

# **Library Data Preparation for IC Compiler™ User Guide**

---

Version J-2014.09-SP4, March 2017

**SYNOPSYS®**

# Copyright Notice and Proprietary Information

©2017 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

## 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.

## Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

All other product or company names may be trademarks of their respective owners.

## Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.  
690 E. Middlefield Road  
Mountain View, CA 94043  
[www.synopsys.com](http://www.synopsys.com)

## Copyright Notice for the Command-Line Editing Feature

© 1992, 1993 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1.Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3.All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

- 4.Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Copyright Notice for the Line-Editing Library

© 1992 Simmule Turner and Rich Salz. All rights reserved.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

- 1.The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
- 2.The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
- 3.Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
- 4.This notice may not be removed or altered.



# Contents

---

About This User Guide . . . . .	x
Customer Support. . . . .	xii
<b>1. Library Data Preparation</b>	
Libraries Used in the IC Compiler Tool . . . . .	1-2
Physical Libraries: Milkyway Database . . . . .	1-2
Logic Libraries: Synopsys .db Files . . . . .	1-4
Library Data Preparation. . . . .	1-5
Milkyway Library Editing in the IC Compiler Tool. . . . .	1-7
Library Preparation in the Milkyway Environment Tool . . . . .	1-9
Library Checking. . . . .	1-10
<b>2. Milkyway Environment Tool</b>	
Starting the Milkyway Environment Tool . . . . .	2-2
Command Entry Methods. . . . .	2-3
GUI Menu Command Entry . . . . .	2-3
Tcl Command Entry. . . . .	2-7
Tcl Help. . . . .	2-9
Tcl Command Scripts . . . . .	2-10
Tcl Command Results . . . . .	2-10
Tcl Variables . . . . .	2-11
Scheme Command Entry . . . . .	2-11
Online Help. . . . .	2-12

Library Data Preparation Flow . . . . .	2-13
Creating a New Milkyway Library . . . . .	2-15
Extended Milkyway Layers . . . . .	2-16
<b>3. Library Preparation Using GDSII and OASIS</b>	
GDSII to Milkyway. . . . .	3-2
Cell-Type Definition File . . . . .	3-3
Layer Mapping File: GDSII or OASIS to Milkyway . . . . .	3-5
Milkyway to GDSII. . . . .	3-8
Objects in Milkyway That Can Be Streamed Out. . . . .	3-9
Layer Mapping File: Milkyway to GDSII or OASIS . . . . .	3-10
OASIS to Milkyway . . . . .	3-14
Milkyway to OASIS . . . . .	3-15
Double-Patterning Cell-Level Mask Swapping . . . . .	3-17
<b>4. Library Preparation Using LEF/DEF</b>	
LEF and DEF Data . . . . .	4-2
LEF to Milkyway . . . . .	4-3
Automated LEF Input Flow . . . . .	4-5
Advanced LEF Input Flow . . . . .	4-7
LEF to Milkyway Layer Mapping File . . . . .	4-9
Mixed Input Flows Using LEF, GDSII, TF, CLF, and PLIB . . . . .	4-10
Milkyway to LEF . . . . .	4-11
DEF Input and Output. . . . .	4-12
Importing DEF Files. . . . .	4-13
Exporting DEF Files. . . . .	4-14
Recommended DEF Flows . . . . .	4-16
<b>5. Library Cell Preparation</b>	
Identify Power and Ground Ports . . . . .	5-2
Flatten Hierarchical Cells . . . . .	5-3
Extract Blockage, Pin, and Via Information for FRAM View . . . . .	5-4
Extraction From Standard Cells and Pad Cells . . . . .	5-6

Extract Blockage Options . . . . .	5-7
Extract Pin by Text Options . . . . .	5-9
Extract Via Options . . . . .	5-12
Extraction From Macros . . . . .	5-12
Extract Blockage Options . . . . .	5-13
Extract Pin by Text Options . . . . .	5-15
Extract Via Options . . . . .	5-16
Identifying Pins . . . . .	5-16
Text on the Same Layer as Its Geometry . . . . .	5-17
Text on a Different Layer From Its Geometry . . . . .	5-17
All Text on the Same Layer . . . . .	5-17
Creating Blockage Areas . . . . .	5-17
Horizontal Via Placement . . . . .	5-19
Exclusion of Internal Pins . . . . .	5-20
Via Regions . . . . .	5-21
Set the Place-and-Route Boundary . . . . .	5-22
Set the Multiple-Height Boundary . . . . .	5-25
Double-Height Standard Cell With Two Power Supplies . . . . .	5-25
Define the Unit Tile Wire Tracks . . . . .	5-27
Define the Wire Tracks . . . . .	5-29
Check the Wire Tracks . . . . .	5-30
Define the Antenna Properties . . . . .	5-31
Antenna Data Preparation for Standard Cells . . . . .	5-31
Gate Size . . . . .	5-32
Antenna Area . . . . .	5-32
Diode Protection Value . . . . .	5-33
Removing Diode Properties . . . . .	5-33
Antenna Data Preparation for Macros . . . . .	5-33
Antenna-Fixing Diode Cells . . . . .	5-36
Import Pin Attributes From .db Files . . . . .	5-36
Create Cell Properties . . . . .	5-37
Creating a Cell Property in Scheme Mode . . . . .	5-37
Creating a Cell Property in Tcl Mode . . . . .	5-38
<b>6. Library Checking</b>	
Library Checking Overview . . . . .	6-2

Validating Logic Libraries . . . . .	6-5
General Logic Checks . . . . .	6-6
Specific Logic Checks . . . . .	6-7
Special Checks . . . . .	6-7
Library Checking Report Format . . . . .	6-8
Validating Physical Libraries . . . . .	6-9
Verifying the Electromigration Constraints . . . . .	6-13
Library Check Reporting Examples . . . . .	6-14
 <b>7. Using Milkyway Libraries in IC Compiler</b>	
Milkyway Library Structure . . . . .	7-2
Milkyway Reference Libraries . . . . .	7-2
Setting Milkyway Reference Libraries . . . . .	7-4
Milkyway Reference Control File . . . . .	7-6
Listing All Cells and Their Physical Views . . . . .	7-6
Opening and Closing Milkyway Libraries . . . . .	7-7
Opening and Closing Milkyway Cells . . . . .	7-9
Creating New Milkyway Libraries and Cells . . . . .	7-13
Creating and Copying Milkyway Libraries . . . . .	7-13
Creating and Copying Milkyway Cells . . . . .	7-15
Splitting a Milkyway Design Library . . . . .	7-16
Reading and Writing GDSII and OASIS in the IC Compiler Tool . . . . .	7-16
read_stream . . . . .	7-16
write_stream . . . . .	7-17
Milkyway Database Versions . . . . .	7-19
 <b>Appendix A. Aserver, Amonitor, and Null Display</b>	
AServer and AMonitor . . . . .	A-2
Null Display . . . . .	A-2
 <b>Appendix B. DEF 5.6 and 5.7 Supported Syntax</b>	
Supported DEF Version 5.6 and 5.7 Syntax . . . . .	B-2

## Index



# Preface

---

This preface includes the following sections:

- [About This User Guide](#)
- [Customer Support](#)

---

## About This User Guide

The *Library Data Preparation for IC Compiler User Guide* describes how to use the IC Compiler and Milkyway™ Environment tools to prepare libraries that can be used for physical implementation of chip designs. The book describes Milkyway libraries, library data preparation from GDSII and OASIS, library data preparation from LEF/DEF, library cell preparation, and library checking.

---

## Audience

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

---

## Related Publications

For additional information about the Milkyway tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

<https://solvnet.synopsys.com/DocsOnWeb>

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

- IC Compiler™
- Milkyway Environment™

---

## Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *IC Compiler Release Notes* and *Milkyway™ Environment Release Notes* on the SolvNet site.

To see the *IC Compiler Release Notes* and *Milkyway Environment Release Notes*,

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

<https://solvnet.synopsys.com/DownloadCenter>

2. Select the tool name, and then select a release in the list that appears.

---

## Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
<b>Courier bold</b>	Indicates user input—text you type verbatim—in examples, such as <code>prompt&gt; write_file top</code>
[ ]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low   medium   high</code>
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl 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

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to 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 sign up for an account.

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

---

### 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 support case to your local support center online by signing in to the SolvNet site at <https://solvnet.synopsys.com>, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail [support\\_center@synopsys.com](mailto: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 North America.
  - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>

# 1

## Library Data Preparation

---

Before you use the IC Compiler tool, logic libraries and physical libraries must be created that accurately reflect the characteristics of the available technology and cells that will be used to fabricate the chip. The physical libraries are prepared from layout data provided by an external source. The processing of this information is called library data preparation, as described the following sections:

- [Libraries Used in the IC Compiler Tool](#)
- [Library Data Preparation](#)
- [Library Checking](#)

---

## Libraries Used in the IC Compiler Tool

The IC Compiler tool performs physical implementation, including placement, clock tree synthesis, routing, and optimization of a chip design. To perform these tasks, it needs to read in a design netlist and both physical and logic libraries. These libraries contain information about the cells used in the design netlist.

A physical library contains information about the geometry of the cells that are placed in the design and connected with power, ground, clock, and signal routes. This library information includes the cell dimensions, border, pin locations, and mask layers, as well as technology information such as wire tracks, antenna rules, and electromigration data.

A logic library contains functional information about these cells, including the logic function, timing characteristics, and power characteristics. The IC Compiler tool needs this information to perform analysis and optimization of the design based on the physical characteristics and the timing, power, noise, and signal integrity requirements.

---

### Physical Libraries: Milkyway Database

In typical IC Compiler flows, the physical library information is contained in the Milkyway™ database, the unifying design storage format for the Synopsys Galaxy™ implementation platform. Galaxy tools such as Design Compiler, IC Compiler, and Formality can access the design and library information contained in the database. The database contains not only leaf-level physical cell information and technology information, but also design-specific physical information such as the placement and routing of the design.

The Milkyway database is organized as a hierarchy of data files. However, you should not create, delete, copy, or edit these files directly using operating system commands such as `cp` and `rm`. Instead, you should access the database by using a Synopsys tool and use the tool commands to read, write, or change the database contents. This will ensure the consistency and integrity of the database.

In the IC Compiler tool, you open a Milkyway database with the `open_mw_lib` command. By default, opening a Milkyway library makes that library accessible to the tool for both reading and writing physical design information. You can open no more than one Milkyway design library at a time. However, the design can contain references to cells contained in other Milkyway libraries, called reference libraries. Multiple users can open same design library in different sessions. However, only one user at a time can have permission to write into a Milkyway design library.

The basic unit of information in a Milkyway library is the cell. A cell is a representation of a physical structure in the chip layout, which can be something as simple as an I/O pad or as large and complex as an entire chip. You open a cell for editing by using the `open_mw_cel` command. The cell must be contained in the Milkyway library that is currently open.

A design is typically built as a hierarchy of cells. The entire chip is a single cell built out of lower-level blocks, which are also cells. These blocks are built out of smaller blocks, and so on, down to the level of leaf-level cells, which are gate-level standard cells.

The Milkyway database can contain different representations of the same cell, called “views” of that cell. These are the main types of views used in the IC Compiler tool:

- **CEL view:** The full layout view of a physical structure such as a via, standard cell, macro, or whole chip; contains placement, routing, pin, and netlist information for the cell.
- **FRAM view:** An abstract representation of a cell used for placement and routing; contains only the metal blockages, allowed via areas, and pins of the cell.
- **FILL view:** A view of metal fill, which is used for chip finishing and has no logic function, created by the `signoff_metal_fill` command in the IC Compiler tool.
- **CONN view:** A representation of the power and ground networks of a cell, created by the PrimeRail or IC Compiler tool and used by PrimeRail for IR drop and electromigration analysis.
- **ERR view:** A graphical view of physical design rule violations found by verification commands in the IC Compiler tool such as `verify_zrt_route` or `signoff_drc`.

A design stored in a Milkyway library must have at least a CEL view, which contains all of the cell information needed for placement, routing, and mask generation. This includes placement information such as tracks, site rows, and placement blockages; routing information such as netlist, pin, route guide, and interconnect modeling information, and all mask layer geometries, which are used for final mask generation.

Each macro cell typically has both a CEL view and a FRAM view. The FRAM view is an abstraction of the cell containing only the information needed for placement and routing: the metal blockage areas where routes are not allowed, the allowed via areas, and the pin locations. The process of creating a FRAM view from a CEL view is often called blockage, pin, and via (BPV) extraction.

Each gate-level standard cell can also have both a CEL view and a FRAM view. The FRAM view is used for placement and routing, whereas the CEL view is used only for generating the final stream of mask data for chip manufacturing.

The IC Compiler tool creates additional, temporary views in the Milkyway database while it performs tasks such as routing, pin assignment, and hierarchy flattening. It empties or deletes these views upon completion of each task. You can see these views in the directory structure of the Milkyway library during performance of the task. However, you should not attempt to modify or use these views outside of the IC Compiler tool, except the case where the task is abnormally terminated. The views called `ROUTE`, `PINASSIGN`, and `SMASH` can be safely removed so that you can restart the corresponding process from the beginning.

---

## Logic Libraries: Synopsys .db Files

The cell logic, timing, and power information is typically contained in a set of Synopsys database (.db) files produced by Library Compiler. The creator of the library determines the electrical behavior of the leaf-level cells by using a simulation-based characterization tool such as Liberty NCX in combination with a circuit simulator such as HSPICE. The characterization tool produces a set of files in Liberty (.lib) format.

The Liberty (.lib) files are ASCII-format files that fully describe the cell logic, timing, and power characteristics of the leaf-level logic cells. Library Compiler compiles the .lib files to produce .db files, which contain the same information as the .lib files, but in a compiled binary format that is more efficient for Galaxy tools to use.

In the IC Compiler tool, you specify the .db files to use for the design by setting the `search_path`, `target_library`, and `link_library` variables:

- The `search_path` variable specifies the directory paths in which the .db files and other files needed by the tool can be found.
- The `target_library` variable specifies the .db library files containing the logic cells that can be used for optimization, for example, different NAND gates having various areas, drive strengths, delays, and power usage.
- The `link_library` variable specifies the .db libraries containing all the logic cells that can be used to resolve hierarchical references in the design during execution of the `link` command.

For example, an IC Compiler script might contain commands like these:

```
set_app_var search_path      "/remote/tech/libs ~/LIBS/mysdc ~/LIBS/def"  
set_app_var target_library  "stdcell.db"  
set_app_var link_library    "* stdcell.db macrocell.db pll.db memory.db"
```

The list of libraries set in the `link_library` variable should include all of those set in the `target_library` variable. The `link_library` variable should also include an asterisk character, which causes the link command to search through all designs already loaded into memory to find referenced cells. It should also include the names of any libraries containing cells that exist in the design but are not targets of optimization, such as macro and RAM cells.

A design attribute (not variable) called `local_link_library` can be used to specify additional libraries that can be used for design linking. It can be set with the `set_local_link_library` command.

The names of the cells in the logic libraries must match the names of the corresponding cells in the physical libraries in the Milkyway database. You can verify that the logic and physical libraries properly match by using the `check_library` command.



---

## Library Data Preparation

Before you use the IC Compiler tool, logic and physical libraries must be created that accurately reflect the characteristics of the available technology and contain the standard cells that will be used to fabricate the chip. The IC Compiler tool needs these libraries to place, route, analyze, and optimize the chip design.

The Synopsys tools for creating .db logic libraries are Liberty NCX, HSPICE, and Library Compiler. Liberty NCX and HSPICE characterize the standard cells and generate a set of .lib files that describe the cell behavior, and Library Compiler compiles the .lib files to produce the .db files. The use of these tools is beyond the scope of this user guide. For more information about creating .db files, see the documentation for the applicable products.

The focus of this user guide is the preparation of physical library data in the Milkyway database format from the technology information provided by an outside source. This information is usually provided in one of the following standard data interchange formats:

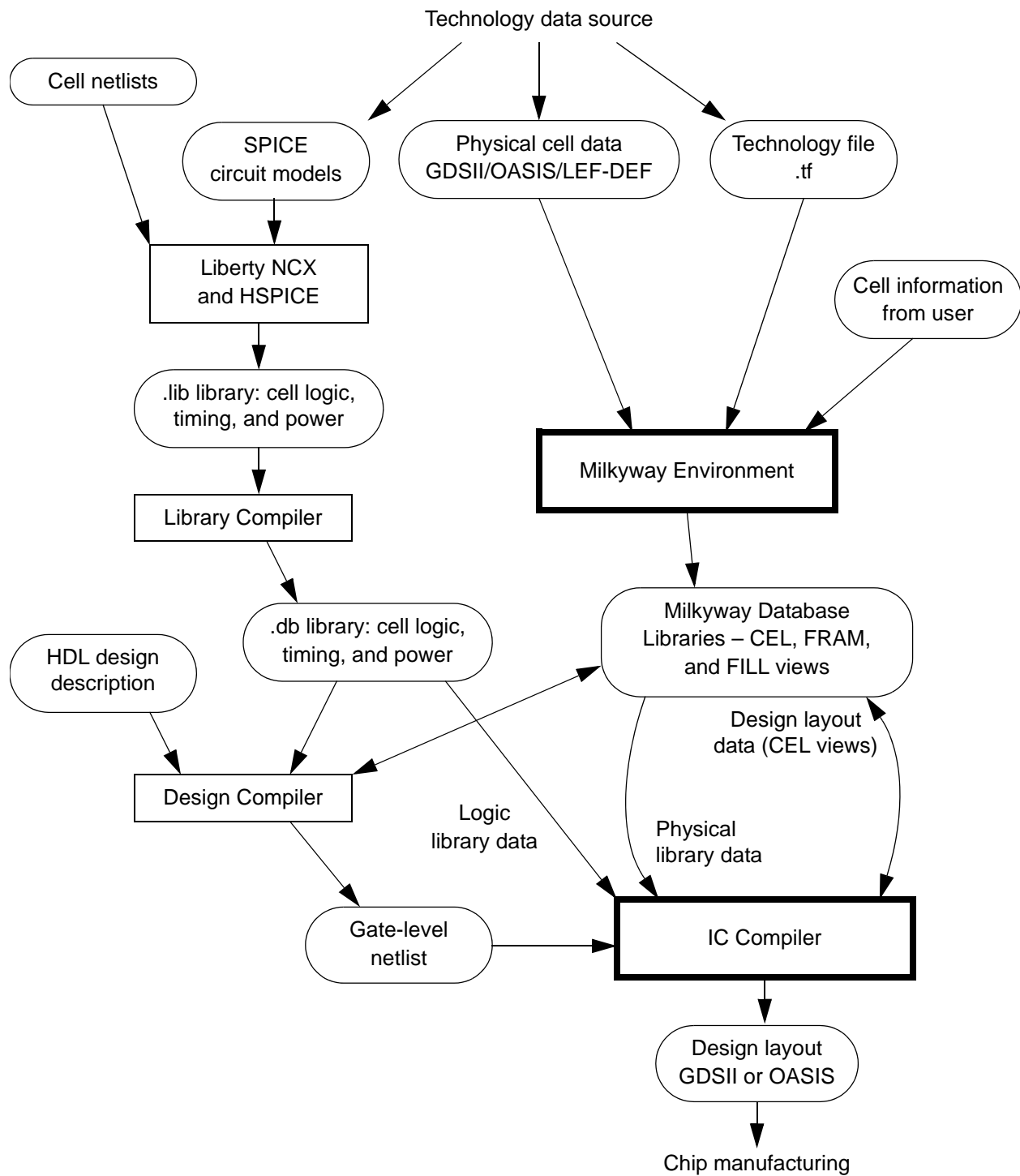
- GDSII stream format, a well-established industry-standard data exchange format for integrated circuit layout information
- OASIS, a newer data stream format designed as an improved replacement for GDSII
- LEF/DEF, a set of standard data exchange formats for physical libraries (Library Exchange Format) and design data (Design Exchange Format)

To convert this information into the Milkyway database format, you need to read in the layout information in the provided format, add library information that is not provided in the exchange format, and write the data out to the Milkyway database. In addition, you need to prepare the standard cells for use in the IC Compiler tool by identifying power and ground pins, flattening cells that are hierarchical, extracting FRAM views from CEL views, setting the place and route boundaries, and defining the wire tracks.

The tool for performing these tasks is the Milkyway Environment tool. The conversion of GDSII and OASIS data streams to the Milkyway format is described in [Chapter 3, “Library Preparation Using GDSII and OASIS.”](#) The conversion of LEF/DEF library and design data to the Milkyway format is described in [Chapter 4, “Library Preparation Using LEF/DEF.”](#) The further preparation of cells for use in the IC Compiler tool is described in [Chapter 5, “Library Cell Preparation.”](#)

[Figure 1-1](#) shows the typical library preparation and usage flow for the IC Compiler tool.

Figure 1-1 Typical Physical and Logic Library Preparation and Usage



## Milkyway Library Editing in the IC Compiler Tool

The IC Compiler tool provides commands to perform several types of library data preparation tasks. You can open a Milkyway library and create, copy, and delete cells contained in the open library. You can also perform blockage, pin, and via extraction from CEL views to create FRAM views for placement and routing.

[Table 1-1](#) lists and briefly describes the IC Compiler commands used to open a Milkyway library and to report, check, or change the library contents.

*Table 1-1 Milkyway-Related IC Compiler Commands*

IC Compiler command	Action performed
<code>archive_design</code>	Archives a Milkyway design
<code>check_library</code>	Checks logic and physical libraries for quality and consistency
<code>close_mw_cel</code>	Closes one or more open cells in the open Milkyway library
<code>close_mw_lib</code>	Closes the current Milkyway library
<code>convert_mw_lib</code>	Converts the design data in a Milkyway library to the current database model version or to the previous version
<code>copy_mw_cel</code>	Copies a cell to create a new cell in the current Milkyway library
<code>copy_mw_lib</code>	Copies a Milkyway library
<code>create_boundary</code>	Creates or modifies a boundary of a design or a library cell
<code>create_macro_fram</code>	Extracts a macro cell from a CEL view to make a FRAM view
<code>create_mw_cel</code>	Creates a new cell in the current Milkyway library
<code>create_mw_lib</code>	Creates a new Milkyway library
<code>current_mw_cel</code>	Specifies or reports the current cell on which commands operate
<code>current_mw_lib</code>	Reports the current Milkyway library; creates a collection containing that library
<code>decrypt_lib</code>	Decrypts a library encrypted with the <code>encrypt_lib</code> command in the Milkyway Environment tool
<code>extract_blockage_pin_via</code>	Extracts blockage, pin, and via information from a CEL view to make a FRAM view

*Table 1-1 Milkyway-Related IC Compiler Commands (Continued)*

<b>IC Compiler command</b>	<b>Action performed</b>
<code>get_mw_cels</code>	Creates a collection of cells in the current Milkyway library that meet specified criteria
<code>list_mw_cels</code>	Lists the cells contained in the current Milkyway library
<code>move_mw_cel_origin</code>	Moves the origin of a Milkyway cell, changing the coordinates of all objects in the cell
<code>open_mw_cel</code>	Opens a cell in the current Milkyway library for viewing or editing
<code>open_mw_lib</code>	Opens a Milkyway library for editing, making it the current Milkyway library
<code>read_stream</code>	Reads a GDSII or OASIS data stream into the current Milkyway library
<code>rebuild_mw_lib</code>	Rebuilds a Milkyway library by scanning all cells in the library directory
<code>remove_mw_cel</code>	Removes cells from the current Milkyway library
<code>report_milkyway_version</code>	Reports the data version number, database model version, and creation information for a cell
<code>rename_mw_cel</code>	Renames a cell in the current Milkyway library
<code>rename_mw_lib</code>	Renames a Milkyway library
<code>report_mw_lib</code>	Reports the unit range or reference libraries of a Milkyway library
<code>save_mw_cel</code>	Saves an edited cell into the current Milkyway library
<code>set_mw_lib_reference</code>	Sets the lower-level reference Milkyway libraries associated with a given Milkyway library
<code>set_mw_technology_file</code>	Sets a technology file (.tf, .plib, or .alf) associated with the current Milkyway library
<code>split_mw_lib</code>	Splits a Milkyway library into multiple libraries, each containing one top-level cell
<code>uniquify_fp_mw_cel</code>	Creates copies of a master library cell, one copy for each instance of the cell used in the design

*Table 1-1 Milkyway-Related IC Compiler Commands (Continued)*

IC Compiler command	Action performed
<code>write_mw_lib_files</code>	Writes the library technology information or library reference information to a file
<code>write_stream</code>	Writes one or more cells in the Milkyway database to a GDSII or OASIS data stream

The IC Compiler tool has some commands that resemble Milkyway Environment library data preparation commands but are not used for library data preparation:

- The `read_lib` and `write_lib` commands can read and write library files of various types. However, they are not used to read or write Milkyway data files.
- The `read_def` and `write_def` commands can read and write DEF files. However, they are used only to read and write floorplanning and scan chain information in the DEF file, not standard cell library information.

## Library Preparation in the Milkyway Environment Tool

The Milkyway Environment tool is a separate, standalone tool for preparing physical libraries from the layout data provided by an outside source. The tool offers interactive, menu-based GUI command entry, Tcl command entry, and Tcl script execution. The tool allows you to perform the following tasks, which are essential for library data preparation:

- Create standard cell libraries
- Import cell data in GDSII, OASIS, and LEF/DEF format
- Specify technology information
- Write technology information to a file
- Flatten cell hierarchy
- Specify power and ground port types
- Optimize the standard cell layout
- Extract blockage, pin, and via information to make FRAM views
- Set place and route boundaries
- Define wire tracks

Usage of the Milkyway Environment tool is described in detail in [Chapter 2, “Milkyway Environment Tool.”](#)

---

## Library Checking

After you prepare new logic libraries in .db format or physical libraries in Milkyway format, you should check the libraries for accuracy and consistency by using the `check_library` command. This command is available in both the IC Compiler and Milkyway Environment tools.

The `check_library` command checks the physical library quality, the consistency between the physical library and the corresponding logic library, and the consistency between multiple logic libraries. It verifies the consistency of cell names, pin names, area values, bus naming conventions, operating condition scaling, antenna rules, and so on. It generates a detailed report on any errors or inconsistencies that are found.

Usage of the `check_library` command is described in detail in [Chapter 6, “Library Checking.”](#)

# 2

## Milkyway Environment Tool

---

The Milkyway Environment tool is a standalone tool for preparing physical libraries from the layout data provided by an outside source. The tool offers interactive, menu-based GUI command entry, Tcl command entry, and Tcl script execution. Usage of the tool is described in the following sections:

- [Starting the Milkyway Environment Tool](#)
- [Command Entry Methods](#)
- [Online Help](#)
- [Library Data Preparation Flow](#)
- [Creating a New Milkyway Library](#)
- [Extended Milkyway Layers](#)

## Starting the Milkyway Environment Tool

The Milkyway Environment tool allows you to perform several tasks that are essential for library data preparation. Before you can use the tool, the software must be installed and licensed at your site. For information on installation and licensing, see the documentation that comes with the product release.

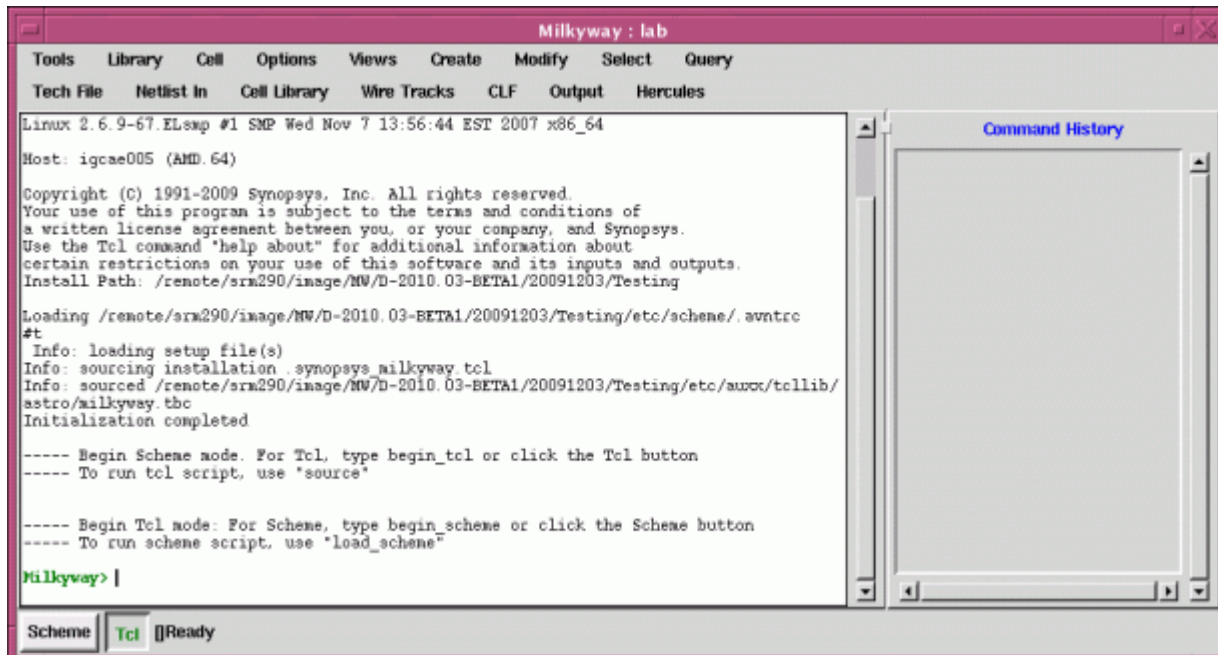
Before you start the Milkyway Environment tool, you might want to change to the directory containing your design files. This will make it easier to access your working files.

To start the Milkyway environment tool, enter the `Milkyway` command at the operating system prompt:

```
% Milkyway -galaxy -tcl
```

This invokes the Milkyway Environment tool in Galaxy mode, selects the Tcl command entry mode, and opens the Milkyway Environment graphical user interface (GUI) window. See [Figure 2-1](#).

Figure 2-1 Milkyway Environment Window



Note:

To view the Milkyway Environment invocation options, use the `Milkyway -help` command at the operating system prompt.

You control the Milkyway Environment tool by using the GUI menu commands at the top or by entering Tcl commands or Scheme commands. The large text area in the middle is for



displaying system messages and entering commands. The command history appears in a separate window on the right.

To exit from the Milkyway Environment tool and end the session, enter the command `exit` at the bottom or use the GUI menu command Tools > Quit.

---

## Command Entry Methods

The Milkyway Environment tool offers three methods of command entry:

- GUI menu commands
- Tcl commands
- Scheme commands

The GUI menu commands and associated dialog boxes provide interactive guidance for performing various tasks such as opening and closing Milkyway libraries, opening and closing cells, importing physical data in GDSII, OASIS, or LEF/DEF format, creating FRAM views from CEL views, and exporting physical design data to external tools.

