.

Validating Logic Libraries

You can use the `check_library` command to check the quality of a logic library. To perform these logic library checks, enable the specific checks you want by running the `set_check_library_options` command. If you do not specify any options for `set_check_library_options`, `check_library` performs default (or general) library checks.

Table 6-1 shows the logic library checks and the `set_check_library_options` command options used to enable them.

Table 6-1 Logic Library Checks

Option	Description
`-compare {construct attribute value}`	General logic library consistency checking.
`-mcmm`	Logic library consistency checking for multicorner-multimode (MCMM).

Table 6-1 Logic Library Checks (Continued)

Option	Description
`-scaling {timing noise power}`	Logic library consistency checking for CCS timing, CCS noise, and CCS or NLDM power scaling.
`-tolerance {`*`type relative_tolerance absolute_tolerance`*`}`	Specifies a relative tolerance and an absolute tolerance for characterization value comparison.
`-upf`	Logic library consistency checking for multivoltage and UPF.
`-validate`	Logic library consistency checking for library characterization.
`-logical`	Specifies to enable all the logic library checks.
`-reset`	Resets the options to the default.
`-all`	Specifies to check all items for the libraries including logic versus physical and physical library checking.

Specify an option for each check that you would like to enable, or if you want to enable all logic library checks, use the `-logical` option. To reset the library checks to the default settings, run the `set_check_library_options -reset` command. To enable all checks for logic and physical libraries, use the `-all` option.

General Logic Checks

The `check_library` command performs the following logic versus logic library general checks. For the libraries to pass a flow check, the general logic checks should not have any problems.

Library Group

For the library group, the `check_library` command reports the library version and the library type, whether the library is power and ground pin based or not. The command also checks if the libraries have the same units and the same slew trip points for CCS timing scaling. The command checks if the libraries have the default operating conditions that are defined. The `check_library` command performs these checks, with the exception of the trip points check, by default. Use the `set_check_library_options -scaling timing` command to perform the trip points check.

Cell Group

The `check_library` command checks for the same number of cells in each library. It also checks the `area`, `function_id`, `dont_use`, and `dont_touch` attributes for cells with the same name for consistency. The `check_library` command performs these checks by default.

Pin and pg_pin Groups in the Cells

The `check_library` command checks if the libraries have the same number of pins, including PG pins. The command also checks if the PG pin names, directions, types, functions, and `voltage_name` in the two libraries are the same. Apart from these checks, the command also checks if the two libraries have the same:

- `input_signal_level` or `output_signal_level` for a rail-based library and the same `related_power_pin` or `related_ground_pin` for pg_pin-based libraries.
- Derived `pin_number` attribute values. If there is a mismatch, the `check_library` command checks the order of pins in each cell group and function Boolean expressions, and others that can also affect the mismatch and are unknown at this point.

The `check_library` command performs these checks by default.

Timing Arcs

The `check_library` command checks for timing arcs with matching related pins, timing type, timing sense, and `when` conditions across the libraries. In any two sets of logic (technology) libraries with the same cell group, if such timing arcs are used in at least one design, they are checked for consistency between these libraries. The `check_library` command performs these checks by default.

Specific Logic Checks

Use the `set_check_library_options -compare construct` command to specify that the `check_library` command compare all groups, subgroups, attributes, and characterized values, such as the timing, power, and current between two libraries.

Use the `set_check_library_options -compare value` command to specify that the `check_library` command compare two libraries for all characterized values.

Special Checks

Some flows and applications require special checking in addition to the general checks. Use the `set_check_library_options -scaling timing` command to specify that the `check_library` command perform the following checks for CCS timing scaling:

- The number of CCS timing driver model load indexes must be the same for each timing arc.
- The output load indexes in each CCS timing driver model timing arc must be the same.
- For Setup Hold Pessimism Reduction, all load and slew indexes should be the same across the libraries.
- The receiver models across all scaling libraries exist and should be the same type.
- The order of the receiver and noise model and the conditional `when` and default pin model are the same across the libraries.
- The base `curve_x` is the same across the libraries at a number of points and for each value for compact CCS.
- All libraries have either the compressed or original format for driver data.
- Characterization waveform type between two libraries should be the same. The command checks for cell or pin level attributes such as `driver_waveform`, `driver_waveform_rise`, `driver_waveform_fall`, and `driver_waveform_name` in the `normalized_driver_waveform`.

Use the `set_check_library_options -scaling noise` command to specify that the `check_library` command perform the following checks for CCS noise scaling.