The Tcl command interface provides a method for entering commands that is familiar to Synopsys product users. You enter Tcl commands at the Milkyway> command prompt, similar to other Tcl-based tools such as Design Compiler, IC Compiler, and PrimeTime.

The Scheme command interface is a command-line interface that is familiar to users of the older Synopsys place-and-route and floorplanning tools, Astro and Jupiter XT. This command interface is provided mainly for the benefit of customers still using the older tools.

The Milkyway Environment tool can operate in the Tcl or Scheme mode but not both modes at the same time. To switch from one mode to the other, click the Scheme or Tcl button in the lower-left corner of the Milkyway Environment window. The GUI menus work in both the Scheme and Tcl modes.

---

## GUI Menu Command Entry

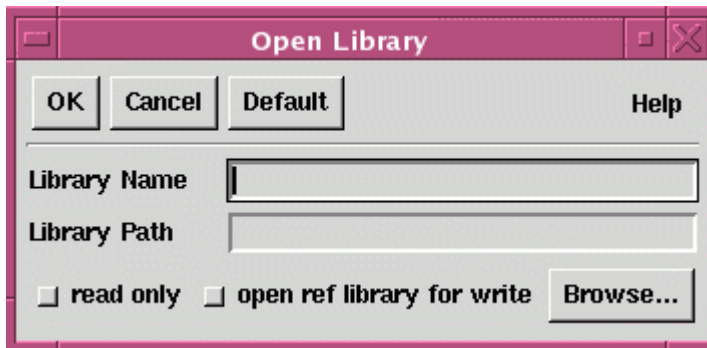
The GUI command menus provide an easy-to-use, directed method to enter commands and to perform all kinds of tasks. The top-level commands are displayed at the top of the Milkyway Environment GUI window, as shown in [Figure 2-2](#).

*Figure 2-2 Top-Level Menu Commands*

Tools	Library	Cell	Options	Views	Create	Modify	Select	Query
Tech File	Netlist In	Cell Library	Wire Tracks	CLF	Output	Hercules		

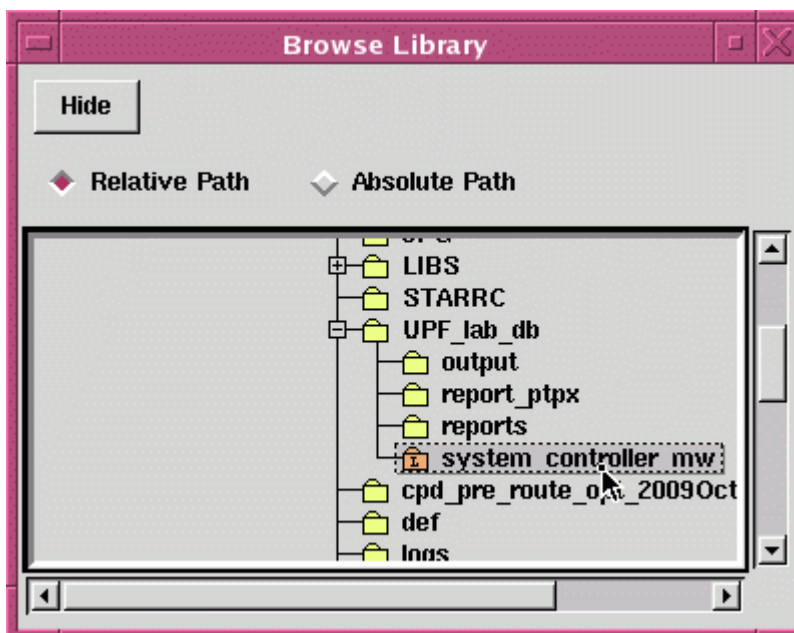
Typically, the first action taken in a session is to open an existing Milkyway library. To do this, choose Library > Open, which displays the Open Library dialog box as shown in [Figure 2-3](#).

Figure 2-3 Library > Open Dialog Box



You can enter the Milkyway library name and the directory path to the library in the text boxes provided, or you can browse and select the Milkyway library from a directory tree. Clicking the Browse button opens the Browse Library dialog box as shown in [Figure 2-4](#).

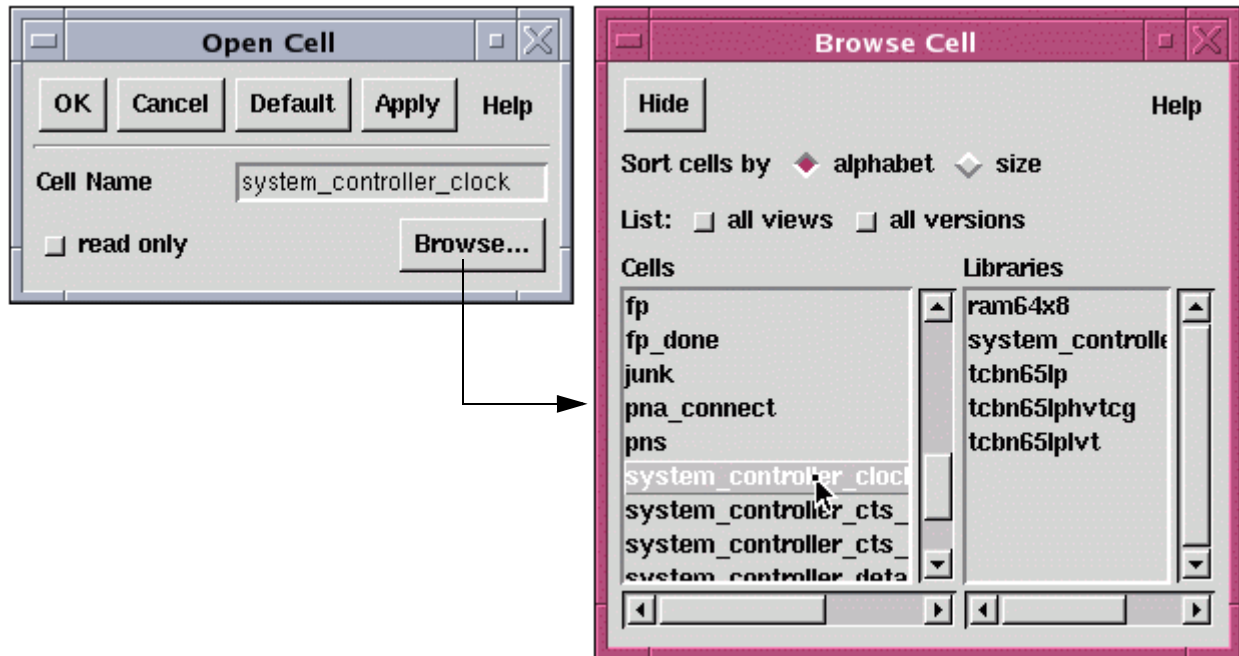
Figure 2-4 Browse Library Dialog Box



You can browse the directory tree to find the desired Milkyway library. Each Milkyway library is displayed as a brown folder marked with the letter "L." Select the desired Milkyway library, click Hide to close the Browse Library dialog box, and then click OK in the Open Library dialog box to open the library.

The next action might be to open a cell. To do this, choose **Cell > Open**, which displays the Open Cell dialog box. Then click the **Browse** button to open the Browse Cell dialog box, which lists the open Milkyway library and other referenced Milkyway libraries, as shown in [Figure 2-5](#).

Figure 2-5 Cell > Open and Browse Cell Dialog Boxes



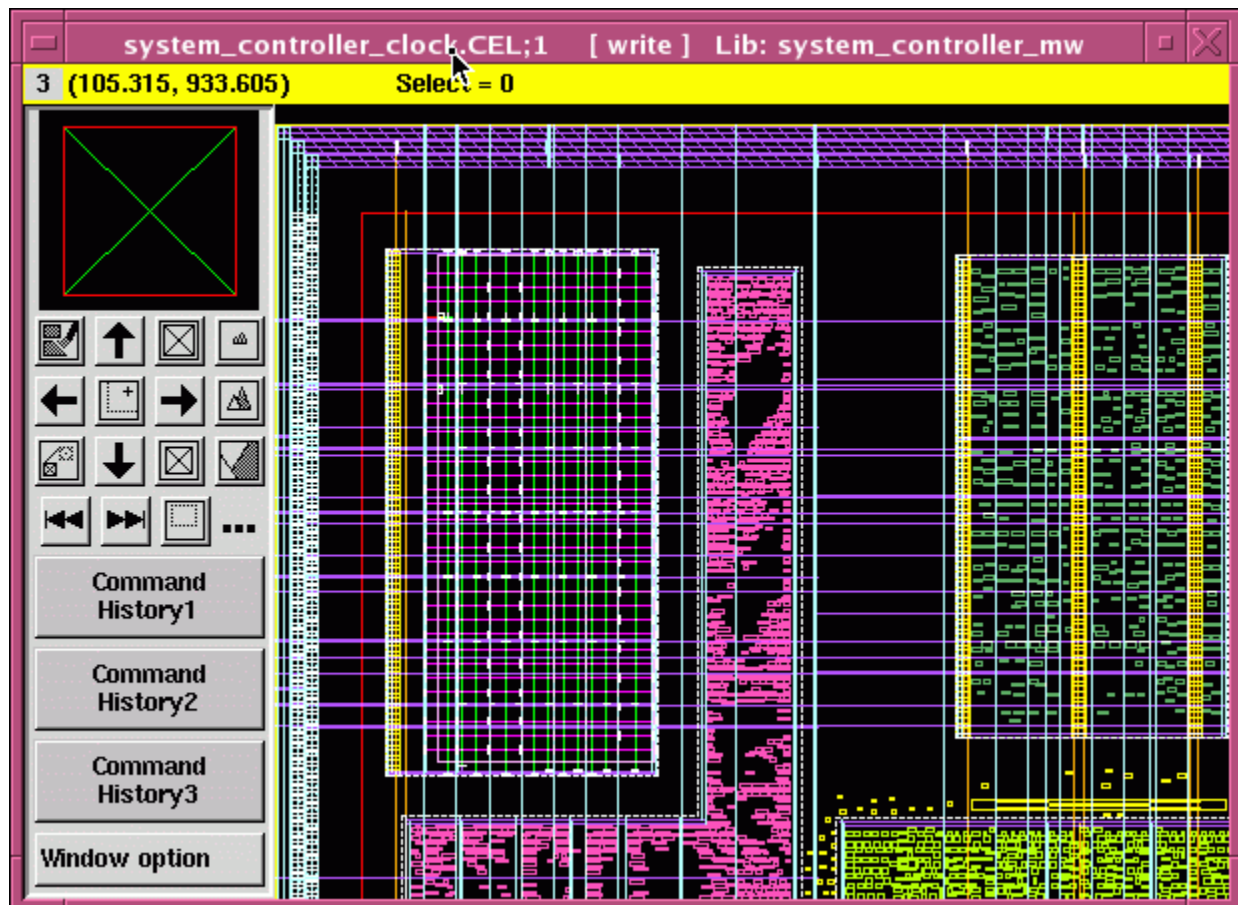
From the Libraries list on the right, select the desired Milkyway library. The cells contained in that library appear in the Cells list. Select the desired cell, and the cell name appears in the Open Cell dialog box. Click **Hide** to close the Browse Cell dialog box, and then click **OK** to open the cell.

**Note:**

If you choose a cell from a reference library rather than the open library, you must enable the read-only option in the Open Cell dialog box. Only cells in the currently open Milkyway library can be opened with read/write privileges.

When you open a cell, the Milkyway Environment tool displays the cell layout graphically in a new window like the one shown in [Figure 2-6](#).

Figure 2-6 Cell View Window



The title bar at the top of the window shows the full name of the cell, the type of access (read-only or write), and the name of the Milkyway library containing the cell. For example,

`system_controller_clock.CEL;1 [ write ] Lib: system_controller_mw`

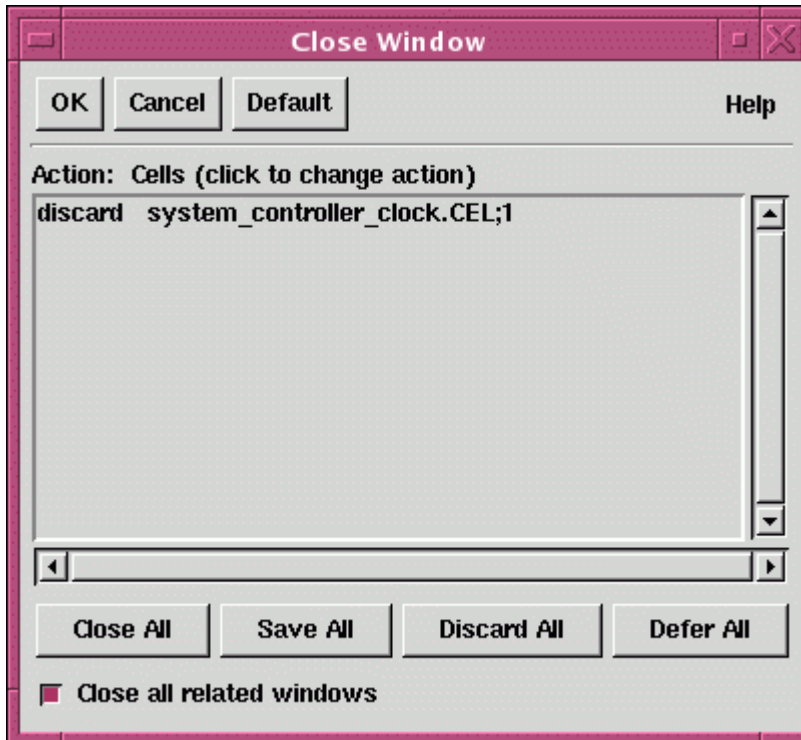
In this example, the Milkyway library name is `system_controller_mw` and the cell name is `system_controller_clock.CEL;1`. The cell view type is CEL and the cell revision number is 1.

The command buttons on the left side of the cell view window let you control the visual display parameters in detail (zoom level, colors, layers, object types, and so on) and to select and query objects in the view.

Like many windows and dialog boxes in the Milkyway Environment tool, the cell view window cannot be closed by clicking on the “X” in the upper-right corner or by using other window management commands. To close the window, you must explicitly close the cell in the Milkyway Environment tool itself.

To close a cell that is currently open, go to the main Milkyway Environment window and choose Cell > Close. This opens the Close Window dialog box like the one shown in [Figure 2-7](#).

Figure 2-7 Cell > Close Dialog Box



In the Close Window dialog box, you can select the action to take: save, close, discard, or defer, and then click OK to carry out the action and close the cell.

Some dialog boxes cannot be opened at the same time as other dialog boxes. For example, you cannot open the Close Window dialog box while an Open Cell dialog box is present. If you attempt to do so, the system message area displays a message like this:

```
command [Close Window] can't nest with [Open Cell]
```

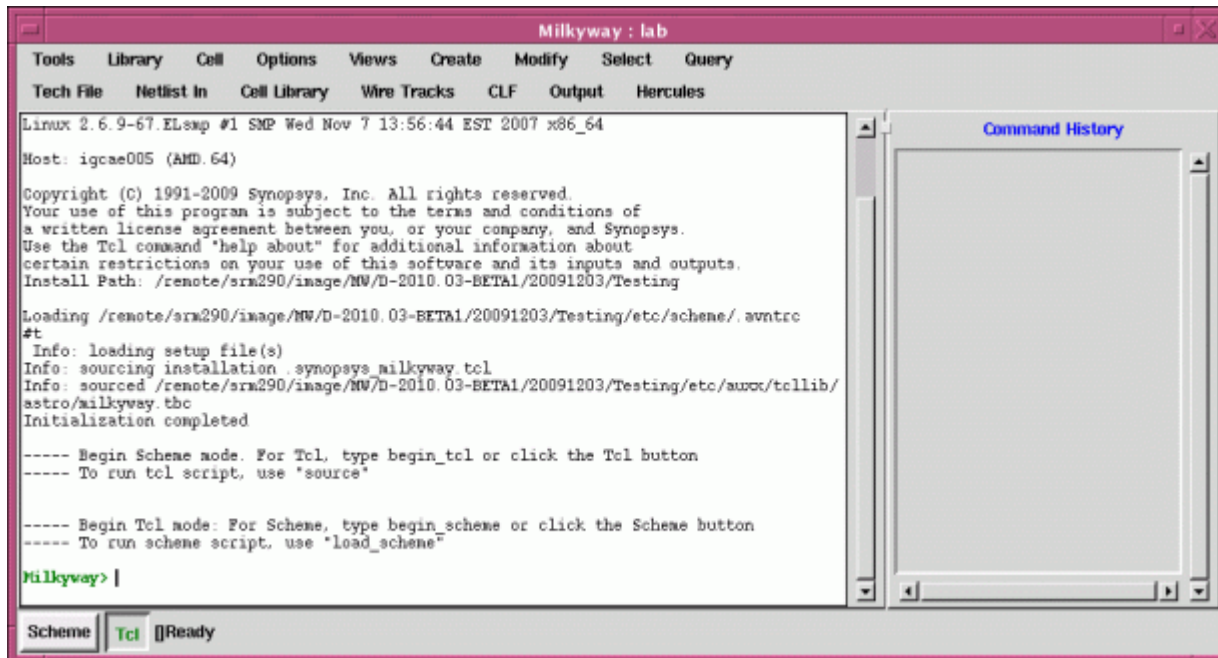
If this happens, first close the incompatible dialog box by clicking its Cancel button. Then you can open the dialog box you want to use.

---

## Tcl Command Entry

To change the Milkyway Environment tool from Scheme to Tcl mode, click the Tcl button near the lower-left corner of the window. In Tcl mode, the background of the system message area is white and the Milkyway> Tcl prompt is green. See [Figure 2-8](#).

Figure 2-8 Milkyway Environment Window in Tcl Mode



The Tcl command syntax is case-sensitive. Commands, command options, arguments, and variables generally consist of lowercase characters and must be entered as such at the Milkyway> prompt. Object names in the design are also case-sensitive.

To open a Milkyway library using the Tcl command interface, use the `open_mw_lib` command:

```
Milkyway> open_mw_lib UPF_lab_db/system_controller_mw
Reference Library: /...
Reference Library: /...
...
{"system_controller_mw"}
```

To open a cell in a Milkyway library, use the `open_mw_cel` command:

```
Milkyway> open_mw_cel -library system_controller_mw \
system_controller_clock
Information: Opened "system_controller_clock.CEL;1 from "/..."
{"system_controller_clock"}
```

The cell is displayed in a GUI viewing window, just like opening the cell using the GUI command `Cell > Open`.

For detailed information on using Tcl commands and Tcl scripting, procedures, and collections, see the *Using Tcl With Synopsys Tools* manual, available on SolvNet.

## Tcl Help

To get help on Tcl commands, use the `help` command. The `help` command by itself lists all the Tcl commands. To get a list of all commands containing the word “cell,” use the following command:

```
Milkyway> help *cell*
create_cell      # creates cell
flatten_cell     # Flatten a Milkyway design
get_cells        # create a collection of cells
...
```

To get the syntax for a particular command, use the `-help` option of that command:

```
Milkyway> create_cell -help
create_cell      # creates cell
    -from_design design_name
                    (design from which to create the cell)
    [-from_library library_name
                    (library from which to create the cell)
    [-rotation 90 | 180 | 270]
                    (Degree to which you want the cell rotated)
    ...
```

To get detailed information about a command, use the `man` command:

```
Milkyway> man create_cell
create_cell      2.  Synopsys Commands          Command Reference
```

### NAME

```
create_cell
    Creates a cell in the current design.
```

### SYNTAX

```
int create_cell
    [-from_library library]
    [-rotation {90 | 180 | 270}]
    [-mirror {x | y}]
    [-ignore_ecol]
    [-without_check_status]
    -origin {x y}
    -from_design from_design
    cell_name

    string cell_name
    string from_design
    string from_library
```

### ARGUMENTS

```
...
```

### DESCRIPTION

```
...
```

## EXAMPLES

```
...
SEE ALSO
...
```

## Tcl Command Scripts

A command script is a text file containing a sequence of Milkyway Tcl commands. To execute a script, use the `source` command:

```
Milkyway> source file_name
```

You can create scripts that use variables, loops, and conditional execution. The flow control commands `if`, `while`, `for`, `foreach`, `break`, `continue`, and `switch` determine the execution order of other commands.

Any line of text in a script file that begins with the pound sign (`#`) is a comment, which the tool ignores. Any text from a semicolon and pound sign (`;` `#`) to the end of a line is also considered comment text.

## Tcl Command Results

Every Tcl command has a result. Many commands result in “1” to indicate success or “0” to indicate failure. For example,

```
Milkyway> report_mw_cel
The library information(system_controller_mw)

0 metal layer(s) defined in the library
Wire tracks defined for all metal layers:  #t
...
===== End report for design: system_controller_clock =====
1
```

For many other commands, the result is a collection. For example, the result of the `get_nets` command is a collection of the nets in the cell:

```
Milkyway> get_nets
{"n704", "feed_data_out[8]", "thirtyfirst_row[0]", ... }
```

You can set the level of information that the message system reports into the log file by using the following command:

```
dbSetMsgLevel "low|medium|high"
```

For example, to get very detailed messages, use the following command:

```
Milkyway> dbSetMsgLevel "high"
```

You can find out the current message level setting by using the `dbGetMsgLevel` command.



## Tcl Variables

The tool offers many options that you control by setting variables. To set a variable, you use the `set_app_var` command:

```
Milkyway> set_app_var variable_name variable_setting
replace all nt_shell with Milkyway
```

When you set a variable, the displayed result is simply the new setting for the variable. For example,

```
Milkyway> set_app_var sh_enable_page_mode true
true
```

To find out the current setting for a variable, use the `printvar` command. For example,

```
Milkyway> printvar sh_enable_page_mode
sh_enable_page_mode = "true"
```

You can use the wildcard character (\*) to view multiple related settings. For example, to see a list of shell-related variables, enter

```
Milkyway> printvar sh_*
sh_allow_tcl_with_set_app_var = "true"
sh_allow_tcl_with_set_app_var_no_message_list = "true"
sh_arch = "amd64"
...
```

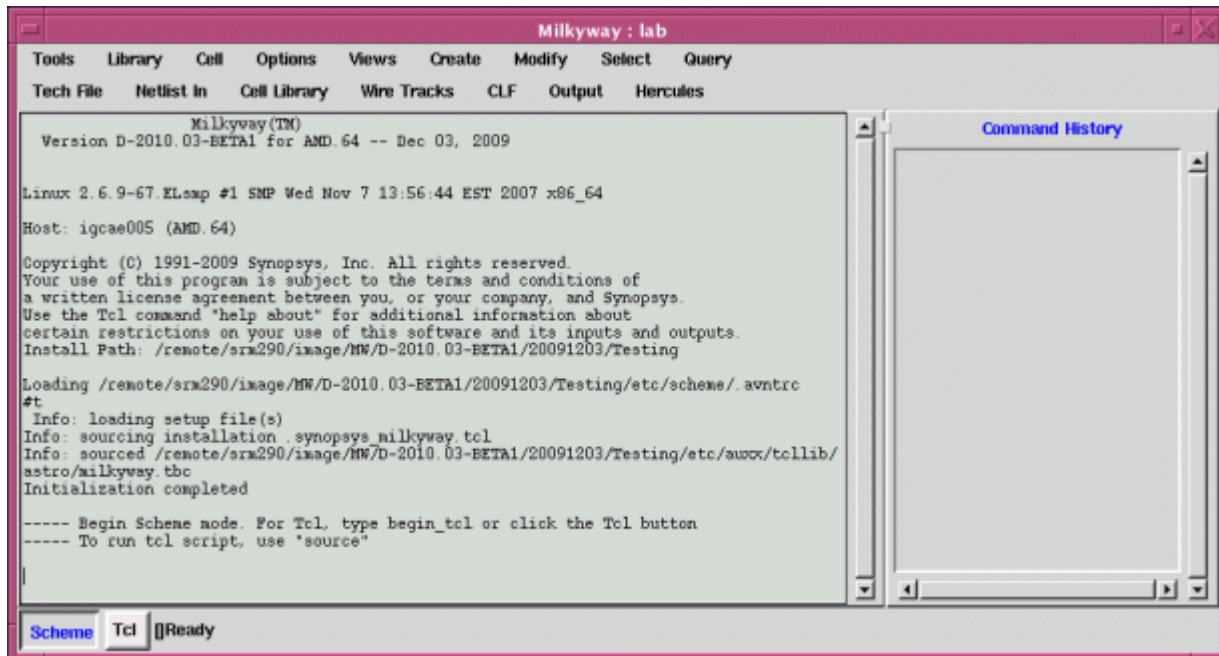
---

## Scheme Command Entry

The Scheme command interface is provided for customers who are familiar with the older Synopsys tools Astro and Jupiter XT. For users of the IC Compiler tool, either Tcl commands or GUI menu commands are recommended.

To change the Milkyway Environment tool from Tcl to Scheme mode, click the Scheme button in the lower-left corner of the window. In Scheme mode, the background of the system message area is gray, as shown in [Figure 2-9](#). You enter Scheme commands at the bottom of this area.

Figure 2-9 Milkyway Environment Window in Scheme Mode



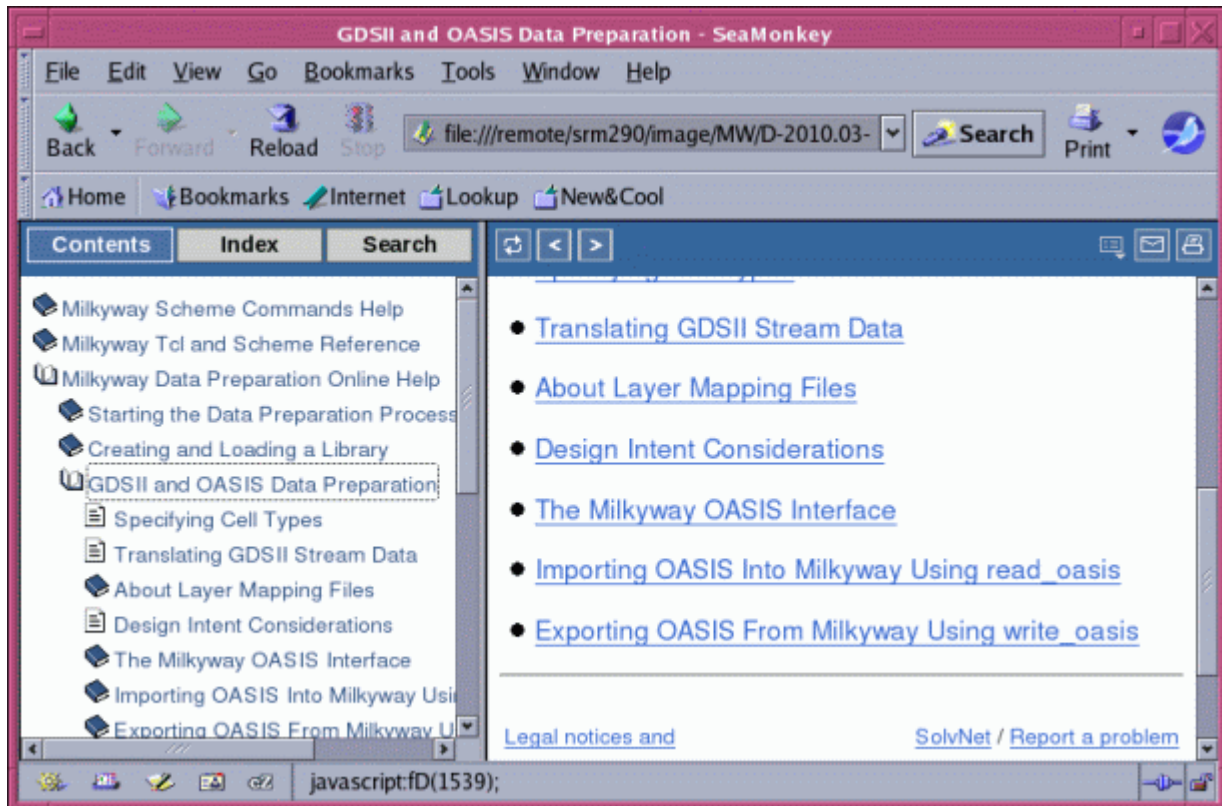
To open a Milkyway library, enter the `geOpenLib` command at the vertical-bar prompt. This opens the Open Library dialog box, just like choosing the Library > Open command in the GUI command menu. You fill in the dialog box and click OK as described in the section [“GUI Menu Command Entry” on page 2-3](#). Similarly, to open a cell, enter the `geOpenCell` command, which opens the Open Cell dialog box.

Some Scheme commands, such as `geOpenLib` and `geOpenCell`, can be entered at the Milkyway> prompt in Tcl mode. Each such command has the same effect whether entered in Scheme or Tcl mode; the same dialog box appears. However, these commands are not Tcl commands and are not listed by the `help` command in Tcl mode.

Other Scheme commands, such as `read_lef` and `read_oasis`, are also Tcl commands. When entered in Scheme mode, they open a dialog box, whereas at the Milkyway> prompt in Tcl mode, they operate as text-based commands that do not open any dialog boxes.

## Online Help

Detailed online help is available on a wide range of usage topics and commands in the Milkyway Environment tool. To view the online help, click the Help button in any GUI dialog box, or in Scheme mode only, use the `help` command. This opens a web browser window containing the online help tool, as shown in [Figure 2-10](#).

*Figure 2-10 Online Help Browser Window*

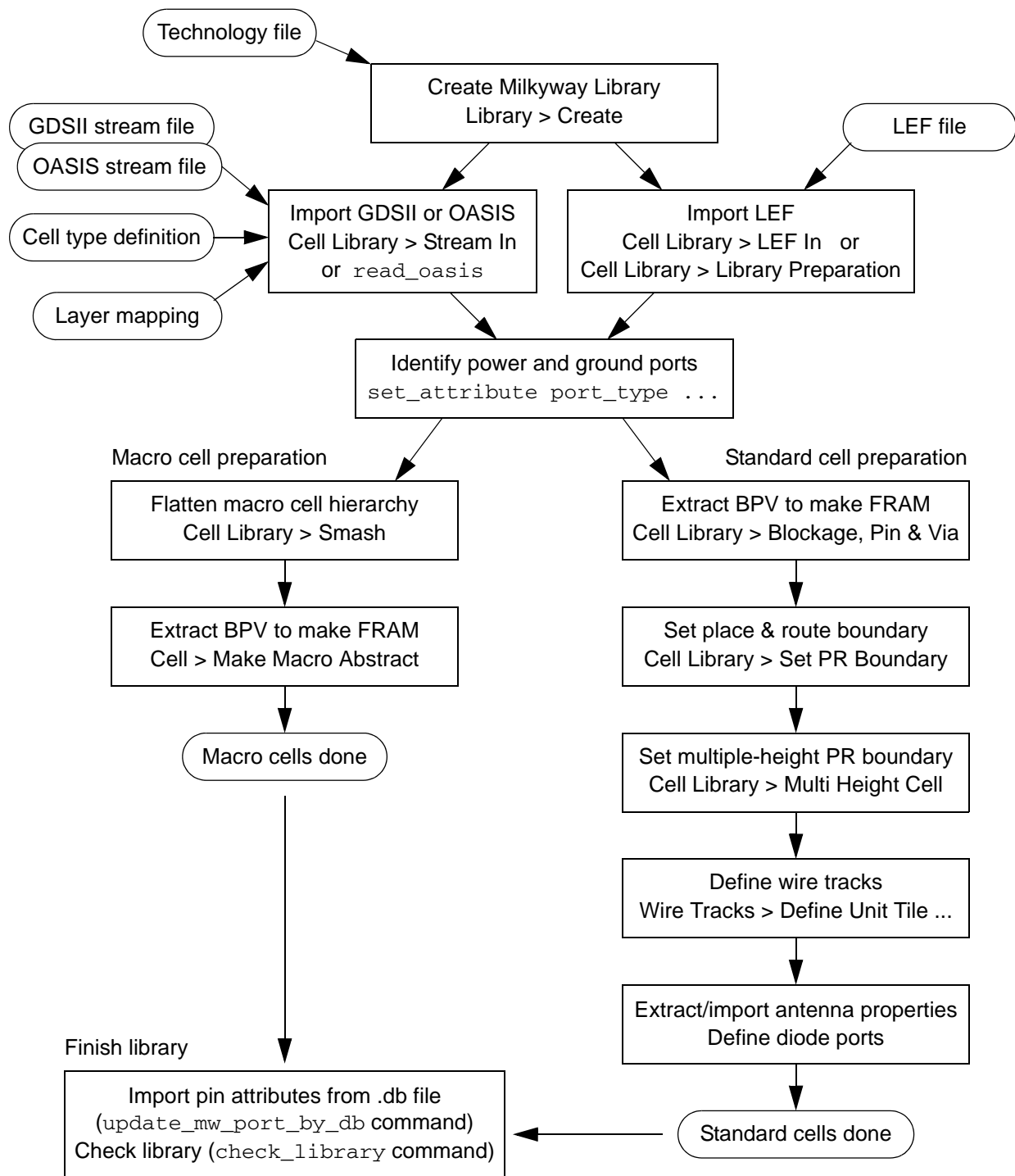
To get help on a particular topic, expand the table of contents on the left and select the topic. You can also click the Index button and select the topic from an alphabetized list or click the Search button and enter the topic in the Search box.

---

## Library Data Preparation Flow

The most important task performed in the Milkyway Environment tool is Milkyway library preparation from the physical data provided by an outside source. [Figure 2-11](#) summarizes the flow. You start by creating a new Milkyway library. Then you import the physical data in GDSII, OASIS, or LEF/DEF format and provide any additional information needed to create Milkyway CEL models for the cells in the library. You then specify the power and ground ports, create the FRAM views of the cells, and specify other physical properties required in the FRAM model. The specific steps you need to perform depend on whether you are preparing macro cells or standard cells.

Figure 2-11 Library Preparation Flow in the Milkyway Environment Tool



The first step, creating a new Milkyway library, is described in the next section, “[Creating a New Milkyway Library](#).” The next step, importing the physical data, is described in [Chapter 3, “Library Preparation Using GDSII and OASIS”](#) and [Chapter 4, “Library Preparation Using LEF/DEF.”](#) The remaining steps are described in [Chapter 5, “Library Cell Preparation.”](#)

If you have a Milkyway Environment script to perform a library preparation function, you can run the script directly from the IC Compiler tool. For example, to run the Milkyway Environment script named `my_mw_run.tcl` from the IC Compiler prompt:

```
icc_shell> prepare_mw_lib -tcl_file my_mw_run.tcl
```

The `prepare_mw_lib` command has options to specify the working directory, log file name, and the Milkyway Environment executable. By default, the Milkyway Environment tool runs in the background, but you can optionally display the Milkyway Environment tool GUI while the script is being executed.

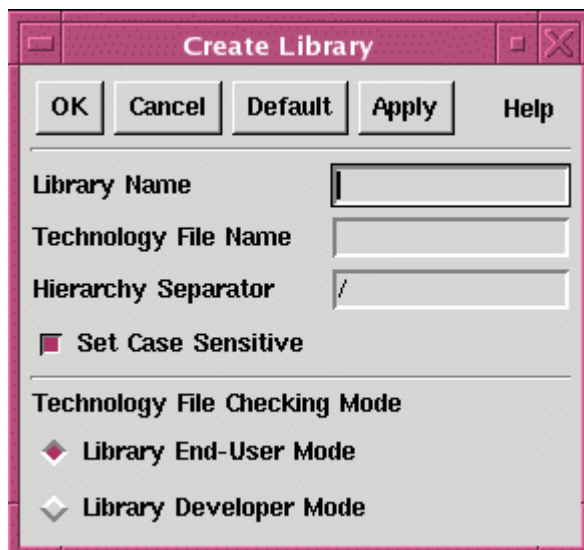
---

## Creating a New Milkyway Library

The first step in Milkyway library preparation is to create a new library. You must have a technology (.tf) file to specify the basic parameters for the library, and the cells you later add to the library must be consistent with that technology file.

To create a new Milkyway library using the Milkyway Environment GUI, choose **Library > Create**. This opens the Create Library dialog box. See [Figure 2-12](#).

Figure 2-12 Library > Create Dialog Box



Enter the new library name, the name of the associated technology (.tf) file, and the hierarchy separator character. Then click OK to create the library.

The library name must start with an alphabetic character. The remaining characters can be alphabetic, numeric, or the underscore character.

The hierarchy separator character must be one of the following: / . | # @ ^

The Technology File Checking Mode options specify the level of detail provided in error messages during library development. Select “Library Developer Mode” to get more detailed messages if you are the library developer. Otherwise, keep the “Library End-User Mode” setting.

You can also create a new library by using the `create_mw_lib` command at the Milkyway> Tcl prompt or the `icc_shell>` prompt in the IC Compiler tool. The `create_mw_lib` command options let you specify the same options as the Create Library dialog box, plus additional options to specify the reference libraries and bus naming style. For details, see the man page.

---

## Extended Milkyway Layers

By default, the Milkyway database supports up to 255 layers, numbered 1 through 255. Layers 1 through 187 are available as user-defined layers and routing layers, while layers 188 through 255 are reserved for the Milkyway system layers.

User-defined layers are layers defined in the technology (.tf) file, such as metal layers and text layers. Among these, the routing layers are the ones used by the IC Compiler router to make interconnections, including wire routes, vias, and contacts.

Milkyway system layers are the layers used to define blockages, route guides, and other physical features that guide the router. The Milkyway system layer numbers are fixed and cannot be assigned or changed by the user.

Some advanced process and routing technologies require more than 255 layers. In these cases, you can extend the number of layers supported in a Milkyway library by executing the `extend_mw_layers` command. In that case, layers 1 through 4000 are available as user-defined layers (including routing layers) and layers 4001 through 4095 are reserved for the Milkyway system layers.

To use extended Milkyway layers in a new Milkyway library, define the layer numbers appropriately in the technology (.tf) file, and execute the `extend_mw_layers` command before you create the Milkyway library:

```
Milkyway> extend_mw_layers
1
Milkyway> create_mw_lib -technology mytech.tf my_mw_lib
Start to load technology file mytech.tf
...
```

After the Milkyway library is created in extended layer mode, it cannot be changed to use the default layer mode.

The reserved Milkyway layers are layer numbers 188 through 255 in the default mode or 4001 through 4095 in extended layer mode. Most layers definitions in the default mode and extended layer mode differ by 3813. For example, layer 188 in default mode is 4001 ( $188+3813$ ) in the extended layer mode. The cell boundary layer is the only exception; it is the highest reserved layer, which is 255 in the default mode or 4095 in extended mode.

The IC Compiler tool automatically recognizes the layer mode of the Milkyway library and uses the correct Milkyway system layer numbers, from 188 to 255 in the default mode or from 4001 to 4095 in the extended layer mode.

Each layer can have multiple data types. Each data type is an integer from 0 to 4095 in both the default and extended layer modes.

In a hierarchical design, the design library can use the extended layer mode while the related reference libraries use the default layer mode. However, if any reference library uses the extended layer mode, the main design library must also use the extended layer mode.





# 3

## Library Preparation Using GDSII and OASIS

---

To prepare a Milkyway reference library from physical cell data, you import the data as a GDSII or OASIS data stream into the Milkyway Environment tool and translate that data into the Milkyway database format. Importing GDSII and OASIS data streams into the Milkyway database and exporting the Milkyway database to GDSII and OASIS output streams are described in the following sections:

- [GDSII to Milkyway](#)
- [Milkyway to GDSII](#)
- [OASIS to Milkyway](#)
- [Milkyway to OASIS](#)
- [Double-Patterning Cell-Level Mask Swapping](#)

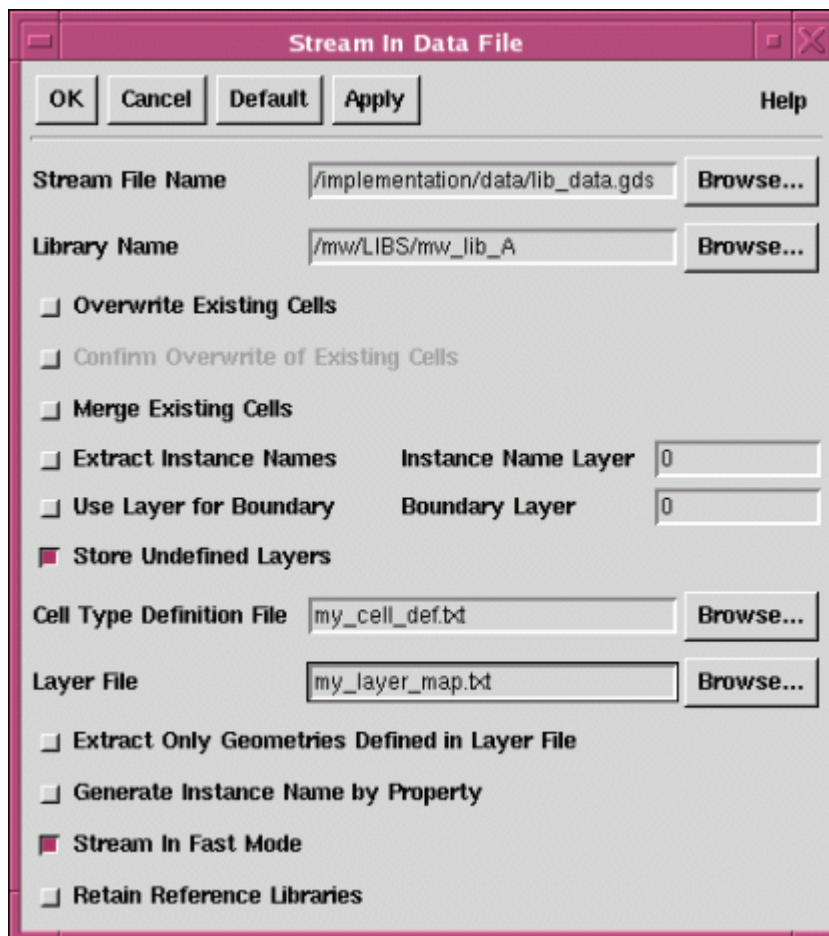
## GDSII to Milkyway

The GDSII stream format is an industry-standard data exchange format for integrated circuit layout information. A GDSII file contains information about layers, wire paths, boundaries, structures, arrays, and text labels in the cell or chip layout.

When an outside source provides library cell information in GDSII format, you need to read the data file into the Milkyway Environment tool and convert the data stream into a cell library in the Milkyway database. In addition to the GDSII file, you typically provide a cell-type definition file and a layer mapping file to specify how to translate the data.

You can read a GDSII file into the Milkyway Environment tool by using either the GUI or the `read_gds` Tcl command. To use the GUI, choose Cell Library > Stream In. This opens the Stream In Data File dialog box, which shows the options for reading GDSII data and converting the data to the Milkyway database format, as shown in [Figure 3-1](#).

Figure 3-1 Cell Library > Stream In Dialog Box



You must at least specify the Stream File Name, which is the name of GDSII file being read in, and the Library Name, which is the name of the Milkyway reference library in which to store the translated data. You might also want to provide the following additional information:

- A cell-type definition file, which identifies the cell types: standard cell, pad cell, filler cell, macro, and so on. The cell function affects the translation of physical data to the Milkyway format. If this file is not provided, all cells are assumed to be standard cells. For information about the file format, see the next subsection, [“Cell-Type Definition File.”](#)
- A layer mapping file, which identifies the corresponding layer names in the GDSII and Milkyway database formats. If this file is not provided, the layer names defined in the GDSII data stream are retained in the Milkyway database. For information about the file format, see [“Layer Mapping File: GDSII or OASIS to Milkyway” on page 3-5.](#)

Some of the dialog box options specify how to handle cells that already exist in the Milkyway library. Others determine how to handle cell boundary layers and layers not defined in the layer mapping file. For a detailed explanation of each option, see the Milkyway Environment online help for the Scheme command `auStreamIn`.

After you set the dialog box options, click OK. The Milkyway Environment tool reads the GDSII data stream, creates a CEL view for each cell, and stores the cells into the specified Milkyway library.

Instead of using the GUI, you can use the equivalent Tcl command in the Milkyway Environment tool, `read_gds`. The Tcl command offers a few options that are not available in the GUI dialog box. These options are related to the processing of text in the GDSII data stream. For details, see the man page for the `read_gds` command, available on SolvNet and in the Milkyway Environment tool.

In the IC Compiler tool, you can read in GDSII data with the `read_stream` command. For details, see [“Reading and Writing GDSII and OASIS in the IC Compiler Tool” on page 7-16.](#)

---

## Cell-Type Definition File

A GDSII or OASIS stream file contains physical information about library cells, but it does not specify the cell types. The Milkyway database needs the cell type information so that it can properly process the cells for use by the IC Compiler tool.

You specify the cell types in a text file. The file contains a list of cell types and the names of cells in the GDSII or OASIS data stream matching that type. For example, consider the following simple cell-type definition file:

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

This file identifies the cells BLOCK1 and BLOCK2 as macro cells, the cells TEST, VIA1, RCAP, and FEED2 and “other” type cells, all other cells as standard cells. Later in the library data preparation flow, each type of cell receives a different type of processing. Cells in the “other” category are translated to Milkyway format but not processed any further.

Any cells in the GDSII or OASIS data stream that are not listed in the cell-type definition file are considered standard cells. In the absence of a cell-type definition file, all cells are considered standard cells.

In general, the cell-type definition file contains statements in the following format:

```
cellType cellNameList
cellType cellNameList
cellType cellNameList
...
```

where *cellType* is one of the types listed in [Table 3-1](#) and *cellNameList* is a list of cell names separated by space characters. An asterisk character represents all cells not listed elsewhere in the file.

**Table 3-1** Cell-Type String in Cell-Type Definition File

Cell-type string	Description of cell type
<code>gdsStandardCell</code>	Standard cells
<code>gdsMacroCell</code>	Macro cells
<code>gdsCornerCell</code>	Corner cells
<code>gdsPadCell</code> or <code>gdsIOCell</code>	Pad cells
<code>gdsStdFillerCell</code>	Standard filler cell
<code>gdsPadFillerCell</code>	Pad filler cells
<code>gdsFCDriverCell</code>	Flip-chip drivers
<code>gdsFCPadCell</code>	Flip-chip pads (bumps)
<code>gdsTapCell</code> or <code>tapCell</code>	Tap cells, which are physical-only cells that have only power and ground pins and no signal pins
<code>gdsOtherCell</code>	Other cells such as feedthrough cells, vias, transistors, and top-level cells, which are not processed further

You can add a comment by inserting a semicolon. Text is ignored from the semicolon to the end of the line.

In the Milkyway Environment GUI, you can define or redefine a cell type for an existing cell by choosing Cell Library > Mark Cell Type.

---

## Layer Mapping File: GDSII or OASIS to Milkyway

Each geometric object in a layout database has an associated layer number and data type number. In the GDSII and OASIS formats, the layer number and data type are integers ranging from 0 to 32767. The Milkyway database supports narrower ranges of numbers:

- Default layer mode: layers 1 through 255, data types 0 through 4095
- Extended layer mode: layers 1 through 4000, data types 0 through 4095

By default, lower layer numbers already supported by the Milkyway database are left unchanged during a translation of physical data from GDSII or OASIS to Milkyway format. Any GDSII and OASIS layer or data type numbers not supported in the Milkyway database must be translated to supported numbers when they are read into the Milkyway database.

The Milkyway database in the default layer mode supports routing layers up to 187 only. If the GDSII or OASIS database uses any higher routing layer numbers, those layers must be mapped to numbers no higher than 187. This limit does not apply to the Milkyway extended layer mode.

To have the layer numbers or data types changed by the stream-in process, specify the desired layer-to-layer mapping in a file and invoke that file in the Cell Library > Stream In dialog box, in the `read_gds` or `read_oasis` command in the Milkyway Environment tool, or in the `set_read_stream_options` command in the IC Compiler tool.

This is the format of the layer mapping file:

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

A layer mapping file in this format can be used for both stream-in and stream-out operations.

The *MilkywayLayer* can be a layer name or number.

The following parts of the layer mapping syntax are for stream-out purposes only, so they are ignored during stream-in:

```
MilkywayObjType  
MilkywayNetType  
PinCode
```

For example,

```
A 12:3 35: 4
```

During stream-in, the tool ignores the character **A** and converts the GDSII data in layer number 35, type 4 to Milkyway layer number 12, datatype 3.

For backward compatibility, the following mapping file syntax is also supported, for stream-in purposes only (not stream-out). Each line in the layer mapping file shows a Milkyway layer number and the corresponding GDSII or OASIS layer number, optionally together with the Milkyway and GDSII or OASIS data types. This is the general syntax:

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

For 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
```

In each line, the first number is the Milkyway layer number, which should be a layer number defined in the technology file associated with the destination Milkyway library. If the layer number is followed by a colon and another number, the number after the colon is the Milkyway data type number, which must be an integer from 0 to 4095.

The next number is the GDSII or OASIS layer number, which must be an integer from 0 to 32767. If the layer number is followed by a colon and another number, the number after the colon is the GDSII or OASIS data type, which must be an integer from 0 to 32767.

You can add a comment to the file by inserting a semicolon. Text is ignored from the semicolon to the end of the line.

To map a GDSII or OASIS layer to a metal blockage layer in the Milkyway database, use one of the reserved Milkyway layer numbers listed in [Table 3-2](#).

*Table 3-2 Milkyway Layer Numbers and Blockage Usage*

<b>Milkyway layer number (default)</b>	<b>Milkyway layer number (extended)</b>	<b>Blockage layer</b>
212	4025	polyBlockage
216	4029	metal4Blockage
217	4030	via3Blockage
218	4031	metal1Blockage
219	4032	metal2Blockage
220	4033	metal3Blockage
223	4036	polyContBlockage

*Table 3-2 Milkyway Layer Numbers and Blockage Usage (Continued)*

<b>Milkyway layer number (default)</b>	<b>Milkyway layer number (extended)</b>	<b>Blockage layer</b>
224	4037	via1Blockage
225	4038	via2Blockage

To discard all objects belonging to a particular layer in the GDSII or OASIS file, map that layer (and possibly data type) to an undefined layer in the technology file and deselect the Save Undefined Layers option in the Stream In dialog box.

For example, suppose the GDSII layer file has objects in layer 12, some of data type 1 and others of data type 2. You want to discard objects in layer 12, data type 2, but retain objects in layer 12, data type 1. In the technology file, layer 130 is not defined. You add the following line to the layer mapping file:

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

This maps the unwanted objects into the undefined layer. In the Stream In dialog box, you deselect the Store Undefined Layers option. The undefined layer is not stored in the Milkyway database.

To save the stream-in layer mapping file into the Milkyway library, use the following command:

```
Milkyway> set_stream_layer_map_file -format in \
        -map_file map_file_name -lib_name mw_library_name
```

The next time you open the Milkyway library in the Milkyway Environment tool and use the `read_gds` or `read_oasis` command, the tool automatically applies the stored layer mapping file to the stream-in process. Similarly, the next time you open the Milkyway library in the IC Compiler tool and use the `read_stream` command, the tool automatically applies the stored layer mapping file to the GDSII or OASIS stream-in process.

In the IC Compiler tool, you can write out the current stream-out layer mapping file by using a command similar to the following:

```
icc_shell> write_mw_lib_files -stream_layer_map_file out -output myfile
```

To remove the stream-in layer map file from the currently open Milkyway library:

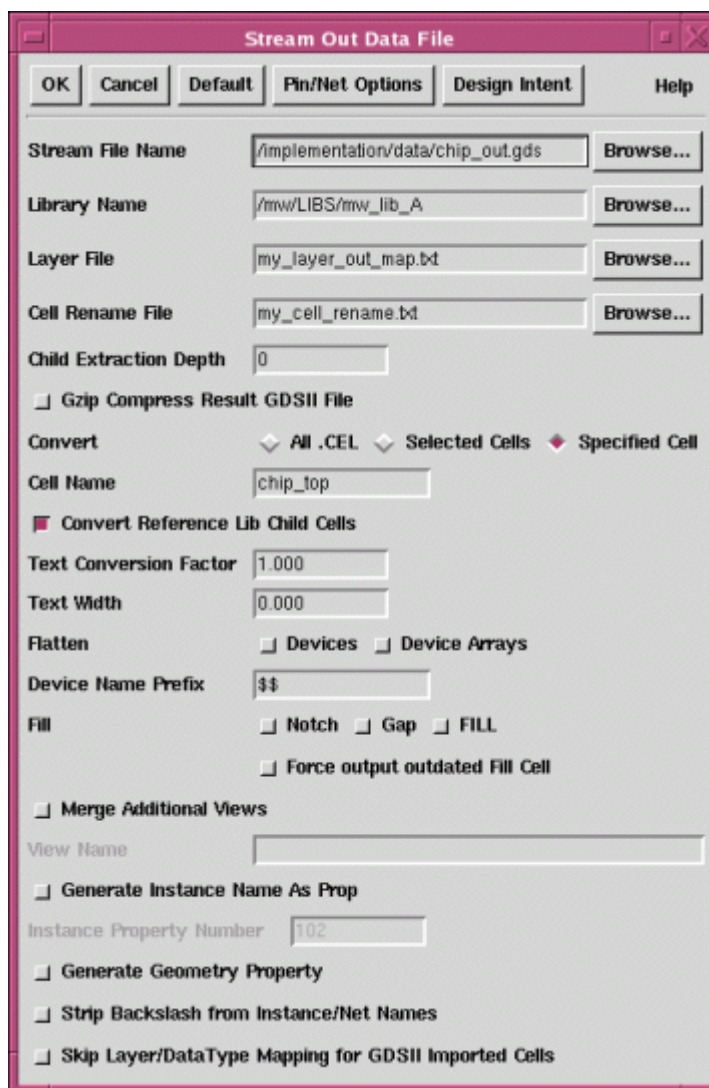
```
Milkyway> set_stream_layer_map_file -format in -remove
```

## Milkyway to GDSII

After completing a place and route or chip assembly design, you can write the completed design back to GDSII Stream format. The GDSII data can then be used for mask creation or for input to external tools.

You can write a GDSII file from the Milkyway Environment tool by using either the GUI or the `write_gds` Tcl command. To use the GUI, choose Output > Stream Out. This opens the Stream Out Data File dialog box, which shows the options for converting the data from the Milkyway database to the GDSII output stream, as shown in [Figure 3-2](#).

Figure 3-2 Output > Stream Out Dialog Box





You must at least specify the Stream File Name, which is the name of the GDSII file being written, and the Library Name, which is the name of the Milkyway library containing the cell data to be written. You can convert all cells, the selected cells, or a specified cell in the Milkyway library.

You might also want to specify a stream-out layer mapping file, which identifies the corresponding layer names in the Milkyway and GDSII formats. If this file is not provided, the layer names defined in the Milkyway database are retained in the written GDSII stream. The file format is different from that of the layer mapping file used for GDSII stream-in conversion. For information about the stream-out file format, see [“Layer Mapping File: Milkyway to GDSII or OASIS” on page 3-10](#).

The dialog box options let you specify many options: which cells in the Milkyway library to write, cell renaming, hierarchical extraction depth, hierarchical flattening, the handling of text objects, notch and gap filling, and several other options. Two buttons at the top, Pin/Net Options and Design Intent, open dialog boxes for setting additional options. For a detailed explanation of each option, see the Milkyway Environment online help for the Scheme command `auStreamOut`.

After you set the dialog box options, click OK. The Milkyway Environment tool translates the specified cells from Milkyway format to GDSII format and writes out the GDSII file.

Instead of using the GUI, you can use the equivalent Tcl command in the Milkyway Environment tool, `write_gds`. For details, see the man page for the `write_gds` command.

In the IC Compiler tool, you can write GDSII data with the `write_stream` command. For details, see [“Reading and Writing GDSII and OASIS in the IC Compiler Tool” on page 7-16](#).

---

## Objects in Milkyway That Can Be Streamed Out

Only physical information can be streamed out from the Milkyway database to the GDSII stream file. [Table 3-3](#) lists the types of Milkyway objects that can be streamed out and the resulting object types in the GDSII output stream.

*Table 3-3 Stream Out Objects in Milkyway*

<b>Milkyway object</b>	<b>GDSII output stream object</b>
CellInstance	SREF (structure reference element)
CellInstArray	AREF (array reference element)
Contact	SREF or boundary element
ContactArray	SREF or boundary element
HorizontalWire	Path

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

Milkyway object	GDSII output stream object
Path	Path
Pin	Boundary element
Polygon	Boundary element
Rectangle	Boundary element
Text	Text
VerticalWire	Path

If you choose the Flatten Devices or Device Arrays option in the Cell Output > Stream Out dialog box, each contact or contact array is converted into several GDSII boundary elements. By default, Milkyway outputs each device or device array as a structure reference element (SREF) with an optional device name prefix if specified in the dialog box.

Because the GDSII format contains only layout information, the Milkyway Environment tool cannot stream out connectivity information or place-and-route objects such as ports, nets, port instances, regions, and cell rows. Extension type objects, such as group, property, and attached file, also cannot be streamed out.

## Layer Mapping File: Milkyway to GDSII or OASIS

For the stream-out conversion process, specify the desired object-to-layer mapping in the layer mapping file. Specify the file name in the Output > Stream Out dialog box, in the `write_gds` or `write_oasis` command in the Milkyway Environment tool, or in the `set_write_stream_options` command in the IC Compiler tool.

Each line in the layer mapping file shows a Milkyway object type, Milkyway layer, and resulting GDSII or OASIS layer in the output stream. This is the general syntax:

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

The first character in the line is the code for the type of Milkyway object to be translated. Use A for all, T for text, or D for data. The optional second character specifies the net type, such as S for signal, P for power, or G for ground. An optional third character specifies the pin code. The Milkyway layer is a name or an integer. The GDSII or OASIS layers and data types are integers.

By default, net text and pin text are mapped into the same layer as the associated net or pin. However, you can specify a different layer for net text or pin text in the layer mapping file.

The following example demonstrates the stream-out layer mapping syntax.