- The `input_voltage` and `output_voltage` indexes for CCS noise models in the libraries across the scaling groups are the same percentage of VDD.
- The command checks that the conditional `when` pin models and the receiver models are in the same order across the libraries.
- The number of CCS noise models is the same across the libraries for all pins and arcs.

Use the `set_check_library_options -scaling power` command to specify that the `check_library` command perform the following checks for CCS or NLPM power scaling to ensure that the libraries have the same:

- Set of `dynamic_current` and `leakage_current` groups for each cell.
- Set of `intrinsic_parasitic` and `leakage_power` tables for each cell.
- `power_cell_type` attribute for each cell.
- Set of `internal_power` tables for each pin or cell.

For voltage scaling, the `check_library` command checks that the voltages are no more than 20 percent apart for good accuracy. This check is performed for reporting and does not determine the success or failure of the check.

Use the `set_check_library_options -upf` command to specify the following checks for multivoltage or UPF flows:

- Different PVT values for different characterizations.
- NLDM or CCS model under different voltage conditions exists.
- Level shifter, isolation, retention and switch cells existing with consistent and complete attributes.
- The `voltage_map` and `pg_pin` groups existing in a multiple `pg_pin` based library with consistent attributes.
- The same `power_down_function` for output or inout pins.
- Power data existing for all operating conditions.

Use the `set_check_library_options -mcmm` command to specify the following checks for a multicorner-multimode flow:

- Different PVT values for different characterizations between two libraries for different operation conditions, that is, libraries with same-name cells that have different nominal PVT values.
- Same cell attributes such as `dont_use`, `dont_touch`, and black box.
- Same `power_down_function` for output or inout pins.
- Power data existing for all operating conditions.

Use the `set_check_library_options -tolerance {`*`type rel_tol abs_tol`*`}` command to specify a relative tolerance and an absolute tolerance for characterization value comparison, such as delay values.

The options for the `type` variable are time, power, current, and capacitance. If the `type` variable is not specified, the values are for output load capacitance. To specify an absolute tolerance, do not include the unit. The unit in the first library is used.

The following example specifies tolerances for time and power:

```
-tolerance {time 0.1 0.2 power 0.3 0.4}
```

If you do not specify tolerances for a type, the default values are used. The following default values are set in compliance with PrimeTime and the Library Quality Assurance System:

	Load Index	Time	Power
Relative Tolerance	0.01	0.04	0.04
Absolute Tolerance	0.001pF	0.015ns	5pW

If you want to determine whether two libraries are identical by comparing them group by group and attribute by attribute, it is recommended that you set the tolerances to smaller values.

Use the `set_check_library_options -validate` command to specify the following checks for library characterization, using the Library Quality Assurance System:

- The same output load index for each individual cell between CCS and NLDM models and the same load indexes for CCS driver and CCS receiver.
- The same number of NLDM delay and slew indexes and values.
- The difference of delays between NLDM and CCS timing within an acceptable range for both compact CCS and expanded CCS.
- That two CCS driver model current waveforms, including compact CCS with expanded CCS data, are consistent within an acceptable margin.

The `check_library` command also reports on the existence of inverter and buffer cells and total numbers among libraries. Use the `set_check_library_options -logic` command to specify that the `check_library` command perform this check.

Library Checking Report Format

The `check_library` command performs checks on physical and logical libraries and generates reports. By default, the `check_library` command generates reports in ASCII format. For characterization data validation, comparison and analysis checks, you can specify that the report be in CSV (comma separated value) format.

The CSV format reports have the following two sections:

- Header and summary
- Comparison values by group type - timing, power, or noise

To generate library checking reports in CSV format, you must first specify the report format for the check_library command using the set_check_library_options command. The syntax used to specify the report format is

```
set_check_library_options -report_format {csv[=csv_file_dir]}
```

By default, the reports are stored in the current working directory. To specify the location to store the reports, use the csv_file_dir argument.

When you perform multiple runs with the same settings, the reports directory is backed up to the csv_file_dir_bak directory.

Reporting on Logic Library Options

To report on the values of the logic library consistency checking options set by the set_check_library_options command, use the report_check_library_options command.

```
% report_check_library_options -logic
```

You can use the -default option to report the default options.

Validating Physical Libraries

You can use the check_library command to check the quality of a Milkyway design library. To perform these physical library checks, you must enable the specific checks you want by running the set_check_library_options command (by default, the check_library command does not perform physical library checks).

Use the check_library command to check the data and quality for a specified library, as shown in the following example:

```
icc_shell> check_library -mw_lib_name testing123.mw
```

To see the enabled physical library checks, run the `report_check_library_options -physical` command. Table 6-2 shows the physical library checks and the options used to enable them.