```
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 -         ; minus sign (hyphen) prevents transfer of all
                   ; text and data on Milkyway layer METAL4
```

You can add a comment to the file by inserting a semicolon. Text is ignored from the semicolon to the end of the line. A hyphen character in the GDSII or OASIS layer position prevents the transfer of all text and data in the specified Milkyway object and layer.

A colon is the delimiter between the GDSII or OASIS layer and the optional GDSII or OASIS data type. For backward compatibility, you can use a space character alone, without a colon, or a colon followed by white space, as the delimiter.

To save the stream-out layer mapping file into the Milkyway library, use the following command:

```
Milkyway> set_stream_layer_map_file -format out \
        -map_file map_file_name -lib_name mw_library_name
```

The next time you open the Milkyway library in the Milkyway Environment tool and use the `write_gds` or `write_oasis` command, the tool automatically applies the stored layer mapping file to the stream-out process. Similarly, the next time you open the Milkyway library in the IC Compiler tool and use the `write_stream` command, the tool automatically applies the stored layer mapping file to the GDSII or OASIS stream-in process.

In the IC Compiler tool, you can write out the current stream-in layer mapping file by using a command similar to the following:

```
icc_shell> write_mw_lib_files -stream_layer_map_file in -output myfile
```

To remove the stream-out layer map file from the currently open Milkyway library:

```
Milkyway> set_stream_layer_map_file -format out -remove
```

[Table 3-4](#) describes the items used in each line of the layer mapping file.

**Table 3-4 GDSII Stream-Out Layer Mapping File Syntax**

Variable	Description
<i>MilkywayObjType</i>	The code for the type of object in the data to be translated: A for all types T for text D for data
<i>MilkywayNetType</i>	The code for the net type of the object in the data to be translated: S for signal P for power G for ground C for clock U for Up conduction layer (upper layer in a via). Example: metal2 in an m1/m2 via D for Down conduction layer (lower layer in a via). Example: metal1 in an m1/m2 via X for power and ground wires or contacts with a “signal” or “tie-off” routing type A for all net types

Table 3-4 GDSII Stream-Out Layer Mapping File Syntax (Continued)

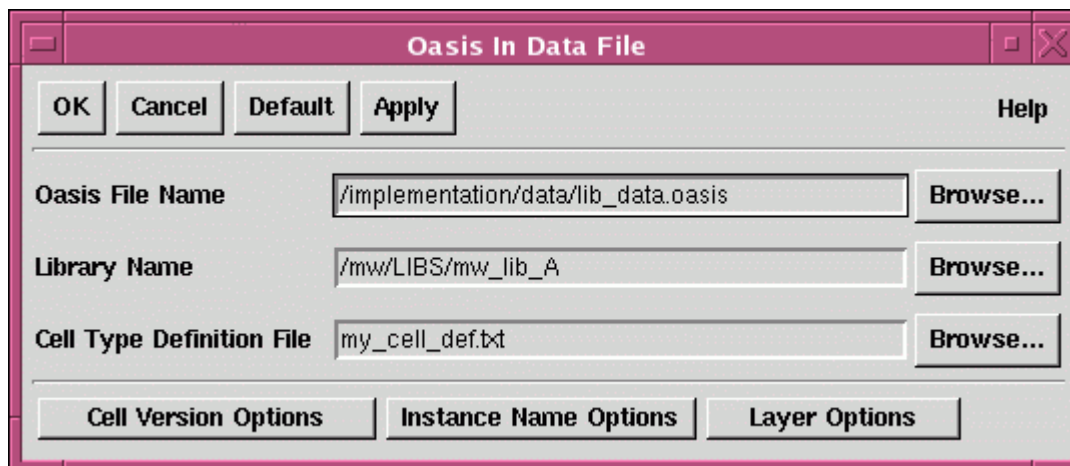
Variable	Description
	<p>Note that U and D are used only for via and flattened via objects. They override any other net type set for the layer when the layer is in a via.</p> <p>If <code>MilkywayNetType</code> is omitted, the tool maps all Milkyway data of the specified object type to the specified GDSII or OASIS layer.</p> <p>If <code>MilkywayNetType</code> is included, the tool examines every object of the object type to determine its net type and maps it to the layer you specify. If an object has no net or a net has no type, the tool assumes that it is a signal net.</p> <p>If a layer file contains contradictory lines, the last line overrides any previous line. For example, if an earlier line of a layer file specifies mapping for a particular object or net type and a later line specifies mapping for the same object type but no net type, the tool translates all data of the specified object type as defined by the later line.</p>
<code>PinCode</code>	<p>Optional. Can be used when only the Pin geometry or Terminal (top-level pin) geometry in the Milkyway database needs to be translated:</p> <p>T for terminal geometries of the current design</p> <p>P for terminal geometries of the current design, as well as the physical pin geometries of the child cells, depending on the child depth option.</p> <p>Note: PinCode P and T only work when <code>MilkywayNetType</code> is set to A. Otherwise, the PinCode rule is ignored.</p>
<code>MilkywayLayer</code>	Specifies the name or number, from 1 to 255 (1 to 4095 in extended layer mode), of a layer to be translated and defined in the technology file.
<code>MilkywayDataType</code>	The data type of the Milkyway data to be translated: an integer from 0 to 4095.
<code>GDSIILayer</code>	Specifies the number of the GDSII or OASIS stream layer to which the objects of <code>MilkywayObjType</code> on <code>MilkywayLayer</code> are translated. Use an integer from 0 to 32767 or use a hyphen character (-) to ignore the Milkyway object and layer during translation.
<code>GDSIIDataType</code>	Specifies the number of the GDSII or OASIS stream data type to which objects of <code>MilkywayObjType</code> and <code>MilkywayNetType</code> , if given, on <code>MilkywayLayer</code> are translated. Use an integer from 0 to 32767.

## OASIS to Milkyway

Like GDSII, OASIS is an industry-standard data exchange format for integrated circuit layout information. OASIS, which stands for Open Artwork System Interchange Standard, is designed as an improved replacement for GDSII that allows integer bit widths to exceed the 16-bit and 32-bit limits imposed by the GDSII standard. It also has several efficiency improvements to support larger designs.

You can read an OASIS file into the Milkyway Environment tool by using either the GUI or the `read_oasis` Tcl command. To use the GUI, set the command entry mode to Scheme and enter the Scheme command `read_oasis`. This opens the Oasis In Data File dialog box, which shows the options for reading OASIS data and converting the data to the Milkyway database format, as shown in [Figure 3-3](#).

Figure 3-3 Scheme `read_oasis` Dialog Box



You must at least specify the Oasis File Name, which is the name of the OASIS file being read in, and the Library Name, which is the name of the Milkyway reference library in which to store the translated data. You might also want to specify the following:

- A cell-type definition file, which identifies the function of each cell: standard cell, pad cell, filler cell, macro, and so on. If this file is not provided, all cells are assumed to be standard cells. The file format is the same as for GDSII input. For information about the file format, see [“Cell-Type Definition File” on page 3-3](#).
- A layer mapping file, which identifies the corresponding layer names in the OASIS and Milkyway database formats. If this file is not provided, the layer names defined in the OASIS data stream are retained in the Milkyway database. The file format is the same as for GDSII input. For information about the file format, see [“Layer Mapping File: GDSII or OASIS to Milkyway” on page 3-5](#).

The buttons labeled Cell Version Options, Instance Name Options, and Layer Options open separate dialog boxes for entering those types of options. For detailed information on the options, see the man page for the `read_oasis` Tcl command in Tcl mode.

After you set the dialog box options, click OK. The Milkyway Environment tool reads the OASIS data stream, creates a CEL view for each cell, and stores the cells into the specified Milkyway library.

Instead of using the GUI, you can use the equivalent Tcl command, `read_oasis`. For details, see the man page for the `read_oasis` Tcl command.

In the IC Compiler tool, you can read in OASIS data with the `read_stream` command. For details, see [“Reading and Writing GDSII and OASIS in the IC Compiler Tool” on page 7-16](#).

---

## Milkyway to OASIS

After completing placement and routing or a chip assembly, you can write the completed design to OASIS format. The OASIS data can then be used for mask creation or for input to external tools.

You can write an OASIS file from the Milkyway Environment tool by using either the GUI or the `write_oasis` Tcl command. To use the GUI, set the command entry mode to Scheme and enter the Scheme command `write_oasis`. This opens the Oasis Out Data File dialog box, which shows the options for converting the data from the Milkyway database to the OASIS output stream, as shown in [Figure 3-4](#).

Figure 3-4 Scheme write\_oasis Dialog Box

**Oasis Out Data File**

OK Cancel Default Help

Oasis File Name  Browse...

Library Name  Browse...

Layer File  Browse...

Child Extraction Depth

Convert ☒ All .CEL ☐ Selected Cells ☐ Specified Cell

Cell Name

☒ Convert Reference Lib Child Cells

☐ Compress Result OASIS File

Compression Level [1-9]

Net/Pin Options Fill Options Contact Options Special Options

You must at least specify the Oasis File Name, which is the name of file being written, and the Library Name, which is the name of the Milkyway library containing the cell data to be written.

You might also want to specify a stream-out layer mapping file, which identifies the corresponding layer names in the Milkyway and OASIS database formats. If this file is not provided, the layer names defined in the Milkyway database are retained in the written OASIS stream. The file format is the same as that used for GDSII stream-out conversion. For information about the stream-out file format, see [“Layer Mapping File: Milkyway to GDSII or OASIS” on page 3-10](#).

The dialog box options let you specify several options, including the cells in the Milkyway library to write, the hierarchical extraction depth, and the output file compression. Four buttons at the bottom, Net/Pin Options, Fill Options, Contact Options, and Special Options, open dialog boxes for setting additional options. For a detailed explanation of each option, see the man page for the Tcl command `write_oasis` in Tcl mode.

After you set the dialog box options, click OK. The Milkyway Environment tool translates the specified cells from Milkyway format to OASIS format and writes out the OASIS file.



Instead of using the GUI, you can use the equivalent Tcl command, `write_oasis`. For details, see the man page for the `write_oasis` command.

In the IC Compiler tool, you can write OASIS data with the `write_stream` command. For details, see [“Reading and Writing GDSII and OASIS in the IC Compiler Tool” on page 7-16](#).

The types of Milkyway objects that can be streamed out are the same as for exporting data in GDSII format. For details, see [“Objects in Milkyway That Can Be Streamed Out” on page 3-9](#).

---

## Double-Patterning Cell-Level Mask Swapping

During stream-out of design data to GDSII or OASIS format in a double-patterning technology, the tool uses an advanced method of correcting color-related spacing violations between geometries in adjacent cells. The tool writes out a duplicate cell with swapped color assignments and uses that cell where necessary to avoid spacing violations. The `place_opt` command in the IC Compiler tool generates the swap information and stores it in a cell attribute called `mask_shift`.

For details, see “Cell-Level Mask Swapping” in the *IC Compiler Advanced Geometries User Guide*.



# 4

## Library Preparation Using LEF/DEF

---

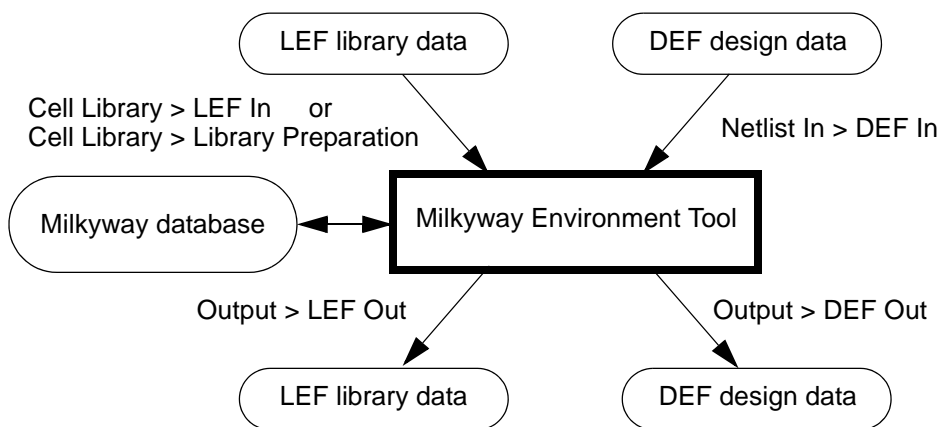
To prepare a Milkyway reference library from cell data in LEF format, you import the data into the Milkyway Environment tool and translate that data into the Milkyway database. You can also import and export design data in DEF format. Importing and exporting LEF and DEF data are described in the following sections:

- [LEF and DEF Data](#)
- [LEF to Milkyway](#)
- [Milkyway to LEF](#)
- [DEF Input and Output](#)

## LEF and DEF Data

The Library Exchange Format (LEF) and Design Exchange Format (DEF) are two related data exchange formats used in the semiconductor industry. A full description of a chip design using these formats includes the design-specific information in DEF format and the library cell information and technology information in LEF format. You can convert both library data and design data between the Milkyway database and the LEF/DEF formats, as summarized in [Figure 4-1](#).

*Figure 4-1 LEF and DEF Data Conversion Using the Milkyway Environment Tool*



The LEF format defines the elements of a process technology and the associated library of cell models, including layer, via, placement site type, and macro cell definitions. The Milkyway Environment tool supports the conversion of library data from LEF to Milkyway format and from Milkyway format to LEF. All versions of LEF through version 5.7 are supported.

The DEF format defines the elements of a specific design that are related to physical layout, including the placement and routing information, design netlist, and design constraints. The Milkyway Environment tool supports the conversion of design data from DEF to Milkyway format and from Milkyway format to DEF. For information about the supported syntax, see [Appendix B, “DEF 5.6 and 5.7 Supported Syntax.”](#) For additional information about the supported LEF version 5.7 syntax, see SolvNet article 029602, “LEF 5.7 Syntax Support in C-2009.06.”

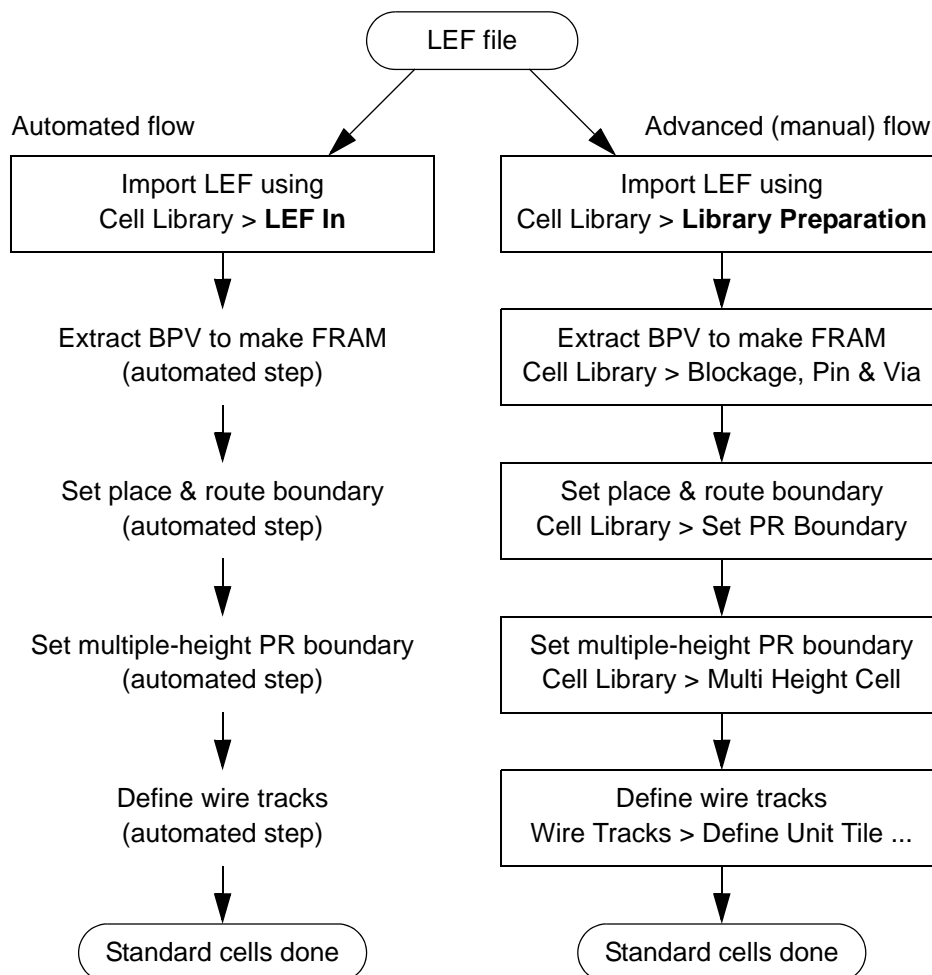
---

## LEF to Milkyway

When an outside source provides library cell information in LEF format, you need to read the data file into the Milkyway Environment tool and convert the data into a cell library in the Milkyway database. The LEF format contains more than just the physical information contained in a GDSII or OASIS data stream, so many of the cell processing steps that must be done manually for a GDSII or OASIS library flow can be done automatically with LEF data.

[Figure 4-2](#) shows the overall flow for creating a Milkyway reference library from LEF data. You can choose to use an automated flow in which the Milkyway Environment tool determines the cell boundary and wire tracks from the DEF data. Alternatively, you can choose an advanced flow in which you manually specify the cell boundary and wire tracks.

Figure 4-2 Library Preparation Flows Using LEF



For the automated flow, choose Cell Library > LEF In. For the advanced flow, choose Cell Library > Library Preparation.

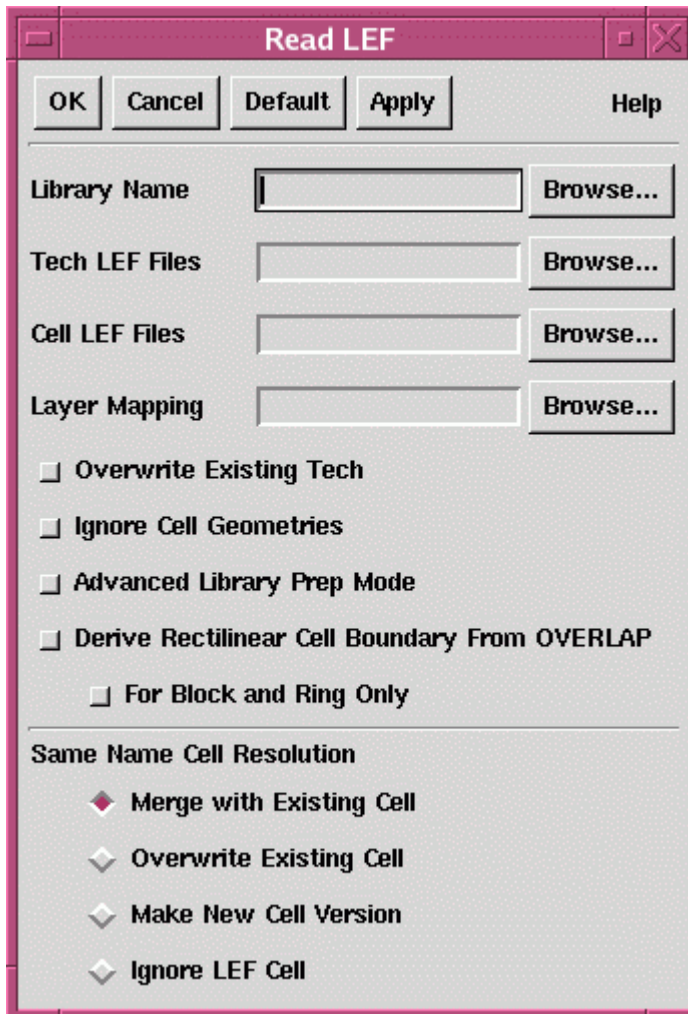
In addition to the flows shown in [Figure 4-2](#), you can choose a mixed flow that uses some combination of library and technology information in LEF, GDSII, CLF, PLIB, and TF formats. For this flow, you choose Cell Library > Library Preparation.

---

## Automated LEF Input Flow

You can import LEF data in the automated mode by using either the GUI or the `read_lef` Tcl command. To use the GUI, choose Cell Library > LEF In. This opens the Read LEF dialog box, as shown in [Figure 4-3](#). The dialog box displays the options for reading DEF data and converting the data to the Milkyway database format.

Figure 4-3 Cell Library > LEF In Dialog Box



Specify the name of the Milkyway library in which to store the converted data, the names of the LEF technology files containing the technology information, and the names of the LEF files containing the physical cell data.

If the specified Milkyway library does not already exist, the tool creates a new Milkyway reference library based on the LEF information. If the Milkyway library already exists, the tool performs an incremental update, 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.

LEF files can contain technology information, physical cell information, or both. The information you need might be contained in a single LEF file or in multiple LEF files. Enter the name or names of the files in the Tech LEF Files and Cell LEF Files text boxes. Multiple file names in a text box must be separated by space characters.

From the files listed in the Tech LEF Files text box, only the technology information is used, not the cell-level information. Similarly, from the files listed in the Cell LEF Files text box, only the cell-level information is used, not the technology information. If a file contains both technology information and cell information that you need, be sure to enter that file name into both text boxes.

You might also want to specify a layer mapping file, which identifies the corresponding layer names in the DEF and Milkyway database formats. If this file is not provided, the first layer in DEF is mapped to layer 1 in Milkyway, the second layer is mapped to layer 2, and so on. For information about the file format, see [“LEF to Milkyway Layer Mapping File” on page 4-9](#).

The dialog box also offers the following options:

- **Overwrite Existing Tech** – Use this option to overwrite, rather than add to, the existing technology information in the Milkyway library.
- **Ignore Cell Geometries** – Use this option to ignore physical cell information, for example, when you want to import only the technology information in DEF format and you plan to get the cell information in GDSII format.
- **Advanced Library Prep Mode** – Use this option to invoke the advanced (manual) flow after the DEF files have been imported. For information about this flow, see [“Advanced LEF Input Flow” on page 4-7](#).
- **Derive Rectilinear Cell Boundary From OVERLAP** – Derive a rectilinear (not rectangular) cell boundary from the `OVERLAP` layer in the LEF file. Select the `For Block and Ring Only` option to derive rectilinear cell boundaries only from `BLOCK` and `RING` type macro cells in the LEF file.
- **Same Name Cell Resolution** – Specify how to handle the an imported cell that has the same name as an existing cell in the target Milkyway library.

After you set the dialog box options, click OK. The Milkyway Environment tool reads the DEF, converts the technology information into Milkyway format, creates a CEL view for each cell, and stores the cells into the specified Milkyway library. It also determines the standard-cell place-and-route boundaries, multiple-height place-and-route boundaries, and



wire tracks of the cells. If the automated flow does not produce the desired results, start again and choose the advanced flow.

Instead of using the GUI for the automated flow, you can use the equivalent Tcl command, `read_lef`. This is the syntax of the Tcl command:

```
read_lef
[-lib_name mw_lib_name]
[-tech_lef_files tech_lef_files]
[-cell_lef_files cell_lef_files]
[-layer_mapping layer_mapping_file_name]
[-overwrite_existing_tech ]
[-ignore_cell_geom]
[-advanced_lib_prep_mode]
[-cell_version merge | overwrite | new | ignore]
```

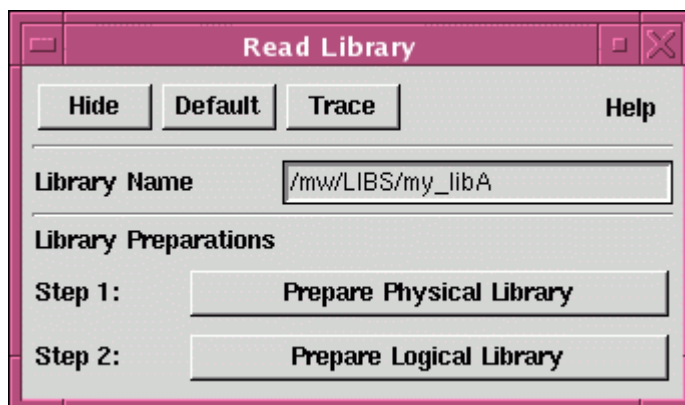
For details, see the man page for the `read_lef` command, available on SolvNet and in the Milkyway Environment tool.

---

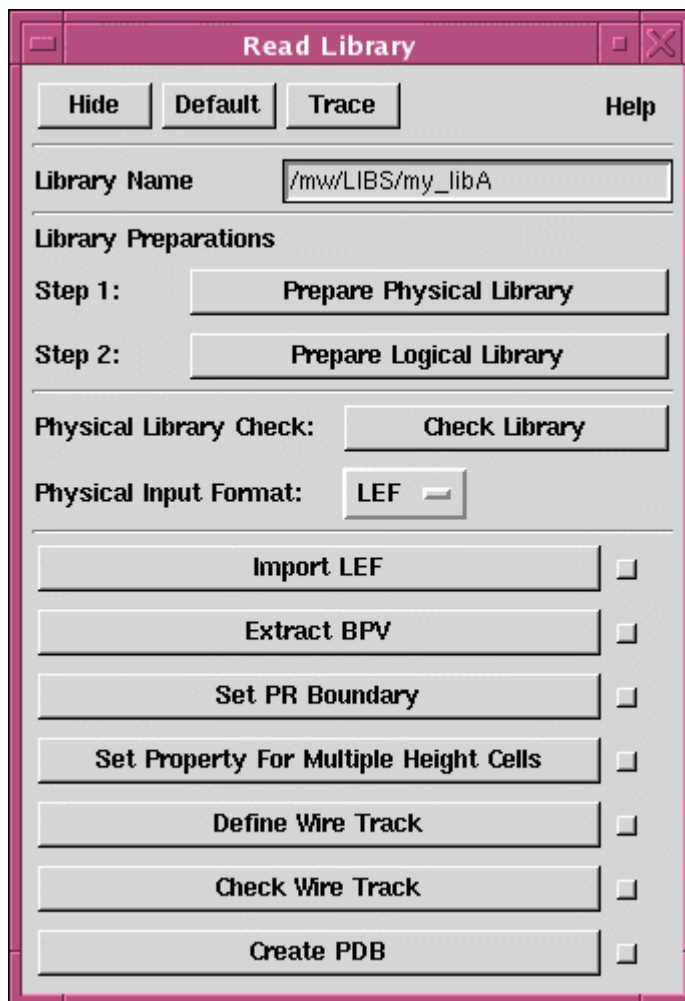
## Advanced LEF Input Flow

To import LEF data in the advanced (manual) mode, choose Cell Library > Library Preparation. This opens the Read Library dialog box, as shown in [Figure 4-4](#). The dialog box initially contains two library preparation buttons, Prepare Physical Library and Prepare Logical Library. For the IC Compiler tool, you only use the Prepare Physical Library button. The other button supports the generation of LM views used by older tools such as Astro.

*Figure 4-4 Initial Cell Library > Library Preparation Dialog Box*



Specify the name of the Milkyway library in which to store the converted data. Then click the Prepare Physical Library button. This expands the dialog box to show a list of steps to be performed in the flow, as shown in [Figure 4-5](#).

*Figure 4-5 Expanded Cell Library > Library Preparation Dialog Box*

The Physical Library Check: Check Library button opens a separate dialog box that lets you perform a check of a physical library in Milkyway format. Library checking is described in [Chapter 6, “Library Checking.”](#)

The Physical Input Format option button specifies the type or types of data files you are importing into the Milkyway format. If you are importing only LEF files, make sure the button is set to LEF as shown in the foregoing figure.

With the Physical Input Format button set to LEF, the list of steps shown below the button reflect the flow for importing data from LEF files. To perform the steps in the flow, click the buttons one at a time, starting from the first step, Import LEF. As each step is completed, a check mark appears in the small box to the right of the corresponding button.

These are the steps in the LEF-only input flow:

1. Import LEF – Opens the Read LEF dialog box described in the previous section, [“Automated LEF Input Flow” on page 4-5](#).
2. Extract BPV – Opens the Extract Blockage dialog box. See [“Extract Blockage Options” on page 5-7](#).
3. Set PR Boundary – Opens the Set PR Boundary dialog box. See [“Set the Place-and-Route Boundary” on page 5-22](#).
4. Set Property for Multiple Height Cells – Opens the Set Multiple Height PR Boundary dialog box. See [“Set the Multiple-Height Boundary” on page 5-25](#).
5. Define Wire Track – Opens the Define Wire Track dialog box. See [“Define the Unit Tile Wire Tracks” on page 5-27](#).
6. Check Wire Track – Opens the Check Wire Track dialog box. See [“Check the Wire Tracks” on page 5-30](#).
7. Create PDB – Skip this step for Milkyway library usage in the IC Compiler tool. If run, this step creates an LM view, which is used only in older tools such as Astro.

---

## LEF to Milkyway Layer Mapping File

A layer mapping file specifies the Milkyway layer resulting from each LEF layer. If no layer mapping file is used, 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 a Perl script to create a layer mapping file. The script is available in the SolvNet article [“Using the lef\\_layer\\_tf\\_number\\_mapper.pl script.”](#) This is the usage 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 .tf technology file and the LEF file, respectively, run this script at the operating system prompt:

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

This generates a map file called test\_y\_gold\_x.map and a log file called test\_y\_gold\_x.log.

The following is an example of a layer mapping file:

LEF layer name	Milkyway Layer number
METAL1	10
VIA12	11
METAL2	20
VIA23	21
METAL3	30

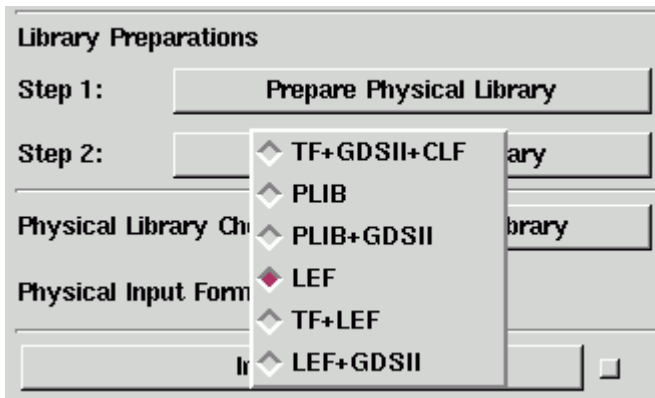
VIA34	31
METAL4	40

Providing a complete mapping of all LEF layers to the desired Milkyway layer numbers is recommended. Without this mapping, the tool assigns default layer numbers.

## Mixed Input Flows Using LEF, GDSII, TF, CLF, and PLIB

For a mixed library input flow that uses some combination of library and technology information in LEF, GDSII, TF, CLF, and PLIB formats, choose Cell Library > Library Preparation. In the Read Library dialog box, click Prepare Physical Library to expand the dialog box. Then set the Physical Input Format button to the combination of library input file types by selecting from the options shown in [Figure 4-6](#).

Figure 4-6 Mixed Library Input File Type Options



The list of steps shown below the button changes to reflect the flow requirements of the selected combination of input file types. To perform the steps in the flow, click the buttons one at a time, starting from the first step at the top of the list. As each step is completed, a check mark appears in the small box to the right of the corresponding button.

For example, if you have the technology information in a LEF file and cell physical data in a GDSII file, follow these steps:

1. Choose Cell Library > Library Preparation to open the Read Library dialog box.
2. Click Prepare Physical Library.
3. Set the Physical Input Format button to LEF + GDSII.
4. Click the buttons one at a time, from top to bottom, to perform the steps in the flow.

If you need information about using a dialog box, click the Help button in the dialog box. This opens an online help window for that dialog box.

## Milkyway to LEF

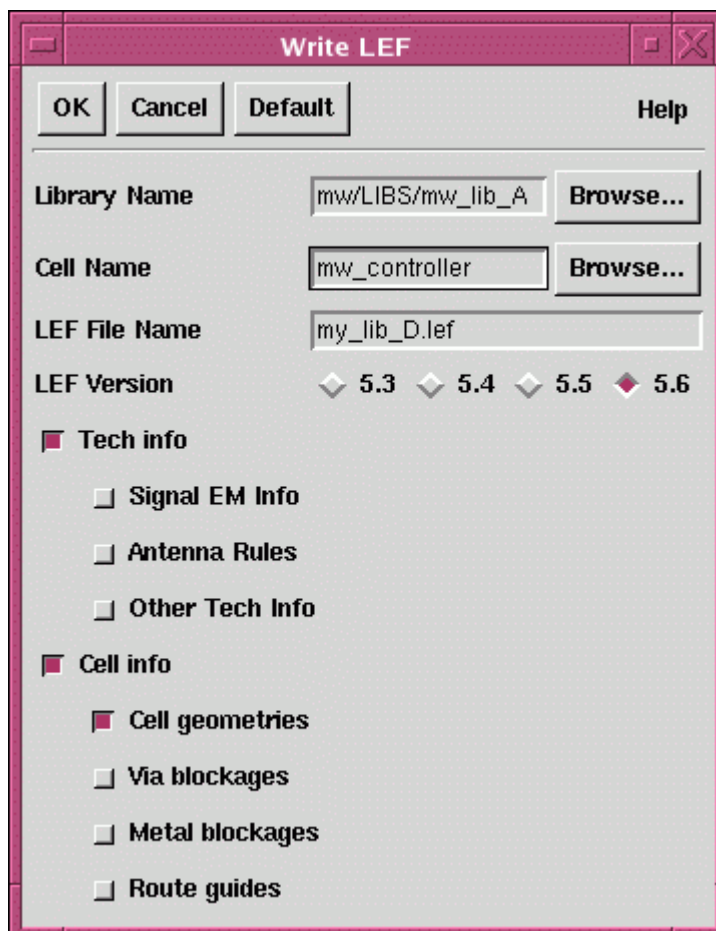
You can export a Milkyway library to LEF format for use by external tools. The LEF format defines the elements of an IC process technology and associated library of cell models and contains library information for a class of designs. This library information includes layer, via, placement site type, and macro cell definitions.

You can write a LEF file from the Milkyway Environment tool by using either the GUI or the `write_lef` Tcl command.

To use the GUI,

1. Choose Output > LEF Out. This opens the Write LEF dialog box, which shows the options for converting the data from the Milkyway database to LEF format, as shown in [Figure 4-7](#).

Figure 4-7 Output > LEF Out Dialog Box



2. You must at least specify the name of the Milkyway library to be converted and the name of the LEF file to be written. You can specify the name of a cell in the library to be converted or browse to a specific cell. If you do not specify a cell name, all the cells in the Milkyway library are written to the LEF file. If the LEF file already exists, it is overwritten.
3. Use the other dialog box options to specify the LEF version number and the types of technology and cell information to be written. For detailed information about these options, click the Help button in the dialog box or see the man page for the Tcl command `write_lef` in Tcl mode.
4. After you set the dialog box options, click OK. The Milkyway Environment tool translates the specified cells from Milkyway format to LEF format and writes out the LEF file.

Instead of using the GUI, you can use the equivalent Tcl command, `write_lef`. The Tcl command supports LEF version 5.7, which the Output > LEF Out dialog box does not support. This is the Tcl command syntax:

```
write_lef
[-lib_name lib_name]
[-output_cell cell_name]
[-output_version 5.3 | 5.4 | 5.5 | 5.6 | 5.7]
[-ignore_tech_info]
[-ignore_tech_signal_em]
[-ignore_tech_antenna_rule]
[-ignore_tech_other_info]
[-ignore_cell_info]
[-ignore_cell_geom]
[-output_via_blockages]
[-output_metal_blockages]
[-output_route_guides]
lef_file_name
```

For details, see the man page for the `write_lef` command.

---

## DEF Input and Output

The Design Exchange Format (DEF) defines the elements of an IC design that are relevant to the physical layout, including the placement and routing information, design 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.

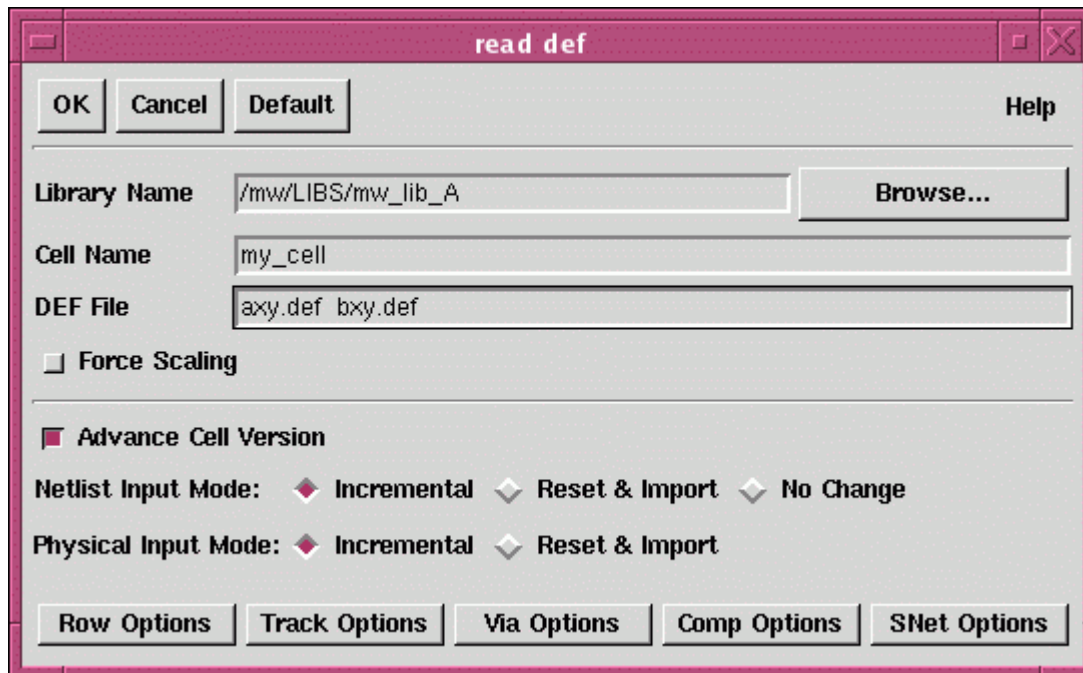
You can read DEF files into a Milkyway library and write DEF files from a Milkyway library. DEF versions 5.3 through 5.7 are supported.

## Importing DEF Files

The DEF format defines the elements of a specific design that are related to physical layout, including the placement and routing information, design netlist, and design constraints.

You can read a DEF file into the Milkyway Environment tool by using either the GUI or the `read_def` Tcl command. To use the GUI, choose Netlist In > DEF In. This opens the Read DEF dialog box, which shows the options for reading DEF data and converting the data to the Milkyway database format, as shown in [Figure 4-8](#).

Figure 4-8 Netlist In > DEF In Dialog Box



You must at least specify the name of the Milkyway library in which to store the converted data, the name of the cell to convert, and the name of the DEF file or files to read.

If you specify multiple DEF files, separate their names with space characters. The order in which you enter file names is important because DEF syntax defined in previous sections is sometimes used in later sections. For example, suppose your first DEF file (axy.def) includes syntax for the REGION section, and your second DEF file (bxy.def) includes syntax for the GROUP section. If the second file includes [+ REGION regionName] syntax in the GROUP section, the correct file order is axy.def first and bxy.def second.

The tool can read DEF files that have been compressed with gzip and named with the extension .gz, such as axy.def.gz or bxy.gz.

Some of the dialog box options specify how to handle cells that already exist in the Milkyway library. The buttons along the bottom expand the dialog box, allowing you to set additional options related to rows, tracks, vias, components, and special nets. For detailed information about these options, click the Help button in the dialog box or see the man page for the Tcl command `read_def` in Tcl mode.

After you set the dialog box options, click OK. The Milkyway Environment tool reads the DEF files, converts the data into a CEL view, and stores the data into the specified Milkyway library. The conversion process checks the syntax used in the DEF file. Any issues found are reported with information messages, warnings, and error messages, depending on the severity of the violation.

Instead of using the GUI, you can use the equivalent Tcl command, `read_def`. The Tcl command offers a few options that are not available in the GUI dialog box. These options are related to the processing of text in the GDSII data stream. This is the command syntax:

```
read_def
[-lib library_name]
[-design design_name]
[-enforce_scaling]
[-allow_physical]
[-no_incremental]
[-advance_cell_version]
[-netl_phys {incr_incr | incr_rimport | rimport_rimport |
             nochange_incr | nochange_rimport}]
[-site site_definition_file_name]
[-verbose]
[-core core_site_name]
[-turn_via_to_inst]
[-inexactly_matched_via_to_inst]
[-skip_signal_nets]
[-lef lef_file_name]
[-blackbox blackbox_ref_file_name]
[-blackbox_size blackbox_size]
[-snet_no_shape_as_user_enter]
[-snet_no_shape_as_detail_route]
def_file_names
```

For details, see the man page for the `read_def` command, available on SolvNet and in the Milkyway Environment tool.

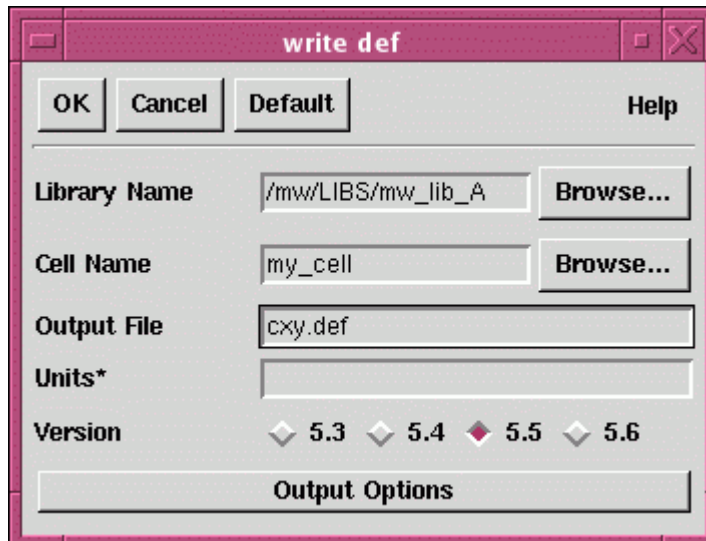
---

## Exporting DEF Files

You can write a DEF file from the Milkyway Environment tool by using either the GUI or the `write_def` Tcl command. To use the GUI, choose Output > DEF Out. This opens the Write DEF dialog box, which shows the options for converting the data from the Milkyway database format and writing the DEF file, as shown in [Figure 4-9](#).



Figure 4-9 Output &gt; DEF Out Dialog Box



You must at least specify the name of the Milkyway library containing the cell data, the name of the cell to convert, and the name of the DEF file to write.

The Units text box lets you specify the distance unit for the output DEF, which can be 100, 200, 1000, 2000, 10000, or 20000. The default is the length precision value of the technology file.

Clicking the Output Options button at the bottom expands the dialog box, allowing you to set additional options regarding the types of Milkyway database objects converted and how to convert them. For detailed information about these options, click the Help button in the dialog box or see the man page for the Tcl command `write_def` in Tcl mode.

After you set the dialog box options, click OK. The Milkyway Environment tool converts the specified cell information to DEF format and writes the specified DEF file. If the DEF file already exists, it is overwritten.

Instead of using the GUI, you can use the equivalent Tcl command, `write_def`. The Tcl command supports DEF version 5.7, which the Output > DEF Out dialog box does not support. This is the command syntax:

```
write_def
-output output_file_name
[-lib library_name]
[-topdesign design_name]
[-version 5.3 | 5.4 | 5.5 | 5.6 | 5.7]
[-unit conversion_factor]
[-compressed]
[-rows_tracks_gcells]
[-vias]
```

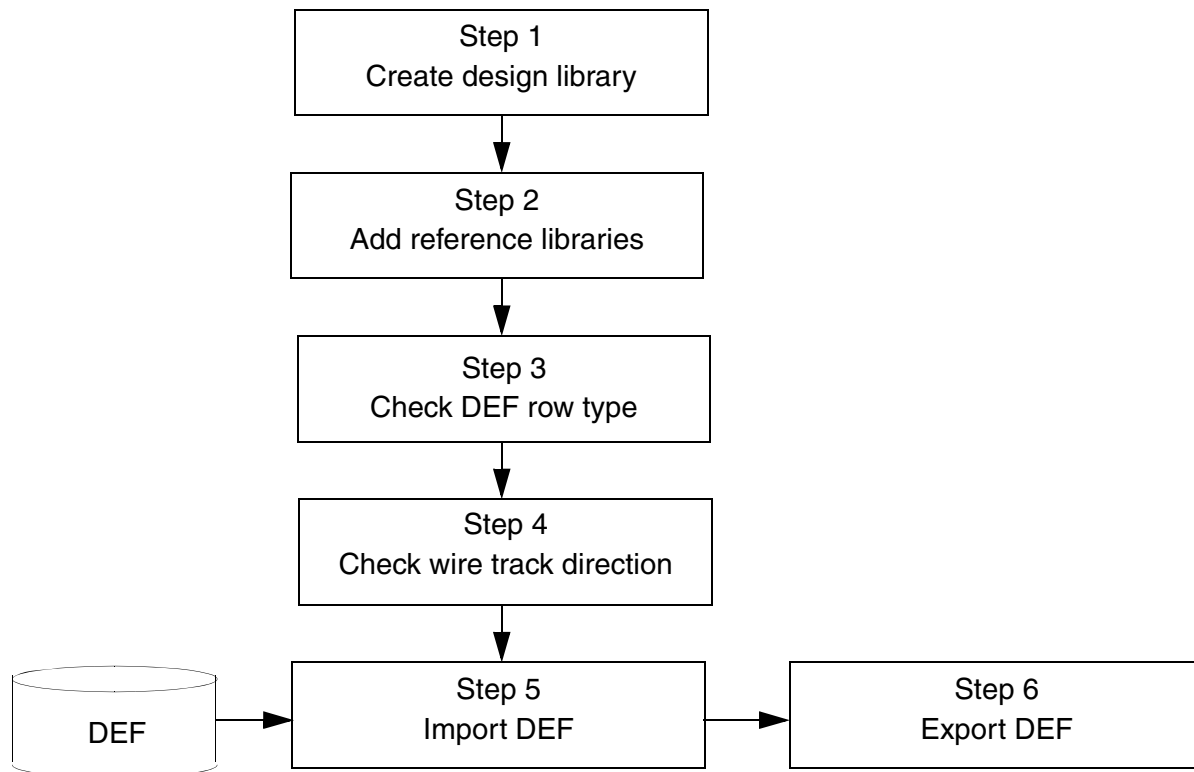
```
[-all_vias]
[-nondefault_rule]
[-lef lef_file_name]
[-regions_groups]
[-components]
[-macro]
[-fixed]
[-placed]
[-pins]
[-blockages]
[-specialnets]
[-nets]
[-routed_nets]
[-diode_pins]
[-scanchain]
[-notch_gap]
[-floating_metal_fill]
[-pg_metal_fill]
[-no_legalize]
[-verbose]
```

For details, see the man page for the `write_def` command, available on SolvNet and in the Milkyway Environment tool.

---

## Recommended DEF Flows

The Milkyway DEF flow supports incremental DEF input using Verilog. This flow supports Synopsys physical implementation tools as well as third-party place-and-route tools. The overall flow for importing DEF files into a Milkyway library and exporting the completed Milkyway library to DEF format is illustrated in [Figure 4-10](#).

*Figure 4-10 Flow for Importing and Exporting DEF Files*

This is the procedure for importing and exporting DEF:

1. Create a Milkyway design library from the technology (.tf) 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”), the tool 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 each layer.

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

Next, after you click the Track Options button, enable the Set Layer Preferred Direction option and 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. With all of the options set as desired, click OK in the Read DEF dialog box to import the DEF file or files into the Milkyway design library.
6. Export the Design information to a DEF file using the Output > DEF Out dialog box.

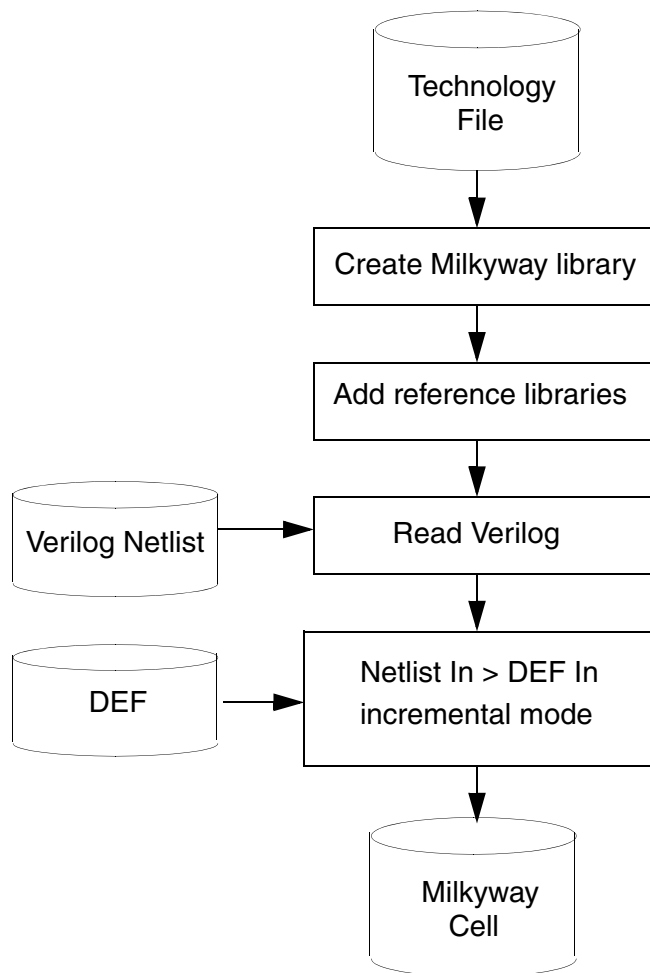
There are two types of vias in DEF. One type is the set of vias defined in the Milkyway technology file. The other type is the set of vias generated by the implementation tool during design preparation and not defined in the technology file.

For each via referenced in the net section wiring, if a match to any of the Milkyway technology contact codes cannot be 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.

DEF can be read incrementally, which means that reading DEF files can add more information to the existing cell in the Milkyway database. For example, if you have a placed cell and want to add clock routing on top of this cell using a DEF file, set the Netlist Input Mode and Physical Input Mode options to Incremental in the Read DEF dialog box.

[Figure 4-11](#) shows the incremental DEF flow of reading Verilog followed by DEF.

Figure 4-11 Verilog Incremental DEF Flow



This is the procedure in the 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 Netlist In > Verilog to CEL dialog box or the `read_verilog` command in Tcl mode.
4. Read the DEF file or files in incremental mode.
  - Use the Netlist In > DEF In (Read DEF) dialog box or the `read_def` Tcl command to read the DEF file. The cell name should be the same as that created during Verilog input.
  - Select the incremental mode in the Read DEF dialog box by setting the Netlist Input Mode and Physical Input Mode options to Incremental.



# 5

## Library Cell Preparation

---

After you translate the physical design data into Milkyway library cells, you need to prepare the cells in the Milkyway Environment tool by performing the steps described in the following sections:

- [Identify Power and Ground Ports](#)
- [Flatten Hierarchical Cells](#)
- [Extract Blockage, Pin, and Via Information for FRAM View](#)
- [Set the Place-and-Route Boundary](#)
- [Set the Multiple-Height Boundary](#)
- [Double-Height Standard Cell With Two Power Supplies](#)
- [Define the Unit Tile Wire Tracks](#)
- [Define the Antenna Properties](#)
- [Import Pin Attributes From .db Files](#)
- [Create Cell Properties](#)

## Identify Power and Ground Ports

After you import the cells to a Milkyway library from physical data that does not contain power and ground information, you must identify the power and ground ports in the cells. This must be done before blockage, pin, and via extraction to make the FRAM view for a cell.

At this point in the library preparation flow, you use the `set_attribute` command in Tcl mode to identify power and ground ports. In the Milkyway Environment tool, open the Milkyway library and open the imported cell. Then, in Tcl mode, use the `set_attribute` command as shown in the following example:

```
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 this example, the port named VDD is identified as a `Power` port and the port named GND is identified as a `Ground` port in the cell. After you set the attributes, save the modified cell.

To find out about the allowed `port_type` settings, use the following command:

```
Milkyway> list_attributes -class port -application
...
Attribute Name      Object      Type      Properties      Constraints
...
port_type           port       string    A               Input, Output, Inout,
Tristate, Power, Ground, Clock, ... BackupPG, InternalPG, BiasPG
```

To report the current settings on the port named VCC, use the following command:

```
Milkyway> report_attribute -class port -application [get_ports VCC]
...
Design      Object      Type      Attribute Name      Value
...
my_cell     VCC        string    port_type           Power
...
```

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`. If the cell has backup or internal power and ground pins, set the `port_type` attribute for each pin to `InternalPG` or `BackupPG`.

For libraries that are to be used with the IEEE 1801 Standard, also known as the Unified Power Format (UPF), you might need to update the .db library and Milkyway library cells with secondary power and ground information. For details, see the *UPF Library Preparation Application Notes*, available on SolvNet.



## Flatten Hierarchical Cells

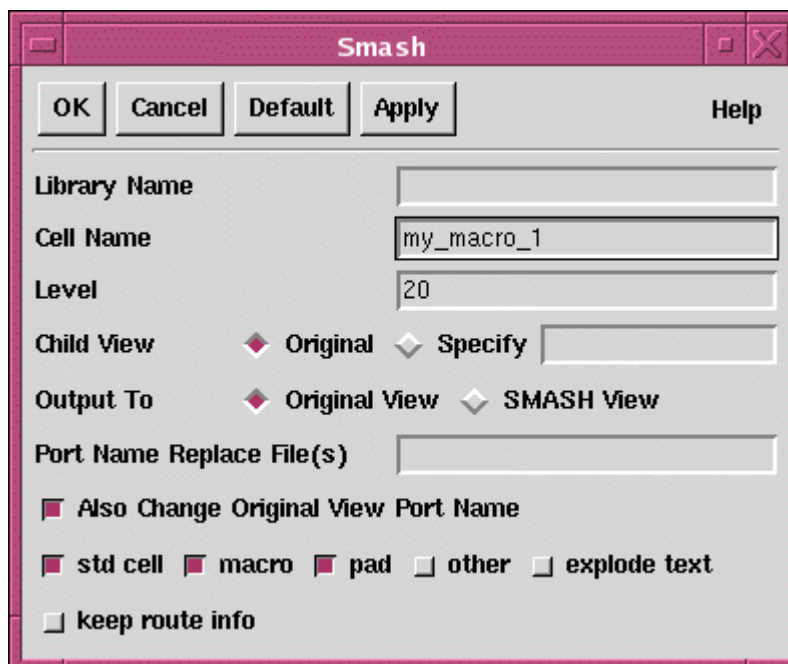
To perform processing that involves geometries inside the hierarchy of a cell, you must first flatten the cell hierarchy to a level sufficient to provide the access required to perform the processing. For example, to remove unnecessary metal2 from metal1 pins at a lower level of hierarchy, you must flatten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

Flattening a cell removes a specified number of levels of hierarchy to make lower-level objects visible at the top level. This is often needed when importing a macro cell. When needed, flattening must be done before blockage, pin, and via extraction for creating a FRAM view.

The Milkyway Environment tool might need access to geometries inside the hierarchy of the cell. In some cases, however, you might want to limit the access to lower levels. For example, you might want to limit the hierarchical access in blocks to avoid unnecessary processing and to achieve the blockage areas you want.

You can flatten a cell by using either the GUI or the `flatten_cell` Tcl command. To use the GUI, choose Cell Library > Smash. This opens the Smash dialog box, as shown in [Figure 5-1](#).

Figure 5-1 Cell Library > Smash Dialog Box



In the Library Name text box, enter the name of the Milkyway library, or leave the box blank to flatten one or more cells in the currently open Milkyway library. In the cell name text box, enter the name of one cell or an asterisk character (\*) to flatten all cells in the library.

The dialog box contains options to specify the number of hierarchical levels flattened, the types of views flattened, the type of new view created by flattening, and the handling of port names. For detailed information about these options, click the Help button in the dialog box or see the man page for the Tcl command `flatten_cell` in Tcl mode.

After you set the dialog box options, click OK. The Milkyway Environment tool traverses the hierarchy of each cell, flattens the hierarchy, and creates a new view of the cell as specified by the dialog box settings.

Flattening a cell in an unopened Milkyway library changes the contents of the unopened library. Flattening a cell in the open Milkyway library changes the cell in memory; you need to save the new or modified cell to keep the result.

Instead of using the GUI, you can use the equivalent Tcl command, `flatten_cell`. For details, see the man page for the `flatten_cell` command, available on SolvNet and in the Milkyway Environment tool.

---

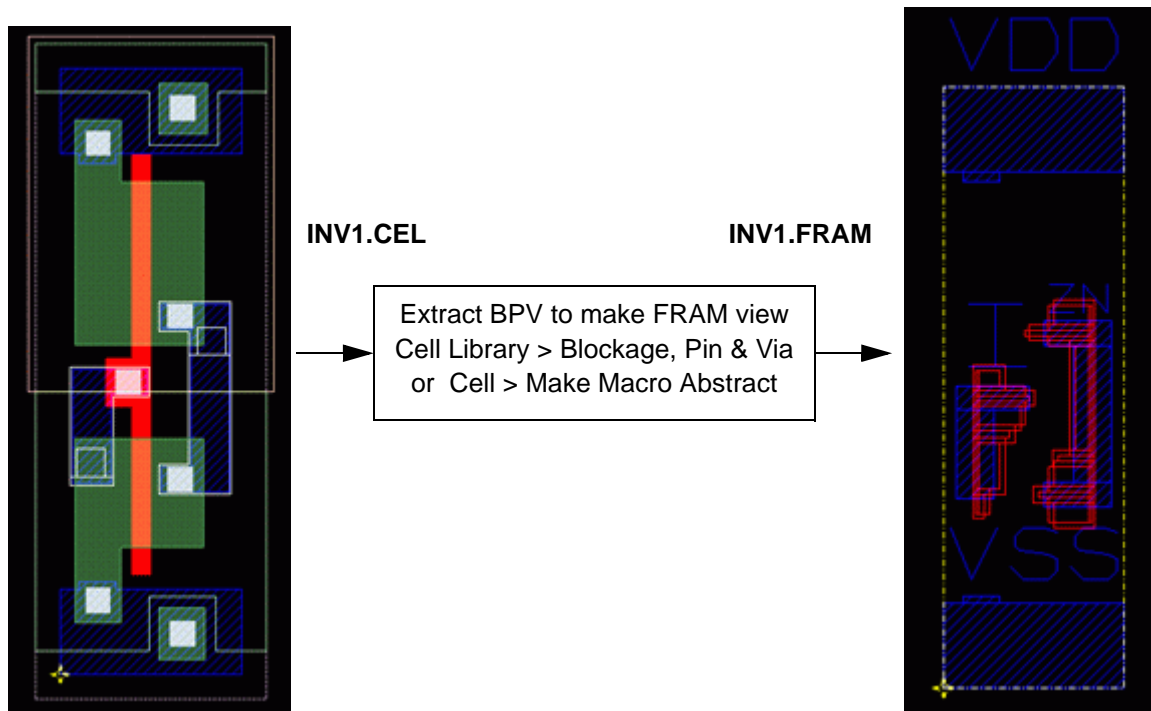
## Extract Blockage, Pin, and Via Information for FRAM View

In a Milkyway reference library that is fully prepared for use in the IC Compiler tool, each cell usually has both a CEL view and a FRAM view. The CEL view contains all of the physical mask layers of the cell, whereas the FRAM view is an abstraction of the cell containing only the information needed for placement and routing at the next higher level of the design hierarchy.

When you place an instance of a cell, you use the FRAM view of the cell. The FRAM view includes the pin names and locations, the metal blockage areas where routes are not allowed, and the allowed via areas. Other cell features are omitted in the FRAM view. The exclusion of unnecessary data can reduce the database size, increase the routing accuracy, and reduce the routing time.

Figure 5-2 shows two views of a standard cell, an inverter. The CEL view is on the left and the FRAM view is on the right. The CEL view contains all of the mask layers of the cell, including the polysilicon, diffusion, and internal contact layers. The FRAM view has only the power rails, input and output pins, and via landing areas (via regions) over the input and output pins. Note that the extraction process slices all metal polygons into rectangles. The FRAM view contains all the information necessary to place the inverter cell in a site row and route the connections to the input and output of the cell.

Figure 5-2 CEL and FRAM Views of a Standard Cell



In the Milkyway library preparation flow, you generate the FRAM view from the CEL view by using a process called blockage, pin, and via (BPV) extraction. In this process, the tool extracts the blockage, pin, and via data from the CEL view needed for the FRAM view and excludes all other data in the cell. The extraction process offers many options regarding generation of the blockages, boundary, pins, pin text, and via regions. The option settings affect the quality and usability of the generated FRAM view.

You can create FRAM views for standard cells, pad cells, and macro cells. For standard cells and pad cells, choose **Cell Library > Blockage, Pin & Via**. For macro cells, choose **Cell > Make Macro Abstract**. The extraction process is basically the same for the two commands. However, the extraction options available in the Make Macro dialog box are targeted specifically for macro cell FRAM creation.

**Note:**

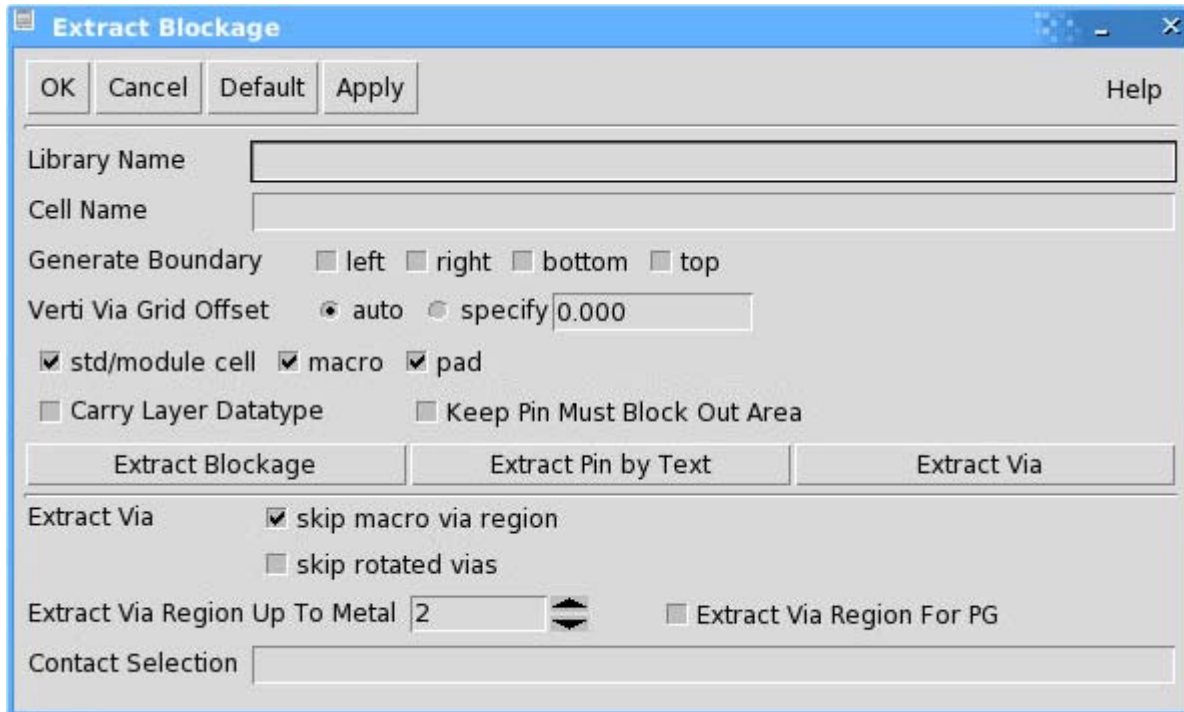
Blockage, pin, and via extraction can be done in the IC Compiler or Milkyway Environment tool. In the IC Compiler tool, use the `extract_blockage_pin_via` command to create a FRAM view for a standard cell or pad cell or use the `create_macro_fram` command to create a FRAM view for a macro. For details, see the man pages for the commands.

For additional information about blockage, pin, and via extraction, see [SolvNet article 025679](#), "Extracting FRAM Views (BPV) in Milkyway and IC Compiler Version D-2010.03."

## Extraction From Standard Cells and Pad Cells

To perform blockage, pin, and via extraction in the Milkyway Environment tool to create a FRAM view for a standard cell or pad cell, choose Cell Library > Blockage, Pin & Via. This opens the Extract Blockage dialog box, as shown in [Figure 5-3](#).

Figure 5-3 Cell Library > Blockage, Pin & Via Dialog Box



The buttons Extract Blockage, Extract Pin by Text, and Extract Via expand the dialog box to show the options related to the generation of blockages, pins, and vias, respectively. The Extract Via options are shown in the foregoing figure.