Table 6-2 Physical Library Checks

Use this option	To check for
`-routeability`	The on-track accessibility of the physical pins.
`-drc`	Design rule constraint (DRC) violations in the routing (FRAM) view of the cells (from `cmCheckLibrary`). This requires write permission for the library directory.
`-view_comparison`	Consistency between the layout (CEL) view and the routing (FRAM) view of the cells in reference library. Missing views and mismatched views from earlier FRAM views are reported.
`-antenna`	Missing antenna properties for cells and missing antenna rules for routing layers.
`-signal_em`	Missing signal electromigration rules for routing layers.
`-same_name_cell`	Cells with identical names in different reference libraries. The names of the cells are reported.
`-rectilinear_cell`	Coordinates of cells with rectangular or rectilinear boundaries and the boundaries for (macro) cells.
`-phys_property {types}`	Physical properties. You can specify one or more of the following types: place, route, cell, and metal_density.
`-physical_only_cell`	Physical-only cells, such as filler cells with and without metal, diode cells with antenna properties, and corner cells.
`-tech_consistency`	Consistency in the technology data between the design library and the reference libraries.
`-tech`	The quality of the technology data for a reference library (from `cmCheckLibrary`).

Table 6-2 Physical Library Checks (Continued)

Use this option	To check for
`-cells`	Specifies a list of cell names. If not specified, all cells in the library are checked.
`-all`	All options.
`-reset`	Resets all options to the default values.

Physical Library Checking Option Descriptions

`-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 of the report, `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 run 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 6-17.

`-view_comparison`

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 6-18.

`-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 the hierarchical antenna property, it is counted as missing the property. For antenna rules, the 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 6-18.

`-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 6-18.

`-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" on page 6-18.

`-rectilinear_cell`

Checks for cells with rectilinear boundaries including cell types and coordinates. For a report example see "Report for -rectilinear_cell Option" on page 6-19.

`-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 6-19.

`-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" on page 6-19.

`-tech_consistency`

Specify checks for technology data consistency between the main or design library specified in the `check_library` command and each reference library. It looks for problems such as missing layer data and mismatched technology data. For a report example, see "Report for -tech_consistency Option" on page 6-20.

`-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.

Verifying the Electromigration Constraints

To perform a check on the signal electromigration constraints in the library, use the following commands:

```
% set_check_library_options -signal_em
1
% check_library

#BEGIN_CHECK_LIBRARY
...
#BEGIN_CHECK_SIGNALEM
... (signal EM checking results here) ...
#END_CHECK_SIGNALEM
...
#END_CHECK_LIBRARY
```

If the signal electromigration constraints are not defined in the design library, you need to add them. In the Milkyway Environment, to load the signal electromigration constraints in .plib format into the design library, use a script similar to the following:

```
read_plib

setFormField "Import Plib" "Library Name" "design_library"
setFormField "Import Plib" "Tech PLIB/PDB File" \
 "signal_em_constraints.plib"
setFormField "Import Plib" "Overwrite Existing Tech" "1"
formOK "Import Plib"
```

If the electromigration constraints are in Advanced Library Format (ALF), you can read them into the Milkyway library with the following IC Compiler command:

```
icc_shell> set_mw_technology_file -alf alf_file_name designlib_name
```

The existing technology data in the current design library is not affected. However, if the existing design library has electromigration data, that information is completely replaced by the new information read from the ALF file.

The `set_mw_technology_file` command accepts only one of the options `-technology`, `-plib`, or `-alf` in any one usage of the command. To load in a technology file and a separate ALF file, two commands are required. For example,

```
icc_shell> set_mw_technology_file -technology mytech.tf my_designlib
icc_shell> set_mw_technology_file -alf my_alf.alf my_designlib
```

To load in a .plib file and a separate ALF file, two commands are required. For example,

```
icc_shell> set_mw_technology_file -technology myplib.plib my_designlib
icc_shell> set_mw_technology_file -alf my_alf.alf my_designlib
```

In this example, if the .plib file contains electromigration rules, those rules are completely replaced by the ones read in from the .alf file.

If you can get an updated incremental .plib file containing the electromigration data from your vendor, you can read it into the Milkyway database with the following IC Compiler command:

```
icc_shell> set_mw_technology_file -plib plib_file_name design_lib_name
```

Library Check 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 use the `report_check_library_options` command to report the values of the options set by the `set_check_library_options` command, as shown:

```
report_check_library_options -option_name
```

The `report_check_library_options` command has the following options:

`-physical`

Reports physical library checking options set by the `set_check_library_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_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
```

Index

M

N

O

P

R

S

T

U

V

W