To perform FRAM view extraction, enter the name of the Milkyway library and the name of the cell from which to extract. To perform FRAM extraction for all cells in the library, enter an asterisk character (\*) for the cell name, and select the type or types of cells from which to extract from the toggle buttons provided: “std/module cell,” “macro,” and “pad.”

Use the Generate Boundary toggle buttons to specify which boundary edges are to be determined by the extraction process. By default, the existing boundary edges of the CEL view are used without change. If you generate a new boundary, the boundary depends on the locations of the extracted blockages, pins, and via landing areas.

The Verti Via Grid Offset option lets you specify how the placement tool places the cell to achieve optimum horizontal via placement. For details, see [“Horizontal Via Placement” on page 5-19](#).

The Carry Layer Datatype option causes layer datatype information to be carried over from the CEL to the FRAM view. This action is needed in some advanced technologies that apply different design rules to different types of metal objects in the same metal layer, each type of object having its own layer datatype and design rules.

The Keep Pin Must Block Out Area option enables usage of system metal blockage in the CEL view to prevent pins from being accessed by vias when the system metal blockages touch or overlap pins. By default, the extraction process trims away system metal blockages to keep enough spacing from pins and then merges the trimmed metal blockage with the non-pin real metal. By using this option, the system metal blockages are retained without trimming, allowing them to restrict the via connection area.

## Extract Blockage Options

When you click the Extract Blockage button, the dialog box displays the blockage extraction options, as shown in [Figure 5-4](#). These options specify how metal blockages are extracted from the CEL view and converted into blockages in the FRAM view.

Figure 5-4 Blockage Extraction Options in Extract Blockage Dialog Box

Extract Blockage    Extract Pin by Text    Extract Via

Extract Blockage    ☐ preserve all metal blockage

☐ Treat All Blockage as Thin Wire

Routing Blockage Output Layer

☒ Metal Blockage    ☐ Route Guide    ☐ zero min-spacing Route Guide

Treat Blockage    ☐ Merge Blockage

Poly	<input type="checkbox"/> As Thin	Poly Threshold	0.000
Metal1	<input type="checkbox"/> As Thin	Metal1 Threshold	0.000
Metal2	<input type="checkbox"/> As Thin	Metal2 Threshold	0.000
Metal3	<input type="checkbox"/> As Thin	Metal3 Threshold	0.000
Metal4	<input type="checkbox"/> As Thin	Metal4 Threshold	0.000
Metal5	<input type="checkbox"/> As Thin	Metal5 Threshold	0.000
Metal6	<input type="checkbox"/> As Thin	Metal6 Threshold	0.000
Metal7	<input type="checkbox"/> As Thin	Metal7 Threshold	0.000
Metal8	<input type="checkbox"/> As Thin	Metal8 Threshold	0.000
Metal9	<input type="checkbox"/> As Thin	Metal9 Threshold	0.000
Metal10	<input type="checkbox"/> As Thin	Metal10 Threshold	0.000
Metal11	<input type="checkbox"/> As Thin	Metal11 Threshold	0.000
Metal12	<input type="checkbox"/> As Thin	Metal12 Threshold	0.000
		PolyCont Threshold	0.000
		Via1 Threshold	0.000

## Blockage Types

System metal blockages defined in the CEL view can be retained in the FRAM view or converted into system metal blockages, route guides, or zero-minimum-spacing route guides in the FRAM view. Proper extraction of blockages can be used to prevent DRC violations between routes and cell-internal metal, while allowing the greatest possible flexibility to the router. You can choose to preserve all metal blockages or have them trimmed away from pins to allow route connections while preventing DRC violations. For a macro or pad cell, you can also choose to have some or all routing layers treated as thin wire rather than fat wire for blockage extraction and DRC purposes.

System metal blockage can be added to the CEL view or created automatically during FRAM view creation. It can be used to cover up the routing area of the cell. The router always treats system metal blockage as thin wire for design rule checking purposes.

Real metal blockage is a real metal layer in the FRAM view, either extracted from all metal geometries other than pins or converted from system metal blockages in the CEL view. The router observes all design rules, just like for any real metal.

Route guides are geometries in the design that guide the behavior of the router. The route guides produced by blockage extraction are route exclusion areas. The router treats ordinary route guides as thin metal but checks only basic spacing rules between the route guide and the route. The router treats zero-minimum-spacing route guides as geometries that routes can touch but not overlap.

## Preserve All System Metal Blockage

To preserve all system metal blockage layers as they exist in the CEL view, select the “preserve all metal blockage” option. The system metal blockages are retained in the FRAM view and remain system metal blockage. Make sure that the preserved system metal blockages do not violate the design rules. This option is recommended for standard cell input in LEF format.

## Treat Real Metal Blockage as Thin Wire

By default, real metal blockages are treated as fat wire if fat or thin wire if thin for design rule checking purposes. The spacing rules can be different for thin versus fat wire, which affects the spacing required between routes and real metal blockages.

In macro and pad cells, the “Treat All Blockage as Thin Wire” option causes all real metal blockages to be treated as thin wire for design rule checking purposes. This option sets a flag on the FRAM model that tells the router to treat all real metal blockage layers as thin. This option setting also affects the trimming of real metal blockages during extraction.

To treat only some layers as thin, do not select the “Treat All Blockage as Thin Wire” option. Use the Treat Blockage options that you can set for each layer: “As Thin,” “block all,” or “merge.” If you select “As Thin” for a layer, the router trims the metal back to emulate the

“treat as thin” behavior. In that case, the router sees the metal as fat, but because it has been trimmed back, the final effect is to observe the min spacing rule.

The amount by which the metal is trimmed back is:

$(\text{max value in fatTblSpacing}) - (\text{minSpacing})$

### Routing Blockage Output Layer

The Routing Blockage Output Layer options specify how metal blockage is implemented in the FRAM view: as system metal blockage, ordinary route guide, or zero-minimum-spacing route guide. The router recognizes all three types of layers as blockage and will not route over them, but the spacing rules vary from one type to another.

### Treat Blockage As Thin and Merge Blockage

Use the Treat Blockage options to control the extraction of real metal blockage layers on an individual layer basis.

The “As Thin” option causes the metal blockage to be trimmed back so that the router, in effect, treats the metal as thin for design rule checking purposes. See the previous section, [“Treat Real Metal Blockage as Thin Wire” on page 5-8](#).

The “Merge Blockage” option merges geometries that are close together. If two nearby geometries are within the specified spacing threshold, the extraction process fills in the space between the two shapes, thereby creating a single larger shape for the blockage in the FRAM view. One possible value to use for the spacing threshold is twice the `minSpacing` value plus the `minWidth` value specified in the technology file. This is the narrowest space that could allow a single route to pass between two shapes in that layer.

### Extract Pin by Text Options

When you click the Extract Blockage button, the dialog box displays the blockage extraction options, as shown in [Figure 5-5](#). These options specify how pins are identified by text objects in the CEL view so that they can be properly extracted to make the FRAM view.



Figure 5-5 Extract Pin by Text Options in Extract Blockage Dialog Box

Extract Pin

☐ Extract Connectivity

through ☐ polyCont ☒ via1 ☒ via2 ☒ via3 ☒ via4 ☒ via5 ☒ via6

☒ via7 ☒ via8 ☒ via9 ☒ via10 ☒ via11

☒ via12 ☒ via13 ☒ via14 ☒ bvia1 ☒ bvia2

☐ text fall through ☐ double fall through

Poly Text	<input type="text"/>	Metal1 Text	<input type="text"/>
Metal2 Text	<input type="text"/>	Metal3 Text	<input type="text"/>
Metal4 Text	<input type="text"/>	Metal5 Text	<input type="text"/>
Metal6 Text	<input type="text"/>	Metal7 Text	<input type="text"/>
Metal8 Text	<input type="text"/>	Metal9 Text	<input type="text"/>
Metal10 Text	<input type="text"/>	Metal11 Text	<input type="text"/>
Metal12 Text	<input type="text"/>	Metal13 Text	<input type="text"/>
Metal14 Text	<input type="text"/>	Metal15 Text	<input type="text"/>
Fall Through Text	<input type="text"/>	Nwell Text	<input type="text"/>
Pwell Text	<input type="text"/>	DeepNwell Text	<input type="text"/>
DeepPwell Text	<input type="text"/>	Back Metal1 Text	<input type="text"/>
Back Metal2 Text	<input type="text"/>	Back Metal3 Text	<input type="text"/>

☐ ignore internal macro pin access edges

☐ expand small pin on blockage for contact connection

Pin must-connect area layer number      Auto-compute pin must-connect area

The “Extract Connectivity through” options cause the extraction process to traverse connectivity through the specified cut and via layers to identify connections between pins on different layers and extract the entire pin.

Select the “text fall through” option to use the metal layers beneath the origin of each text label to identify the geometry annotated by the text. Select the “double fall through” option if more than one geometry exists beneath the origin of a text label in two different layers and you want both to be named by the same text label.

For each of the layer text boxes (Poly Text, Metal1 Text, and so on), enter the name or number of the layer used to identify the geometries of that layer. Leave the box blank if the text layer is the same as the associated geometry layer.

Select the “expand small pin on blockage for contact connection” option if you have small metal1 pins that are less than the minimum width inside the metal1 blockage. The extraction

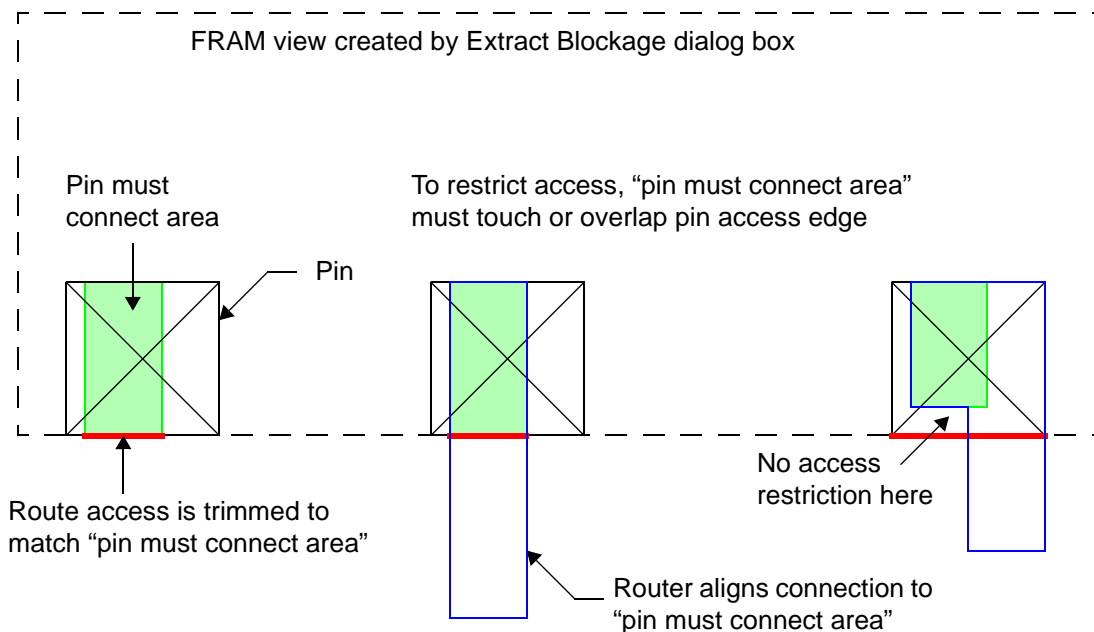


process cuts an extra-large space through the metal1 blockage around the metal1 pins to give the router access to the pins.

Click the “Pin must-connect area layer number” button to view the pin must-connect options. For each connection layer, the respective text box specifies the associated spare layer being used to designate the required via connection area within the pin geometry. By default, the router can connect a wire to any part of a large pin. To restrict the connection to a subarea of a large pin, specify the subarea polygon on a spare layer and specify the spare area associated with each connection layer using this option.

When the “pin must connect area” touches or overlaps the edge of the pin, access to the “pin must connect area” is restricted to that area only, as shown by the red line in [Figure 5-6](#). In that case, the IC Compiler router aligns the connection to the “pin must connect area.”

Figure 5-6 “Pin Must Connect Area” Route Access



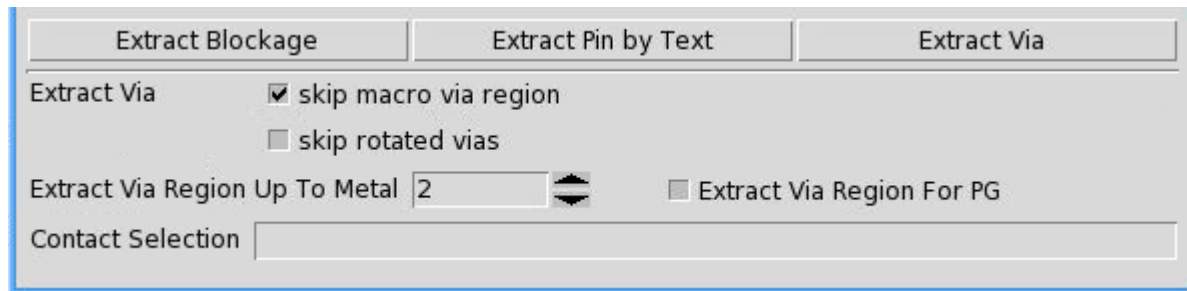
This feature allows precise control over route access to pins. Note that if the “pin must connect area” geometry does not touch or overlap the access edge of the pin, there is no restriction on route access through the pin to that area.

Click the “Auto-compute must-connect area” button to view the auto-compute must-connect options. These options specify, for each connection layer, the width threshold for automatic determination of the required via connection area within the pin geometry. To automatically restrict the connection to a subarea of a pin with an irregular shape, use this option and specify each layer and its associated width threshold.

## Extract Via Options

When you click the Extract Blockage button, the dialog box displays the via extraction options, as shown in [Figure 5-7](#).

Figure 5-7 Blockage Via Options in Extract Blockage Dialog Box



Select the “skip macro via region” option to skip making via regions for macros.

Select the “skip rotated vias” option to skip consideration of rotated vias during calculation of via regions. Otherwise, the extraction process considers the use of both default and rotated vias during via region calculation.

The “Extract Via Region Up To Metal” option specifies the top metal layer for which to extract via regions.

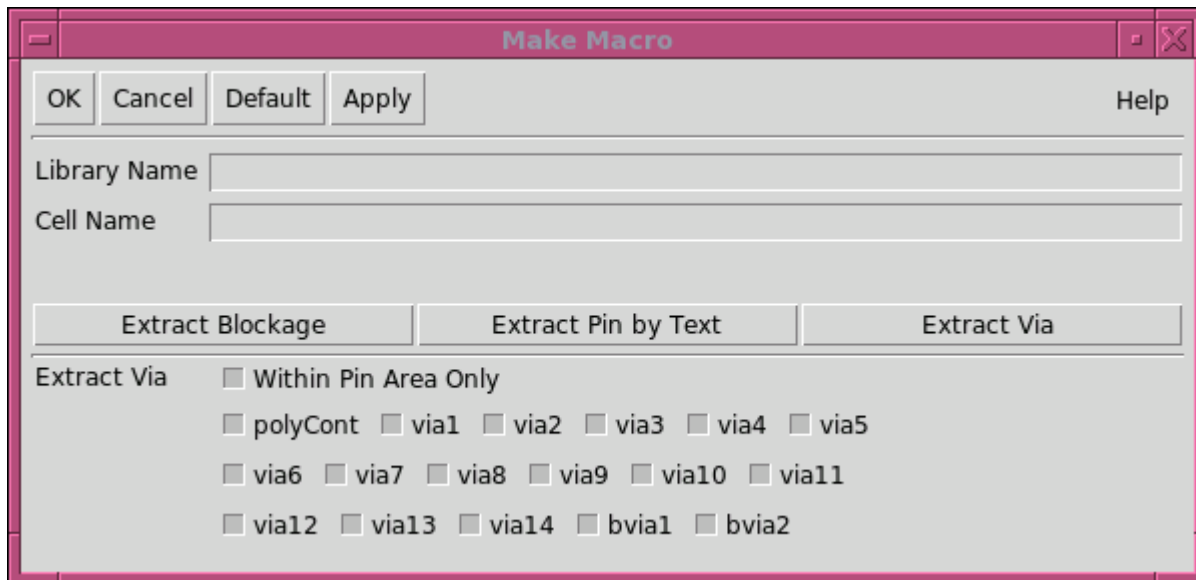
Select the “Extract Via Region For PG” option to extract via connection areas for PG pins. By default, via regions are extracted only for signal pins, not PG pins.

In the “Contact Selection” text box, contact code numbers of any additional vias that are to be considered during calculation of via regions. Otherwise, only default vias are considered.

---

## Extraction From Macros

To perform blockage, pin, and via extraction in the Milkyway Environment tool to create a FRAM view for a macro cell, choose Cell > Make Macro Abstract. This opens the Extract Blockage dialog box, like the example shown in [Figure 5-8](#).

*Figure 5-8 Cell > Make Macro Abstract Dialog Box*

The buttons Extract Blockage, Extract Pin by Text, and Extract Via expand the dialog box to show the options related to the generation of blockages, pins, and vias, respectively. The Extract Via options are shown in the preceding figure.

To perform FRAM view extraction, enter the name of the Milkyway library and the name of the cell from which to extract. By default, these text boxes are filled in with the name of the currently open Milkyway library and the name of the current cell.

## Extract Blockage Options

When you click the Extract Blockage button, the dialog box displays the blockage extraction options, as shown in [Figure 5-9](#). These options specify how metal blockages are extracted from the CEL view and converted into blockages in the FRAM view.

Figure 5-9 Blockage Extraction Options in Make Macro Dialog Box

Treat Blockage				Threshold X	Threshold Y	Block Core With Margin	FeedThru Layer
Poly	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 1	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 2	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 3	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 4	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 5	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 6	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 7	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 8	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	
Metal 9	<input type="checkbox"/> As Thin	<input checked="" type="radio"/> block all	<input type="radio"/> merge	0.000	0.000	-1	

The Extract Blockage options are similar to those described in the section [“Extract Blockage Options” on page 5-7](#) for the Cell Library > Blockage, Pin & Via (Extract Blockage) dialog box. However, the Cell > Make Macro Abstract (Make Macro) dialog box has additional options to control the generation of core blockages over the whole macro area.

Select the “block all” option to create a blockage on the entire cell area for the specified layer, which prevents over-the-cell routing for that layer in the macro. For each such layer, use the “Block Core With Margin” text box to specify the amount of open area between the core blockage and the cell boundary:

- A value of –1 results in no blockage.
- A value of 0 results in a core blockage with automatic computation of the margin.
- A positive value results in a core blockage with a margin between the blockage and the cell boundary equal to the entered value.

In the “FeedThru Layer” text box, enter the name of a layer that reserves some areas that retain the metal as it exists in the original CEL view. This selection prevents the extraction process from covering the specified area with blockage and also prevents merging the metal geometries in that area.

The Extract Blockage dialog box allows a single merging threshold, whereas the Make Macro dialog box allows different thresholds to be entered for the X and Y directions.

## Extract Pin by Text Options

When you click the Extract Blockage button, the dialog box displays the blockage extraction options, as shown in [Figure 5-10](#).

Figure 5-10 Extract Pin by Text Options in Make Macro Dialog Box

The dialog box is titled "Extract Pin by Text" and is part of the "Make Macro" process. It contains the following options:

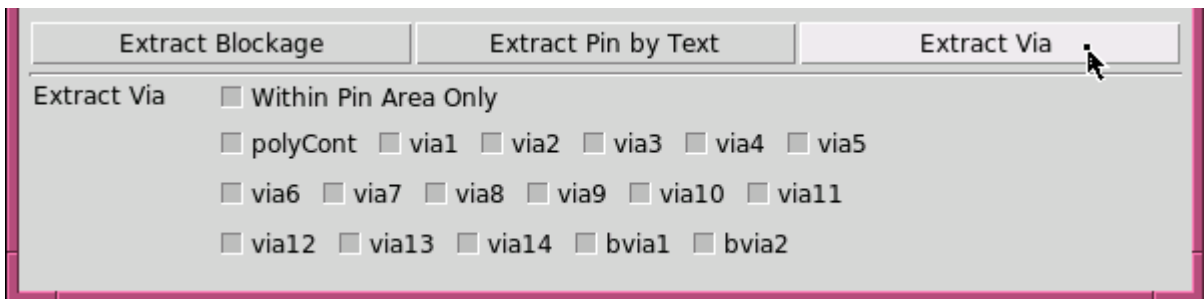
- ☐ Identify Macro Pin By Pin Text
  - ☐ Extract Connectivity
    - through ☐ polyCont ☒ via1 ☒ via2 ☒ via3 ☒ via4 ☒ via5
    - ☒ via6 ☒ via7 ☒ via8 ☒ via9 ☒ via10 ☒ via11
    - ☒ via12 ☒ via13 ☒ via14 ☒ bvia1 ☒ bvia2
  - ☐ expand small pin on blockage for contact connection
  - ☐ text fall through ☐ double fall through
- Text input fields for:
  - Poly Text
  - Metal1 Text
  - Metal2 Text
  - Metal3 Text
  - Metal4 Text
  - Metal5 Text
  - Metal6 Text
  - Metal7 Text
  - Metal8 Text
  - Metal9 Text
  - Metal10 Text
  - Metal11 Text
  - Metal12 Text
  - Metal13 Text
  - Metal14 Text
  - Metal15 Text
  - Fall Through Text
  - Nwell Text
  - Pwell Text
  - DeepNwell Text
  - DeepPwell Text
  - Back Metal1 Text
  - Back Metal2 Text
  - Back Metal3 Text
- Buttons at the bottom:
  - Pin must-connect area layer number
  - Auto-compute pin must-connect area

The Extract Pin by Text options of the Make Macro dialog box are similar to those described in the section [“Extract Pin by Text Options” on page 5-9](#) for the Cell Library > Blockage, Pin & Via (Extract Blockage) dialog box.

## Extract Via Options

When you click the Extract Blockage button, the dialog box displays the blockage extraction options, as shown in [Figure 5-11](#).

Figure 5-11 Blockage Via Options in Make Macro Dialog Box



The Extract Via options of the Make Macro dialog box are similar to those described in the section [“Extract Via Options” on page 5-12](#) for the Cell Library > Blockage, Pin & Via (Extract Blockage) dialog box.

---

## Identifying Pins

The extraction process 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.

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. The text location identifies the associated geometry.

In the case of multiple-layer pins with text on only one layer, you select the “Extract Connectivity” option in the “Extract Pin by Text” options to trace the connectivity of the pin and extract the entire pin.

## Text 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 metal.

## Text 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 dialog box: “Poly Text” is the list of layers used for the text identifying poly geometries, “Metal1 Text” is the list of layers used for the text identifying Metal1 geometries, and so on.

Enter 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.

## All Text on the Same Layer

When all text labels are on the same layer, you need to set the “Extract Pin by Text” options as follows:

- Select “text fall through” 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, the extraction process selects the geometry according to the following priorities, from highest to lowest: Metal3, Metal2, Metal1, Poly.
- Select “double fall through” 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, the extraction process selects the second geometry as Metal1 if present, and Poly otherwise.

Enter the name or the number of the Fall Through Text layer, the layer used for text labels. The extraction process 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.

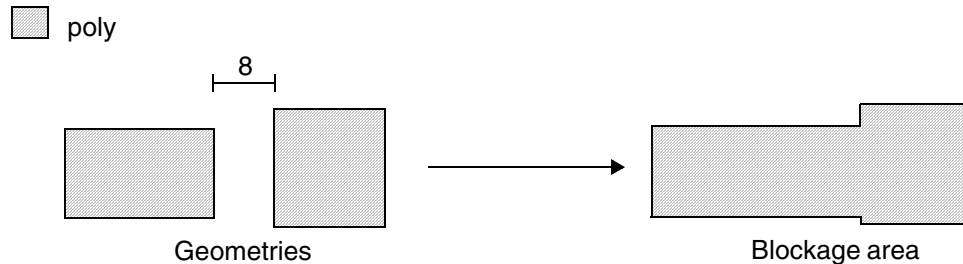
---

## 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 is twice the `minSpacing` value plus the `minWidth` value in the technology file. This is the narrowest space that can allow a single minimum-width route to pass through.

For example, if the technology file specifies a poly `minSpacing` of 2 and a poly `minWidth` of 4, the merging distance is  $2 * 2 + 4 = 8$ . Therefore, in [Figure 5-12](#), the extraction process merges the geometries on the left to create the blockage area on the right.

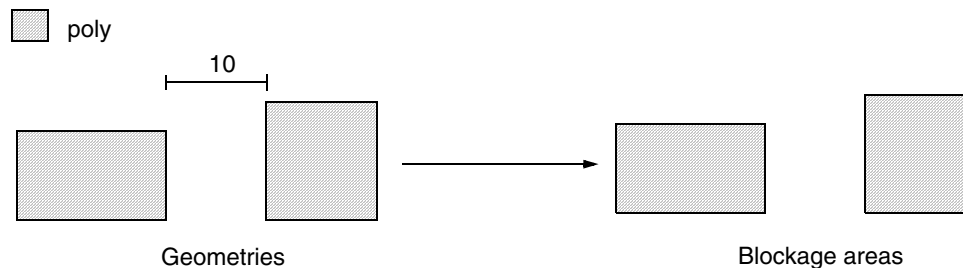
*Figure 5-12 Merging Same-Layer Geometries*



In the Extract Blockage or Make Macro dialog box, when you click the Extract Blockage button and select the merge option for a layer, you can set the merging thresholds for each layer. Entering a threshold value changes the merging distance to the entered value plus twice the `minSpacing` value defined in the technology file.

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 default merging distance is  $2 * 2 + 0 = 4$ . Therefore, in [Figure 5-13](#), no merging occurs.

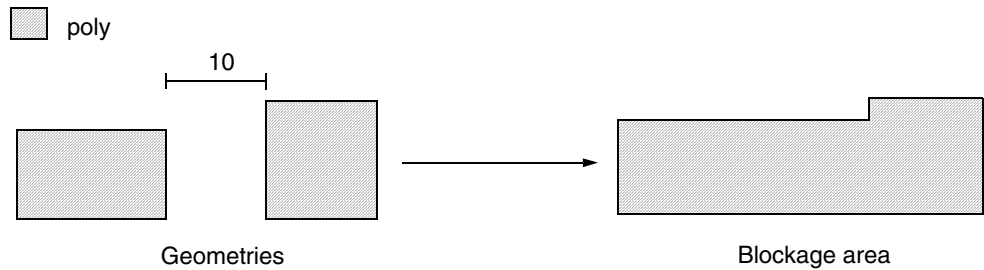
*Figure 5-13 No Merging Occurs With Threshold = 0*



However, if you specify a poly threshold of 6, the merging distance increases to  $2 * 2 + 6 = 10$ . In that case, the extraction process merges the geometries on the left to create the blockage area on the right, as shown in [Figure 5-14](#).



Figure 5-14 Merging Occurs With Threshold = 6



### Horizontal Via Placement

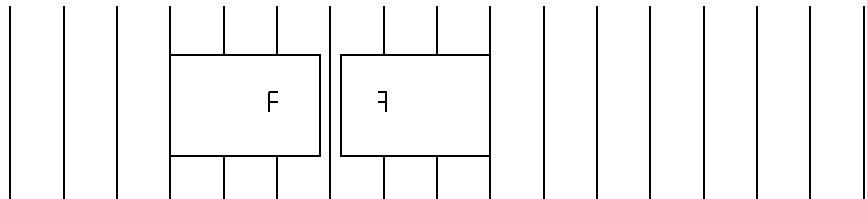
To specify how you want to place standard cells to achieve optimum horizontal via placement, use the Verti Via Grid Offset option in the Cell Library > Blockage, Pin & Via (Extract Blockage) dialog box.

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

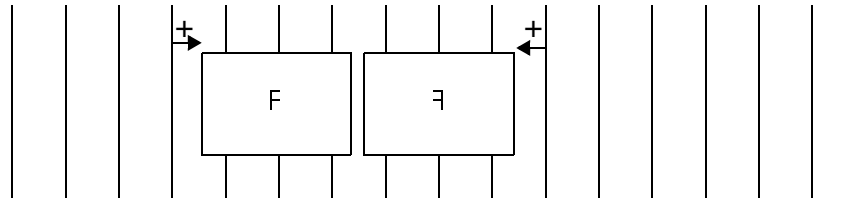
Option	Instructs how to place cells so that
auto	The maximum number of vias are on a vertical wire track
specify	The left side of an unflipped cell or the right side of a flipped cell is at a distance equal to the value you enter from a vertical wire track

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 5-15](#), select “specify” and enter the value 0.

Figure 5-15 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 5-16](#), select “specify” and enter a positive number for the distance. This value must be less than the pitch of the vertical routing layer.

*Figure 5-16 Specifying Placement of Cell at Specified Distance From Vertical Wire Track*

## Exclusion of Internal Pins

You can optionally exclude internal pins during blockage, pin, and via extraction. When you use this option, the FRAM view excludes the internal pins contained of the original CEL view, resulting in a simpler FRAM view without the unnecessary internal pins.

To invoke this option, set the Tcl variable `external_pin_file` to a list of text files or a list of .db files before you perform blockage, pin, and via extraction. The text files or .db files specify which pins are internal pins that are not be extracted. For example,

```
Milkyway> set_app_var external_pin_file {my_text_file.txt}
```

```
Milkyway> set_app_var external_pin_file {my_lib1.db my_lib2.db}
```

If you set the variable to the name a text file, you specify the names of the cells and their external pins explicitly in the text file using the following format:

```
cell_name1 {external_pin_name1 external_pin_name2 ...}
cell_name2 {external_pin_name1 external_pin_name2 ...}
```

Wildcard characters are allowed in the file. For example,

```
buf* { A B OUT }
```

In this example, all cell names beginning with “buf” are defined to have the external pins A, B, and OUT. Any other pins in those cells are considered internal pins and are not extracted.

If you set the `external_pin_file` variable to the name of a .db file, the tool looks inside the .db file to determine which pins are external and internal for each cell. External pins are extracted, including both signal pins and `pg_port` pins, and internal pins are not extracted. Bias `pg_port` pins are always extracted, whether external or internal.

After you set the variable, when you perform blockage, pin, and via extraction using Cell Library > Blockage, Pin, & Via or Cell > Make Macro Abstract, the extraction process skips extraction of internal pins for the cells listed in the text files or .db files.

---

## Via Regions

In a FRAM view, a via region is an area consisting of one or more rectangular boxes, which may be overlapping, over a pin where the router can make a connection to a port. The via region provides guidance to the router about where to drop a via to make a connection.

By default, the router can use either a wire in the pin layer or a via above the pin layer to make a connection to a pin. In the IC Compiler tool, you can restrict the allowed methods of making connections in each layer by using the following command:

```
icc_shell> set_route_zrt_common_options \  
-connect_within_pins_by_layer_name { {layer mode} {layer mode} ...}
```

When this option is set for a pin layer, the router attempts to make the lower-metal enclosure fall entirely within the metal pin in the FRAM view. This “within-pin” connection ensures that the router will not introduce any DRC violations in the lower metal layer of the cell being connected.

However, if the router is unable to fit the lower-metal enclosure within the pin, it makes the connection anyway, causing the lower-metal enclosure to extend beyond the pin shape. This type of connection is said to “change the pin shape.” In that case, the router must spend time checking for and fixing lower-metal design rule violations in the cell.

The manner of via region generation depends on the technology geometry size. For technology nodes at 90 nm and above, the blockage, pin, and via extraction process allows full flexibility to the router without considering whether the pin shape needs to be changed. For technology nodes at 65 nm and below, by default, the blockage, pin, and via extraction process creates more restrictive via areas that ensure “within-pin” via connections that do not change the metal pin shape, so that routing can be performed more quickly.

Enforcing all “within-pin” connections can be too restrictive in certain situations, such as when pin is small and the minimum enclosure around the via is large. In that case, you can override the default behavior of the `extract_blockage_pin_via` command by setting the following variable:

```
icc_shell> set_app_var via_can_change_pin_shape true
```

This allows the `extract_blockage_pin_via` command to create via regions that extend outside of pins for greater router flexibility, at the cost of more router runtime.

If pins are smaller than the minimum via size, then no via regions are generated at all by default. To allow via regions to be generated under these conditions, do the following:

```
icc_shell> set_app_var \  
generate_via_region_when_pin_width_all_less_than_via_width true
```

This causes the `extract_blockage_pin_via` command to generate via regions for pins, even when the width of the whole pin is smaller than the via width.

The IC Compiler tool lets you edit via regions in FRAM views so that you can control the behavior of the router in making connections to ports through vias. These are the commands that you can use:

- `create_via_region` creates a new via region associated with a specified port and having specified coordinates.
- `write_via_region` writes out existing via regions as a set of `create_via_region` commands, which lets you edit the existing via regions.
- `remove_via_region` removes an existing via region so that you can replace it with a new via region.

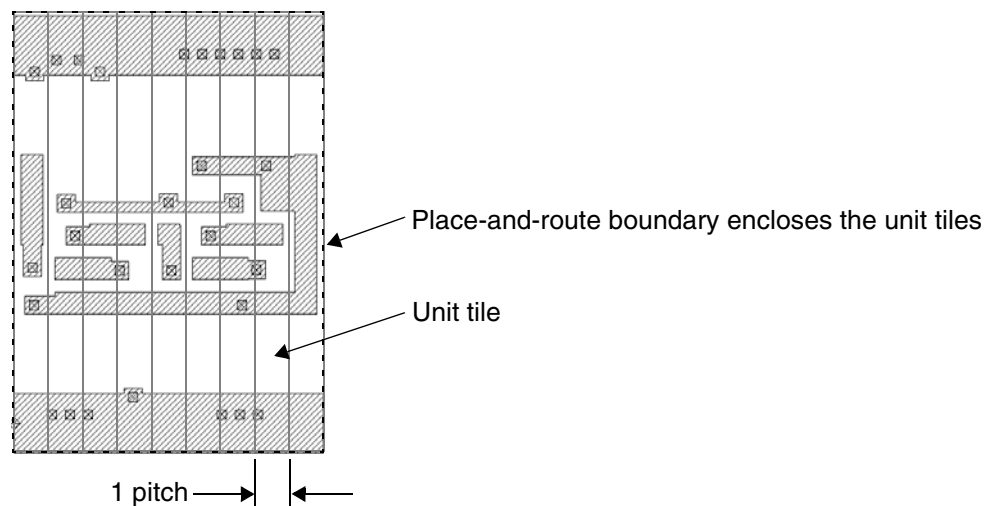
For details, see the man pages for these commands.

---

## Set 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 5-17](#).

*Figure 5-17 Place-and-Route Boundary Example*



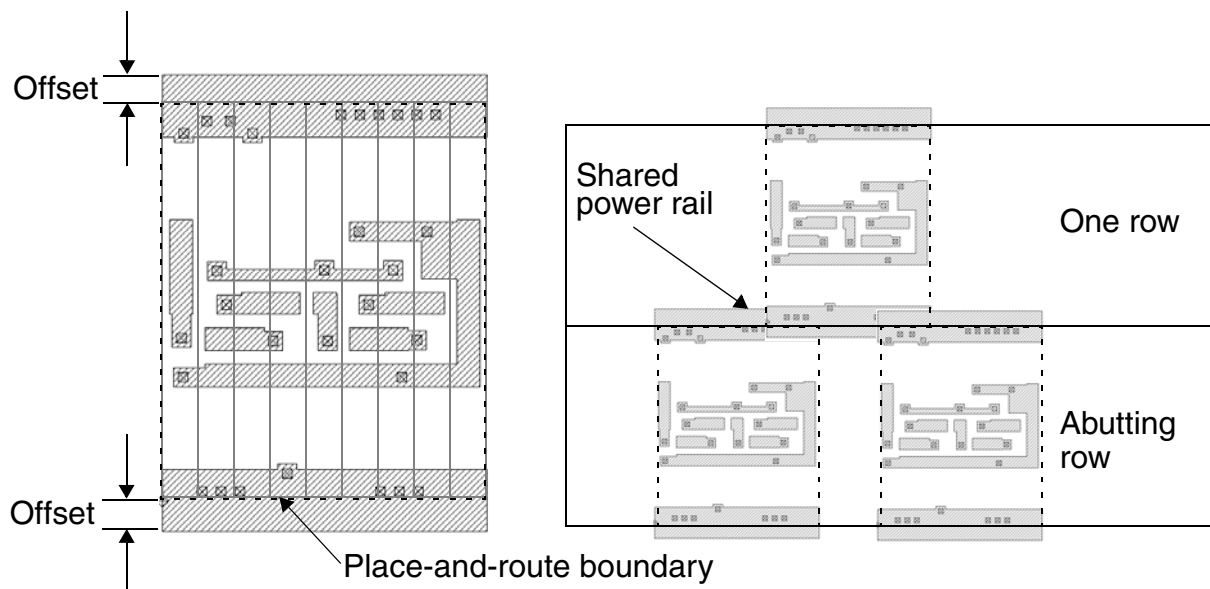
You generate place-and-route boundaries for cells and create the unit tile in your library by choosing the Cell Library > Set PR Boundary command in the Milkyway Environment tool.

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 a whole 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. A multiheight standard cell must be designed to use the same power rail as the single-height cells in the first lower rail position in the cell.

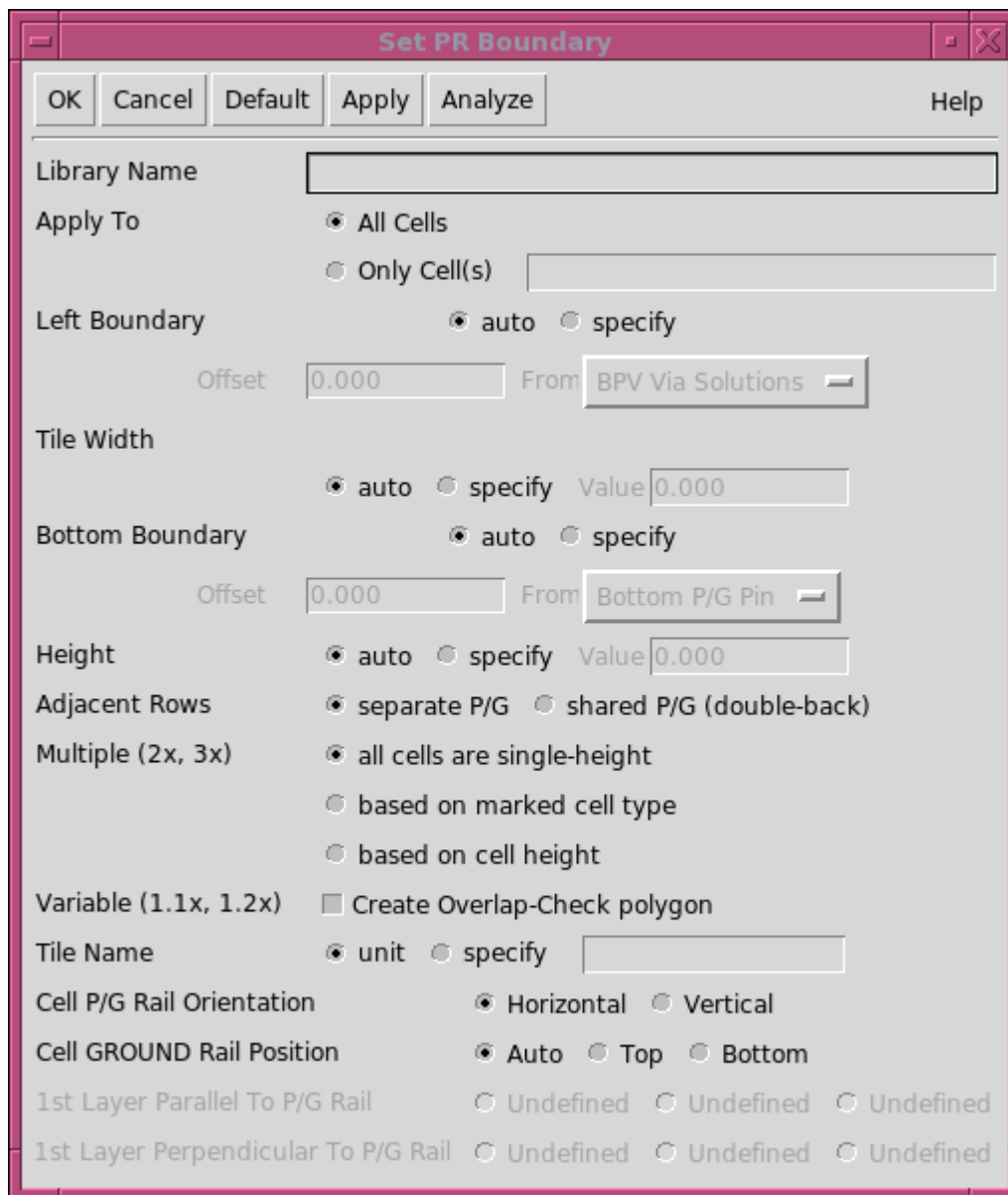
If the cells are designed to overlap, you should make the place-and-route boundary smaller. For example, to make the power and ground pins of abutting rows overlap in a back-to-back floorplan, decrease the height of the place-and-route boundary so the cells abut along the center of the power and ground pins, as shown in [Figure 5-18](#).

*Figure 5-18 Overlapping Abutting Rows Example*



To set the place-and-route boundary in the Milkyway Environment tool, choose Cell Library > Set PR Boundary. This opens the Set PR Boundary dialog box, as shown in [Figure 5-19](#). The dialog box has options to specify the library and cell or cells, the method of determining the left and bottom boundaries with respect to the cell, the cell height, the height multiple, the type of overlap, and the power and ground rail orientation. For full details, see the online help topic “auSetPRBdry” in the Milkyway Environment online help.

Figure 5-19 Cell Library &gt; Set PR Boundary Dialog Box



The dialog box is titled "Set PR Boundary" and contains the following controls:

- Buttons:** OK, Cancel, Default, Apply, Analyze, Help.
- Library Name:** A text input field.
- Apply To:**
  - ☒ All Cells
  - ☐ Only Cell(s) [Text input field]
- Left Boundary:**
  - ☒ auto ☐ specify
  - Offset: [0.000] From: [BPV Via Solutions]
- Tile Width:**
  - ☒ auto ☐ specify Value: [0.000]
- Bottom Boundary:**
  - ☒ auto ☐ specify
  - Offset: [0.000] From: [Bottom P/G Pin]
- Height:**
  - ☒ auto ☐ specify Value: [0.000]
- Adjacent Rows:**
  - ☒ separate P/G ☐ shared P/G (double-back)
- Multiple (2x, 3x):**
  - ☒ all cells are single-height
  - ☐ based on marked cell type
  - ☐ based on cell height
- Variable (1.1x, 1.2x):**
  - ☐ Create Overlap-Check polygon
- Tile Name:**
  - ☒ unit ☐ specify [Text input field]
- Cell P/G Rail Orientation:**
  - ☒ Horizontal ☐ Vertical
- Cell GROUND Rail Position:**
  - ☒ Auto ☐ Top ☐ Bottom
- 1st Layer Parallel To P/G Rail:**
  - ☐ Undefined ☐ Undefined ☐ Undefined
- 1st Layer Perpendicular To P/G Rail:**
  - ☐ Undefined ☐ Undefined ☐ Undefined

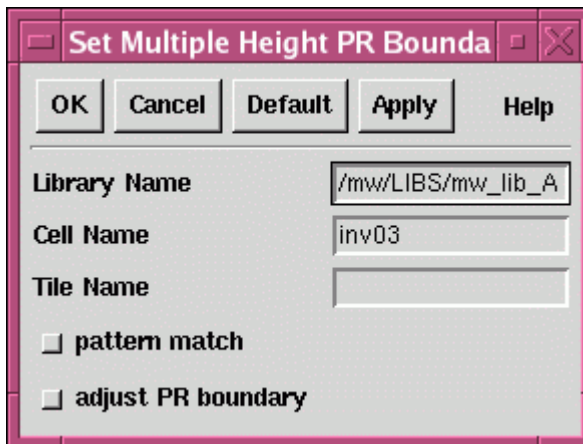
Enter the required information in the Set PR Boundary dialog box. To get suggested values for the offset and height values, click the Analyze button. With the options filled in, click the OK button to generate the cell boundary or boundaries.

---

## Set the Multiple-Height Boundary

You can adjust the place-and-route boundary for a double-height or triple-height cell by choosing the Cell Library > Multi Height Cell command in the Milkyway Environment tool. This command checks the power and ground height and width, suggests floorplan options, and checks the row spacing rule in the technology file. The command opens the Set Multiple Height PR Boundary dialog box, as shown in [Figure 5-20](#).

Figure 5-20 Cell Library > Set PR Boundary Dialog Box



In the Library Name text box, enter the name of the Milkyway library. In the Cell Name text box, enter the name of a single cell or a wildcard pattern; use the “pattern match” option with a wildcard pattern. In the Tile Name text box, enter the name of the tile used for adjusting the boundary.

To adjust the place-and-route boundary as well as set the cell row orientation, select the “adjust PR boundary” option. Otherwise, you only set the cell row orientation.

Click the OK button to perform the requested actions.

Note:

A multiple-height standard cell must be designed to use the same power rail as the single-height cells in the first lower rail position in the cell.

---

## Double-Height Standard Cell With Two Power Supplies

By default, a standard cell has one primary power pin and one primary ground pin, located on opposite ends of the cell (top and bottom). When the cell is placed in a row, the power and ground pins are aligned with the power and ground rails of the row.

The Milkyway Environment tool supports an alternative type of standard cell that has double the standard height, two different power pins (for example, VDD and VDD5) at opposite ends of the cell, and a single ground pin at the center of the cell.

During library cell preparation, these are the key steps for setting the place-and-route boundary for a double-height, dual-power cell:

1. Set the `mwlib_second_power_on_top` variable to `true` if the second power rail is at the top, or leave it set to `false` if the second power rail is at the bottom.
2. Open the Set PR Boundary dialog box (choose Cell Library > Set PR Boundary or execute the `auSetPRBdry` command in Scheme mode). See [Figure 5-21](#).

*Figure 5-21 Set PR Boundary Dialog Box Settings for Double-Height, Dual-Power Cell*

The screenshot shows the "Set PR Boundary" dialog box with the following settings:

- Library Name:** my\_mw\_library
- Apply To:** ☒ All Cells, ☒ Only Cell(s) my\_2x\_cell
- Left Boundary:** ☒ auto, ☐ specify. Offset: 0.000. From: BPV Via Solutions
- Tile Width:** ☐ auto, ☒ specify. Value: 1.28
- Bottom Boundary:** ☒ auto, ☐ specify. Offset: 0.000. From: Bottom P/G Pin
- Height:** ☐ auto, ☒ specify. Value: 1.28
- Adjacent Rows:** ☐ separate P/G, ☒ shared P/G (double-back)
- Multiple (2x, 3x):** ☐ all cells are single-height, ☒ based on marked cell type, ☐ based on cell height
- Variable (1.1x, 1.2x):** ☐ Create Overlap-Check polygon
- Tile Name:** ☐ unit, ☒ specify. Value: 2p1gunit
- Cell P/G Rail Orientation:** ☒ Horizontal, ☐ Vertical
- Cell GROUND Rail Position:** ☒ Auto, ☐ Top, ☐ Bottom



3. Enter the Milkyway library name.
4. Set “Apply To” to “Only Cell(s)” and enter the name of the double-height cell to which you are applying the boundary.
5. Set “Height” to “specify” and enter the full height of the cell, which must span two power supply rails.
6. Set “Multiple (2x, 3x)” to “based on marked cell type”.
7. Set “Tile Name” to “specify” and enter the unit tile name in the library.
8. Click OK to close the dialog box and create the place-and-route boundary.

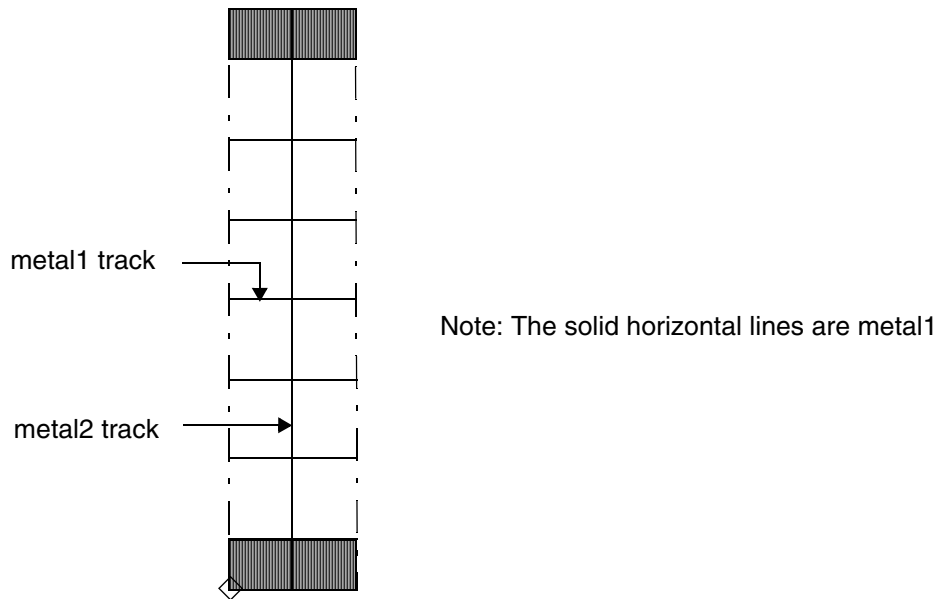
The `mwlib_second_power_on_top` variable setting has no effect on the preparation of standard cells that have only one primary power pin and one primary ground pin.

---

## Define the Unit Tile Wire Tracks

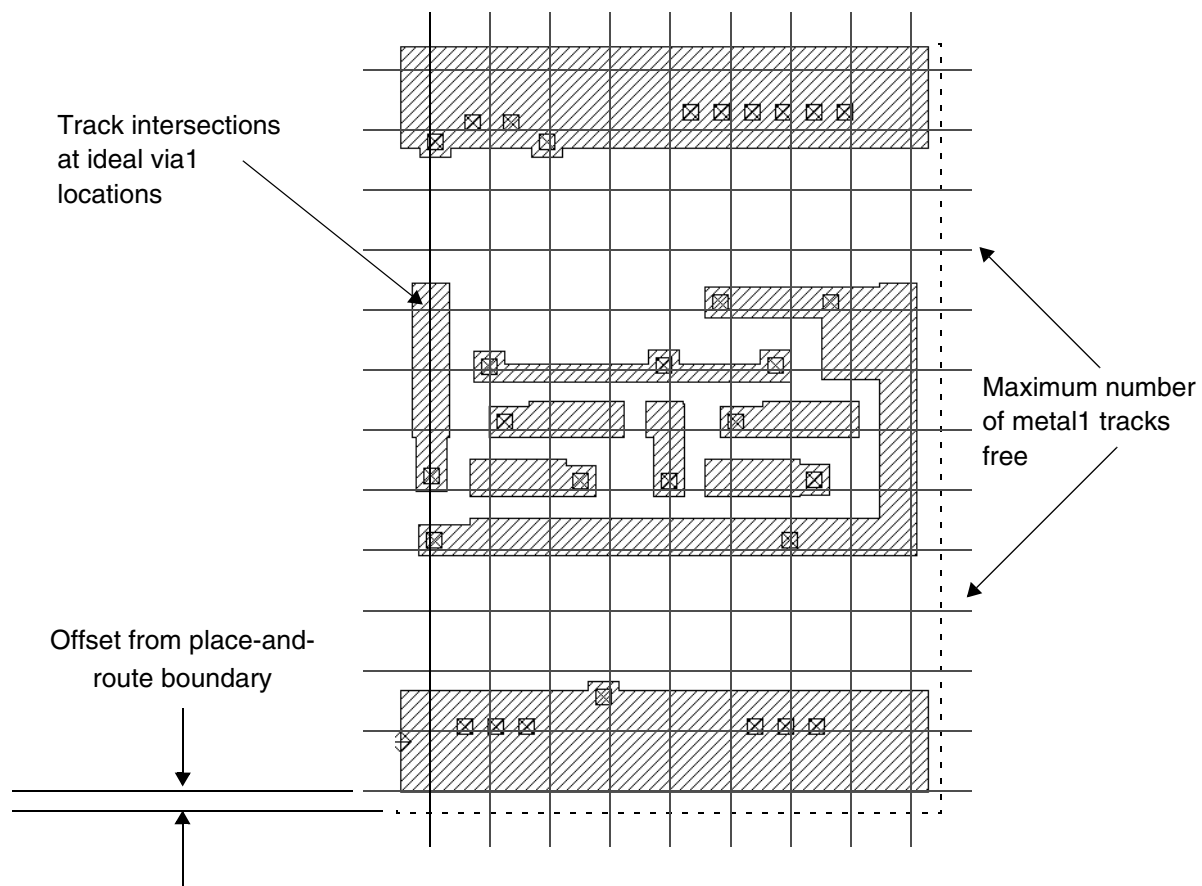
After you adjust the cell boundary and create the unit tiles, you need to create wire tracks in the unit tile. The unit tile cell must contain wire tracks, which assist the router in placing wires for each of the routing layers.

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 5-22](#) shows an example of a unit tile with the wire tracks defined.

*Figure 5-22 Unit Tile With Horizontal Wire Metal 1 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 a whole multiple of 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, as shown in [Figure 5-23](#).

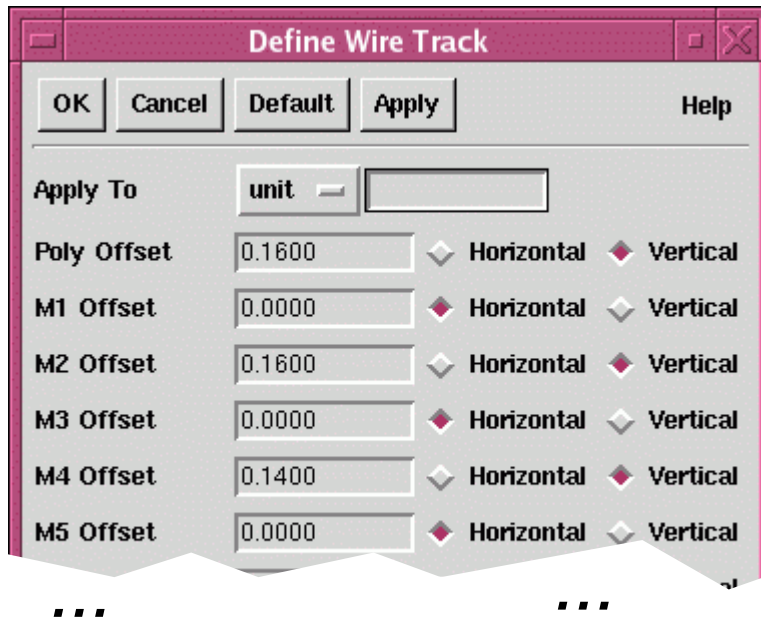
*Figure 5-23 Standard Cell Placed on Array of Unit Tiles*

---

## Define the Wire Tracks

To define the wire tracks, choose Wire Tracks > Define Unit Tile Wire Tracks in the Milkyway Environment tool. The command opens the Define Wire Track dialog box, as shown in [Figure 5-24](#).

Figure 5-24 Wire Tracks &gt; Define Unit Tile Wire Tracks Dialog Box

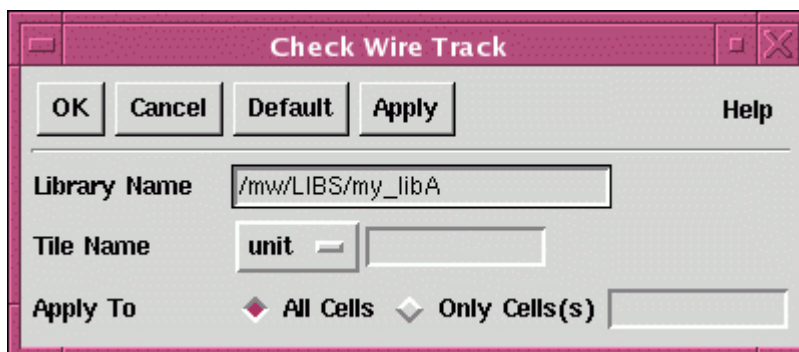


Use the option button to select the unit tile: unit, unit1, or specified. If there is more than one unit tile specified in the technology file, be sure to set the wire tracks for each one. For each layer in the unit tile, specify the offset and the preferred wire track direction. Click the OK button to complete the wire track definition.

## Check the Wire Tracks

After you define the wire tracks, you can have the pins checked for routability. Choose Wire Tracks > Check Pin on Track. This opens the Check Wire Track dialog box, as shown in [Figure 5-25](#).

Figure 5-25 Choose Wire Tracks &gt; Check Pin on Track Dialog Box



Enter the name of the Milkyway library, the tile name, and the cells to check. Then click OK. The command displays a report on the pins that do not have on-track routability in the specified cells.

---

## Define the Antenna Properties

In modern IC manufacturing technologies, MOS transistor gate oxides are easily damaged by electrostatic discharge, especially during the manufacturing process. The static charge collected on lower-level wires during multilevel metallization processing can destroy gate oxides and ruin the chip. This issue is called the “antenna problem.”

To protect the MOS device from being damaged, the area of the metal connected to a gate oxide during a given metal mask stage must be limited to a certain value. When the metal area exceeds this value, a reverse-biased diode can be used to provide a discharge path to protect the gate oxide at a cell input.

The diffusion areas of output pins provide protection for gate oxides at input pins. However, at intermediate stages of the manufacturing process, unconnected metal wires extending from the input pins can cause antenna problems to occur. For many process technologies, antenna rules specify the maximum ratio of antenna area (the metal area of dangling connections) to gate area (the gate area of input pins). This ratio can be a constant value or a value derived by an antenna equation.

Antenna rules can be stored in the Milkyway database. During library preparation, you can specify the antenna rules so that they are ready to be used in routing by the IC Compiler tool.

---

## Antenna Data Preparation for Standard Cells

For each standard cell, you can create a Cell Library Format (CLF) file containing the antenna definition. You load this file into the cell library by using the CLF > Load command in the Milkyway Environment GUI. Milkyway can store several antenna properties for every port in the cell library.

There are three basic antenna properties of a cell:

- gate size
- antenna area
- diode protection value

## Gate Size

All input pins of cells should be protected. It is the pin gate size that determines the input pin's capability to resist excessive charges. The gate size can be defined in the CLF file by using the following syntax:

```
defineGateSize "cellName" "portName" gateSize
```

For example,

```
defineGateSize "AND3B" "A" 5.4
```

If the gate size information is not complete, in the IC Compiler tool, you can set the value of the default gate size for all input pins. The syntax is

```
icc_shell> set_route_zrt_detail_options -default_gate_size real_value
```

The default setting is 0.0 and the allowed range is 0.0 to 1e+06.

## Antenna Area

By default, the pin area is included while computing the total antenna area. However, some libraries might not expose the full geometry of the pins, which can cause the antenna checker to underestimate the real antenna area. In that case, you can define the real metal area in the CLF file by using the following syntax:

```
(defineAntennaArea "cellName" "portName" `(
  ("layerName" #f area)
  ("layerName" #t area)
  .....))
```

where

*area* indicates metal area of the port in square user units

#f refers to polygon area

#t refers to sidewall area

For example,

```
(defineAntennaArea "and3x2" "B" `(
  ("metal1" #t 2.2)
  ("metal1" #f 1.1)
  ("metal2" #t 2.2)
  ("metal2" #f 2.4)
  ("contact" #f 2.1)
  ("via1" #f 1.7)
  ))
```

When the antenna area is specified for a pin, the checker uses the specified area instead of computing the pin area.

## Diode Protection Value

In advanced antenna fixing, diode protection values are required for all output pins in standard cells. Diode protection values are provided by the foundry or library vendor.

The value can be loaded from a CLF file written with the following syntax:

```
defineDiodeProtection "cellName" "portName" `(protectionValue)
```

For example,

```
defineDiodeProtection "and3x2" "Y" `(0.532)
```

When the CLF diode protection information is missing in the child library, in the IC Compiler tool, you can set the default diode protection value for all output pins. If you do not have specific diode protection settings, use the following syntax:

```
icc_shell> set_route_zrt_detail_options \
            -default_diode_protection real_value
```

Note that this setting does not apply to pins of the top-level cell. The default setting is 0.0 and the allowed range is 0.0 to 1e+06.

## Removing Diode Properties

These CLF antenna properties can be removed from a library by using the following Scheme command:

```
dbPurgeLibAntennaProperty "library_name"
```

---

## Antenna Data Preparation for Macros

Before you prepare antenna data for a macro cell, both the CEL view and FRAM view of the macro must be present in the Milkyway database.

In a hierarchical design, the gates, routing metals, and diodes in the child cells are not visible from the top cell. For a place and route macro (a soft macro), use one of the following commands to compute the antenna properties and load them into the FRAM view.

```
# For a soft macro design routed with Zroute:
icc_shell> extract_zrt_hier_antenna_property -cell_name cell_name

# For a soft macro design routed with the classic router:
Milkyway> axComputeHierAntennaProp "cell_name"
# or
icc_shell> extract_hier_antenna_property -cell_name cell_name
```

For a hard macro, use the following command to generate the antenna properties:

```
# For a hard macro design, IC Validator tool:
Milkyway> signoffHierAntenna
```

```
# or
icc_shell> signoff_calculate_hier_antenna_property -diffusion_layers ...
```

For details, see the man page for the applicable IC Compiler command.

If your design contains hard macros, to extract the antenna information for the hard macros, use the `signoff_calculate_hier_antenna_property` command.

If you are using a hierarchical flow and create a block abstraction for a block, use the `extract_zrt_hier_antenna_property` command to extract the antenna information from the block. Then you can use that information at the next level of hierarchy.

To use the IC Validator tool when running the `signoffHierAntenna` or `signoff_calculate_hier_antenna_property` command, specify the location of the licensed IC Validator executable by setting the `ICV_HOME_DIR` environment variable. You can set this variable in your `.cshrc` file.

To specify the location of the IC Validator executable, use commands similar to those shown in the following example:

```
% setenv ICV_HOME_DIR /root_dir/icv
% set path = ($path $ICV_HOME_DIR/bin/AMD.64)
```

Make sure that the version of the IC Validator executable that you specify is compatible with the IC Compiler version that you are using.

For more information about the IC Validator tool, see the IC Validator documentation, which is available on SolvNet.

After the antenna data preparation, you can use the `clfDumpAntennaProp` and `dbPurgeLibAntennaProperty` Scheme commands to dump and purge the antenna properties that are stored in the library. This is the syntax:

```
clfDumpAntennaProp (dbGetCurrentLibId) "fileName"

dbPurgeLibAntennaProperty "libName"
```

To complete an antenna check on the top-level cell, the following antenna properties for a macro must be extracted and loaded into each pin of the macro:

- Diode protection: `mX_diode_protection`, where *X* refers to metal layer numbers
- Gate sizes: `mX_gate_size`, where *X* refers to metal layer numbers
- Antenna area/ratio: `mX_modeY_area/ratio`, where *X* refers to metal layer numbers and *Y* refers to antenna area modes 1 - 6

The simplest way to write the extracted properties into the Milkyway database is through the CLF file. Use the CLF > Load command in the Milkyway Environment GUI or the `auLoadCLF` Scheme command to open the Load CLF File dialog box, read the CLF file, and store the antenna properties in the FRAM view in the Milkyway database.



The gate size, diode protection value, and antenna area ratio of a pin are defined with the `defineDiodeProtection` and `defineHierAntennaProp` CLF commands. The syntax is:

```
(defineDiodeProtection "cellName" "portName"
'(m1_diode_protection m2_diode_protection m3_diode_protection ...))
(defineHierAntennaProp "cellName" "portName" '(
  ("Metal1_layerName" m1_gate_size m1_model_area m1_mode2_ratio
    m1_mode3_area m1_mode4_area m1_mode5_ratio m1_mode6_area)
  ("Metal2_layerName" m2_gate_size m2_model_area m2_mode2_ratio
    m2_mode3_area m2_mode4_area m2_mode5_ratio m2_mode6_area )
  ...
)
)
```

For example,

```
(defineDiodeProtection "H14_BRFD8X24AR1SB" "di[17]"
'(0 0 0.2025 0.2025 0.2025 0.2025))
(defineHierAntennaProp "H14_BRFD8X24AR1SB" "di[17]" '(
  ("metal3" 0.1188 2.8322 13.2727 5.41195 11.0187 54.2489 22.419)
  ("metal4" 0.1188 0 0 5.41195 0 0 22.419)
  ("metal5" 0.1188 0 0 5.41195 0 0 22.419)
  ("metal6" 0.1188 0 0 5.41195 0 0 22.419)
))
```

The antenna area ratio modes refer to the following area ratios:

- Mode 1 (`mX_mode1_area`) refers to the area (polygon) of metal X when the connectivity is traced up to metal layer mX.
- Mode 2 (`mX_mode2_ratio`) refers to the maximum internal ratio (polygon) of metal layer (X-1) when the connectivity is traced up to metal layer mX.
- Mode 3 (`mX_mode3_area`) refers to the total metal area (polygon).
- Mode 4 (`mX_mode4_area`) refers to the area (sidewall) of metal X when the connectivity is traced up to metal layer mX.
- Mode 5 (`mX_mode5_ratio`) refers to the maximum internal ratio (sidewall) of metal layer (X-1) when the connectivity is traced up to metal layer mX.
- Mode 6 (`mX_mode6_area`) refers to the total metal area (sidewall).

For the layer below the pin layer, the hierarchical antenna properties should be set to 0.

If `mX_gate_size` is zero, such as an output pin, `mX_mode2_ratio` and `mX_mode5_ratio` must be 0.

For example,

```
defineDiodeProtection "H14_BRFD8X24AR1SB" "do[17]"
'(0 0 0.8856 0.8856 0.8856 0.8856)
(defineHierAntennaProp "H14_BRFD8X24AR1SB" "do[17]" '(
```

```
( "metal3" 0 5.0834 0 7.18415 19.5411 0 28.779)
( "metal4" 0 0 0 7.18415 0 0 28.779)
( "metal5" 0 0 0 7.18415 0 0 28.779)
( "metal6" 0 0 0 7.18415 0 0 28.779)
))
```

Because do[17] is a metal3 output pin, the gate size is 0 and the hierarchical antenna properties for metal1 and metal2 are ignored. Moreover, all mode2 and mode5 antenna properties for metal3, metal4, and metal5 are 0.

Zroute requires `defineHierAntennaProp` to be defined for all layers. Otherwise, it treats the data as incomplete, skips antenna analysis, and issues a ZRT-311 warning message. If you get this error message, see [SolvNet article 027178](#), “Debugging the ZRT-311 Message .”

---

## Antenna-Fixing Diode Cells

If a cell will be used as a diode to fix antenna violations, the diode port must be identified as such in the FRAM view. To do so, use the following Scheme commands:

```
define libId (dbGetCurrentLibId)
dbConvertPortToDiodePort libId "cell_name" "port_name"
```

where *cell\_name* is the cell name (without the .FRAM extension) and *port\_name* is the name of the diode connection port. This converts the ordinary port into a diode port.

To confirm that the diode port has been properly identified, you can query it as follows:

1. Open the FRAM view of the diode cell.
2. Click the Query-object button.
3. Click on the pin of the diode port.

The query result should look like the following example:

```
OBJECT TYPE : DIODE-PIN [ID #x1000]
PIN NAME    : DA1
LAYER       : metall (46)
...
```

The `OBJECT TYPE` is identified as `DIODE-PIN` rather than simply `PIN`.

---

## Import Pin Attributes From .db Files

The .db logic library associated with a Milkyway library might contain information relevant physical library such as signal pin direction, cell bias or secondary PG pin information, bus pin information, and analog pin information.

To update a Milkyway library with information contained in a .db library, use the `update_mw_port_by_db` Tcl command in the Milkyway Environment tool. For example,

```
Milkyway> update_mw_port_by_db -mw_lib my_mw_lib \
      -db_file {a1.db b1.db} -bias_pg
```

This command updates all FRAM cells in one specified Milkyway library with information from the specified list of .db files. The command options let you specify which types of attributes are updated. By default, the command updates only the signal pin direction and secondary power/ground pin information in FRAM cells.

For more information about the types of information updated in the Milkyway library and the data type options, see the man page for the `update_mw_port_by_db` command.

---

## Create Cell Properties

You can optionally create your own cell properties, set the property values, and query those values in the Milkyway Environment tool, using Scheme or Tcl commands.

---

### Creating a Cell Property in Scheme Mode

To create a cell property in Scheme mode, use the following procedure:

1. Open the Milkyway library containing the cell:  
Library > Open (Browse to library name, click OK)
2. Open the cell to receive the new property:  
Cell > Open (Browse to cell name, click OK)
3. Create the cell property and set its value:

```
dbCreateCellProperty (geGetEditCell) "property_name" value
```

For example,

```
dbCreateCellProperty (geGetEditCell) "myProp1" "L1R0"
dbCreateCellProperty (geGetEditCell) "myProp2" 32.4
```

The value can be a string, integer, floating-point number, or Boolean expression.

Note:

The `geGetEditCell` command returns the cell identification number of the currently open cell. You can optionally perform Step 3 using two commands, for example:

```
geGetEditCell
4
dbCreateCellProperty 4 "myProp1" L1R0
```

4. To query the property value, use the “fetch” command:

```
dbFetchCellPropertyValueByName (geGetEditCell) property_name
```

5. To change the value of the property, use the “update” command:

```
dbUpdateCellProperty (geGetEditCell) property_name value
```

6. Save the edited cell:

```
Cell > Save or
```

```
Cell > Save As
```

---

## Creating a Cell Property in Tcl Mode

To create a cell property in Tcl mode, use the following procedure:

1. Open the Milkyway library containing the cell:

```
Milkyway> open_mw_lib mw_lib_name
```

2. Open the cell to receive the new property:

```
Milkyway> open_mw_cel cell_name
```

3. Create the cell property and set its value:

```
Milkyway> dbCreateCellProperty [geGetEditCell] property_name \
property_type value
```

For example,

```
Milkyway> dbCreateCellProperty [geGetEditCell] myProp1 string L1R0
Milkyway> dbCreateCellProperty [geGetEditCell] myProp2 double 32.4
```

The property type can be string, integer, float, double, or boolean.

Note:

The `geGetEditCell` command returns the cell identification number of the currently open cell. You can optionally perform Step 3 using two commands, for example:

```
Milkyway> geGetEditCell
4
Milkyway> dbCreateCellProperty 4 myProp1 string L1R0
```

4. To query the property value, use the “fetch” command:

```
Milkyway> dbFetchCellPropertyValueByName [geGetEditCell] property_name
```

5. To change the value of the property, use the “update” command:

```
Milkyway> dbUpdateCellProperty [geGetEditCell] property_name value
```

6. Save the edited cell:

```
Milkyway> save_mw_cel
```



# 6

## Library Checking

---

Advanced IC design relies on high-quality libraries. Ensuring that the logic and physical libraries are correct and consistent is an essential step before design creation. You should check any new physical or logic library before you use it by running the `check_library` command in the IC Compiler tool, as described in the following sections:

- [Library Checking Overview](#)
- [Validating Logic Libraries](#)
- [Validating Physical Libraries](#)
- [Library Check Reporting Examples](#)

---

## Library Checking Overview

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

The Milkyway Environment tool also has a `check_library` command that can perform some of the same types of library checking as the command in the IC Compiler tool. However, the command options, command option syntax, default behavior, and setup options are different between the two tools. For information about the Milkyway Environment command, see the man page for the command in the Milkyway Environment tool in Tcl mode or on SolvNet.

Library Compiler also has a `check_library` command, which is very similar to the command in the IC Compiler tool. For more information about the command in Library Compiler, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*, available in the Library Compiler documentation set on SolvNet.

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

```
icc_shell> check_library
```

By default, the `check_library` command performs consistency checking 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 check 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]
```

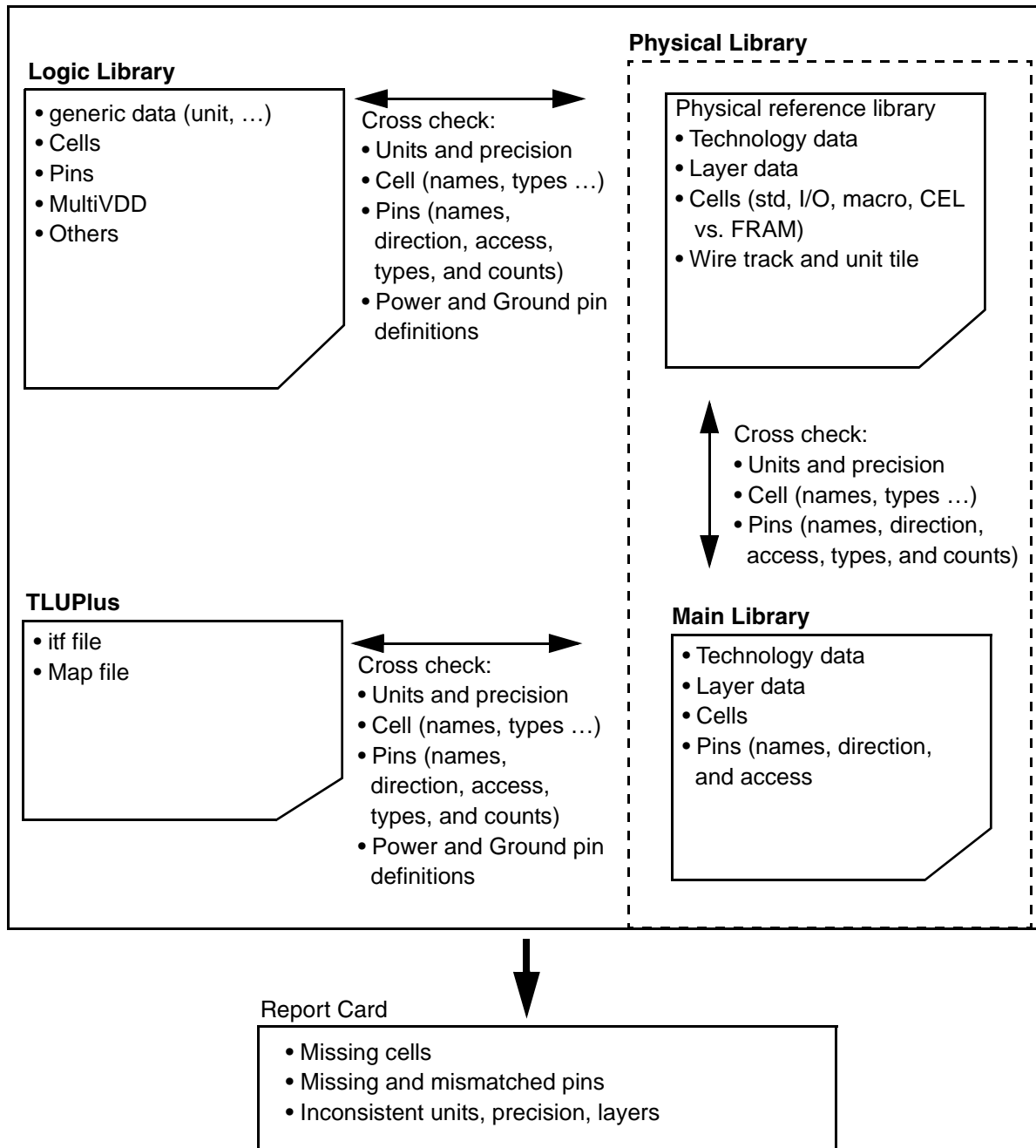
The libraries that you explicitly specify in the command override the default libraries. The `-logic_library_name` option specifies a list of logic libraries to be checked for consistency. If the `-logic_library_name` option is not explicitly specified but the `set_min_library` `max_library` command 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 types of library checking performed by the `check_library` in the IC Compiler tool.

Figure 6-1 Library Checking Overview



By default, the `check_library` command verifies that all cells in the logic library and in 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 by using the `set_check_library_options` command. For example, you can explicitly specify the following types of library consistency checking:

- To verify that the area for each cell (other than pad cells) is consistent between the logic library and the physical library.

```
icc_shell> set_check_library_options -cell_area
```

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

```
icc_shell> set_check_library_options -cell_footprint
```

- To 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. For a full description of the options of the `set_check_library_options` command, see the man page for the command.

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 the `check_library` command reports any inconsistencies, you must fix them before you begin using the libraries.

## 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.

[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
<code>-compare {construct attribute value}</code>	General logic library consistency checking.
<code>-mcm</code>	Logic library consistency checking for multivoltage-multimode (MCM) operation.
<code>-scaling {timing noise power}</code>	Logic library consistency checking for CCS timing, CCS noise, and CCS or NLDM power scaling.
<code>-tolerance {type relative_tolerance absolute_tolerance}</code>	The relative tolerance and an absolute tolerance for characterization value comparison.
<code>-upf</code>	Logic library consistency checking for multivoltage and UPF.
<code>-validate</code>	Logic library consistency checking for library characterization.
<code>-logical</code>	Enables all the logic library checks.
<code>-reset</code>	Resets the options to the default.
<code>-all</code>	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, library type, and whether the library is power and ground pin based. The command also checks to see if the libraries have the same units and the same slew trip points for CCS timing scaling. The command checks to see if the libraries have default operating conditions 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`, `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 to see if the libraries have the same number of pins, including PG pins. The command also checks to see 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 to see if the two libraries are the same in the following ways:

- The `input_signal_level` or `output_signal_level` attribute for a rail-based library is the same, or the `related_power_pin` or `related_ground_pin` attribute is the same for a `pg_pin`-based library.
- Derived `pin_number` attribute values are the same. If there is a mismatch, the command performs additional checking in an attempt to determine the cause.

The `check_library` command performs these checks by default.

### Timing Arcs

The `check_library` command checks for timing arcs that have related pins, timing type, timing sense, and `when` conditions that match 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 check for CCS timing scaling across the different libraries. This command checks for matching numbers of driver and load index values, consistent usage of receiver models, consistent waveform usage for characterization, and consistent usage of model compression.

Use the `set_check_library_options -scaling noise` command to check for CCS noise scaling, including consistent voltage index values, ordering of conditional “when” pin models, and number of noise models.

Use the `set_check_library_options -scaling power` command to check for CCS or NLPM power scaling to ensure that the libraries have the consistent modeling groups, tables, and attributes.

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 check for multivoltage or UPF flows. The command then verifies that different models exist for different characterizations, different voltages are characterized, power devices have consistent and complete attributes, and power data is provided for all operating conditions.

Use the `set_check_library_options -mcm` command to perform checking for a multicorner-multimode flow. The command then verifies that different characterization information exists for different operating conditions, cells have consistent attributes, and the power and power-down information exists and is consistent.

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 *type* are `delay`, `slew`, `constraint`, `slew_index`, `load_index`, `time`, `power`, `current`, `energy` and `capacitance`. If you 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, you can set the tolerances to smaller values.

Use the `set_check_library_options -validate {timing}` command to enable the following checks for library characterization:

- 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.

---

## Library Checking Report Format

The `check_library` command performs checks on physical and logic 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, which is compatible with most spreadsheet programs.

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.

---

## 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_library_name mw_lib_A
```

To set the physical library checks, run the `set_check_library_options` 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
<code>-routeability</code>	The on-track accessibility of the physical pins.
<code>-drc</code>	Design rule constraint (DRC) violations in the routing (FRAM) view of the cells. This requires write permission for the library directory.
<code>-view_comparison</code>	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.
<code>-antenna</code>	Missing antenna properties for cells and missing antenna rules for routing layers.
<code>-signal_em</code>	Missing signal electromigration rules for routing layers.

Table 6-2 Physical Library Checks (Continued)

Use this option	To check for
<code>-same_name_cell</code>	Cells with identical names in different reference libraries. The names of the cells are reported.
<code>-rectilinear_cell</code>	Coordinates of cells with rectangular or rectilinear boundaries and the boundaries for macro cells.
<code>-phys_property {types}</code>	Physical properties. You can specify one or more of the following types: place, route, cell, and metal_density.
<code>-physical_only_cell</code>	Physical-only cells, such as filler cells with and without metal, diode cells with antenna properties, and corner cells.
<code>-rail view_data</code>	The power and ground (PG) rail data readiness in the CONN view and the data consistency with the FRAM view. The CONN view contains PG rail information for the PrimeRail tool.
<code>-tech_consistency</code>	Consistency in the technology data between the design library and the reference libraries.
<code>-tech</code>	The quality of the technology data for a reference library.
<code>-cells</code>	Specifies a list of cell names. If not specified, all cells in the library are checked.
<code>-all</code>	All options.
<code>-reset</code>	Resets all options to the default values.

To see the physical library checks that are currently enabled, run the `report_check_library_options -physical` command.

### 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 poor accessibility. For a report example, see [“Report for the -routeability Option” on page 6-14](#).

#### `-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. None of the cell content, such as pins, is checked for mismatch. For a report example, see [“Report for the -view\\_comparison Option” on page 6-14](#).

#### `-antenna`

Checks for a missing antenna property for cells and checks antenna rules in the layers. If an input pin is missing the gate size or an output pin is missing diode protection, it is counted as missing the property. If a macro 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 the -antenna Option” on page 6-15](#).

#### `-signal_em`

Checks for signal electromigration rule (current model and model type) for each routing layer. For a report example, see [“Report for the -signal\\_em Option” on page 6-15](#).

#### `-same_name_cell`

Checks for cells with identical names among different reference libraries linked to the specified main 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 the -same\\_name\\_cell Option” on page 6-15](#).

#### `-rectilinear_cell`

Checks for cells with rectilinear boundaries. For a report example, see [“Report for the -rectilinear\\_cell Option” on page 6-16](#).

#### `-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. For macros, it reports cell boundary and height.

`-phys_property {route}` checks and reports routing properties for each routing layer. For a report example, see [“Report for the -phys\\_property Option” on page 6-16](#).

`-physical_only_cell`

Checks for physical-only cells (filler cells with and without metal, diode cells with antenna properties, and corner cells). For a report example, see [“Report for the -physical\\_only\\_cell Option” on page 6-16](#).

`-rail view_data`

Checks the power and ground (PG) rail data readiness in the CONN view and the data consistency with the FRAM view. The CONN view contains PG rail information for the PrimeRail tool. This check includes the port types and associated net names in the CONN view and matching ports and net names in the FRAM view.

`-tech_consistency`

Checks technology data for 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 the -tech\\_consistency Option” on page 6-17](#).

`-tech`

Checks 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 for DRC violations in the FRAM views of cells in the specified library. It 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 specified with any other options, this option is ignored.

`-all`

Performs all checks described previously.

`-reset`

Resets the options to the default values.

---

## Verifying the Electromigration Constraints

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

```
icc_shell> set_check_library_options -signal_em
1
icc_shell> 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 tool, to load the signal electromigration constraints in .plib format into the design library, use a command similar to this:

```
Milkyway> read_plib -lib_name my_designlib \
             -tech_plib_files signal_em_constraints.plib \
             -overwrite_existing_tech
```

If the electromigration constraints are in Advanced Library Format (ALF), you can read them into the Milkyway library using an IC Compiler command similar to this:

```
icc_shell> set_mw_technology_file -alf my_alf_file.alf my_designlib
```

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 three 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 also 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 library with the following IC Compiler command:

```
icc_shell> set_mw_technology_file \
          -plib signal_em_constraints.plib my_designlib
```

---

## Library Check Reporting Examples

After checking a library, the IC Compiler tool generates a report in table format. To facilitate reading, each option check report starts with a `#BEGIN_CHECK_ITEM` comment line and concludes with an `#END_CHECK_ITEM` comment line.

You can use the `report_check_library_options` command to report the values of the options set by the `set_check_library_options` command. For details, see the man pages for these commands.

The following `check_library` reports for different `set_check_library_options` settings are provided as examples. For more detailed information on a given check option report, see the man page for the `set_check_library_options` command.

### Report for the -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 the -view\_comparison 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 the -antenna Option**

#BEGIN\_CHECK\_ANTENNA

List of antenna rules

Layer Name	Mode	Diode mode	Default metal ratio	Default 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 the -signal\_em Option**

Warning: Signal EM rules are missing in the library.

#BEGIN\_CHECK\_SIGNALLEM

List of signal EM data

Layer name	Current model	Model type
METAL1	peak	table
METAL1	rms	table
METAL1	static	table

#END\_CHECK\_SIGNALLEM

**Report for the -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 the -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 the -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

Layer	Preferred direction	Track direction	Offset	Pitch	Remarks
METAL1	H	H	0.280	0.560	OK
METAL2	V	V	0.330	0.660	OK

**Report for the -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 the -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
```





# 7

## Using Milkyway Libraries in IC Compiler

---

IC Compiler uses the Milkyway database as the storage format for physical library cells and physical chip layout information, including placement and routing results. The entire chip is a single cell built out of lower-level blocks, which are also cells. These blocks are built out of smaller blocks, and so on, down to the level of leaf-level cells. The physical representations of these cells are stored in Milkyway libraries.

Usage of Milkyway libraries in the IC Compiler tool is described in the following sections:

- [Milkyway Library Structure](#)
- [Milkyway Reference Libraries](#)
- [Opening and Closing Milkyway Libraries](#)
- [Opening and Closing Milkyway Cells](#)
- [Creating New Milkyway Libraries and Cells](#)
- [Reading and Writing GDSII and OASIS in the IC Compiler Tool](#)
- [Milkyway Database Versions](#)

---

## Milkyway Library Structure

The Milkyway database contains the physical library information needed by the IC Compiler tool. The database contains not only leaf-level physical cell information and technology information, but also design-specific physical information such as the design placement and routing results.

The Milkyway database is organized as a hierarchy of data files. However, you should not attempt to modify these files directly using operating system commands. Instead, you should always access the database by using the IC Compiler tool or another Synopsys tool and use the tool commands to modify the database contents. This will ensure the consistency and integrity of the database.

A Milkyway library that you open for editing is called a design library. You can open no more than one design library at a time. A library can contain any number of cells. You can open and display multiple cells at the same time from one library. A cell can be built using instances of cells from other libraries that are not currently open. These other libraries are called reference libraries.

In the IC Compiler tool, you can open a Milkyway design library by using the `open_mw_lib` command or the File > Open Library command in the GUI window. You can open a cell in that library for editing by using the `open_mw_cell` command or the File > Open Design command in the GUI window.

For information about the data and view types contained in Milkyway libraries, see [“Physical Libraries: Milkyway Database” on page 1-2](#).

---

## Milkyway Reference Libraries

A design is typically built as a hierarchy of cells. The entire chip is a single cell built out of lower-level blocks, which are also cells. These blocks are built out of smaller blocks, and so on, down to the level of leaf-level cells. The physical representations of these cells are stored in Milkyway libraries.

When you open a Milkyway library, you can have both read and write access to the cells in that library. The open library containing the design you are working on is called the design library. The lower-level cells in the design can be found in either the design library that is currently open or in Milkyway libraries that are not open. The other Milkyway libraries containing lower-level cells are called reference libraries. The design you are working on contains instances of cells that are defined in those libraries.

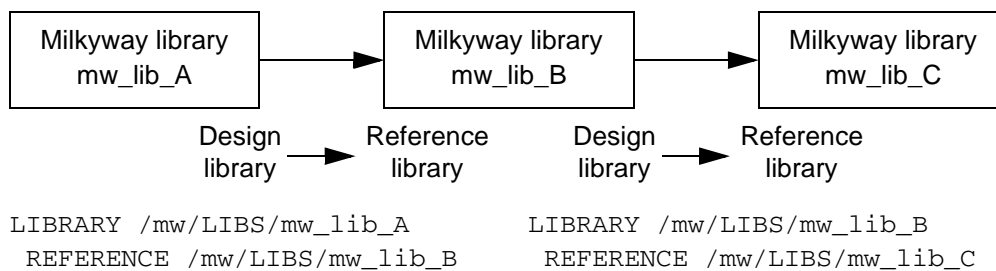
A reference library typically contains data that you do not modify yourself. For example, a reference library might contain standard cells provided by an ASIC vendor, intellectual property provided by an IP block seller, or lower-level blocks designed by another group at

your company. Design information that you create or modify yourself must be stored in a design library.

The terms “design library” and “reference library” describe a one-way relationship between two Milkyway libraries. If you define mw\_lib\_B to be a reference library of design library mw\_lib\_A, then cells contained in mw\_lib\_B can be used to build cells contained in mw\_lib\_A. However, this does not imply that cells contained in mw\_lib\_A can be used to build cells in mw\_lib\_B.

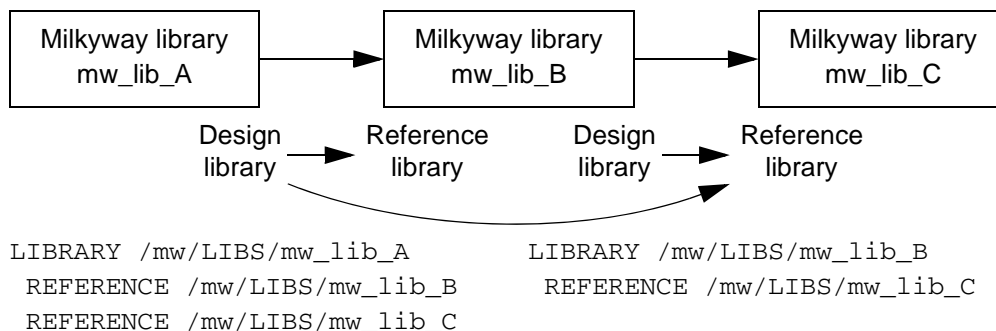
A given Milkyway library can serve as both a design library and a reference library. For example, you can define mw\_lib\_B to be a reference library of mw\_lib\_A and define mw\_lib\_C to be a reference library of mw\_lib\_B. Then cells in mw\_lib\_C can be used to build designs in mw\_lib\_B, and cells in mw\_lib\_B can be used to build cells in mw\_lib\_A. Thus, mw\_lib\_B is a reference library of mw\_lib\_A and also a design library to mw\_lib\_C, as shown in [Figure 7-1](#).

Figure 7-1 Design and Reference Library Relationships Example 1



Each reference library definition is one level deep. In [Figure 7-1](#), cells in mw\_lib\_C cannot be used *directly* in mw\_lib\_A because the relationship is two levels deep. However, library mw\_lib\_C can also be explicitly defined as a reference library of mw\_lib\_A, which would allow its cells to be used directly in mw\_lib\_A. See [Figure 7-2](#).

Figure 7-2 Design and Reference Library Relationships Example 2



In the IC Compiler tool, the Milkyway library that is currently open is the design library. The associated Milkyway libraries containing read-accessible cells are the reference libraries. You define the association between design and reference libraries with the `set_mw_lib_reference` command.

### See Also

- [Setting Milkyway Reference Libraries](#)
- [Milkyway Reference Control File](#)
- [Listing All Cells and Their Physical Views](#)

---

## Setting Milkyway Reference Libraries

In the IC Compiler tool, the `set_mw_lib_reference` command defines the Milkyway reference libraries associated with a specified Milkyway design library. To use this command, the Milkyway design library must be *closed*. If the design library is currently open, close it first with the `close_mw_lib` command.

You can specify the list of reference libraries explicitly in the command itself or in a separate text file called the reference control file. You must specify relative or absolute paths to the top-level Milkyway directory names.

For example, to define the Milkyway libraries `mw_lib_B` and `mw_lib_C` to be reference libraries of design library `mw_lib_A`, first close `mw_lib_A` if it is open. Then use the following command:

```
icc_shell> set_mw_lib_reference \
           -mw_reference_library {/mw/LIBS/mw_lib_B /mw/LIBS/mw_lib_C} \
           /mw/LIBS/mw_lib_A
```

To define the reference libraries using an external file instead of an explicit list in the command, use the following command:

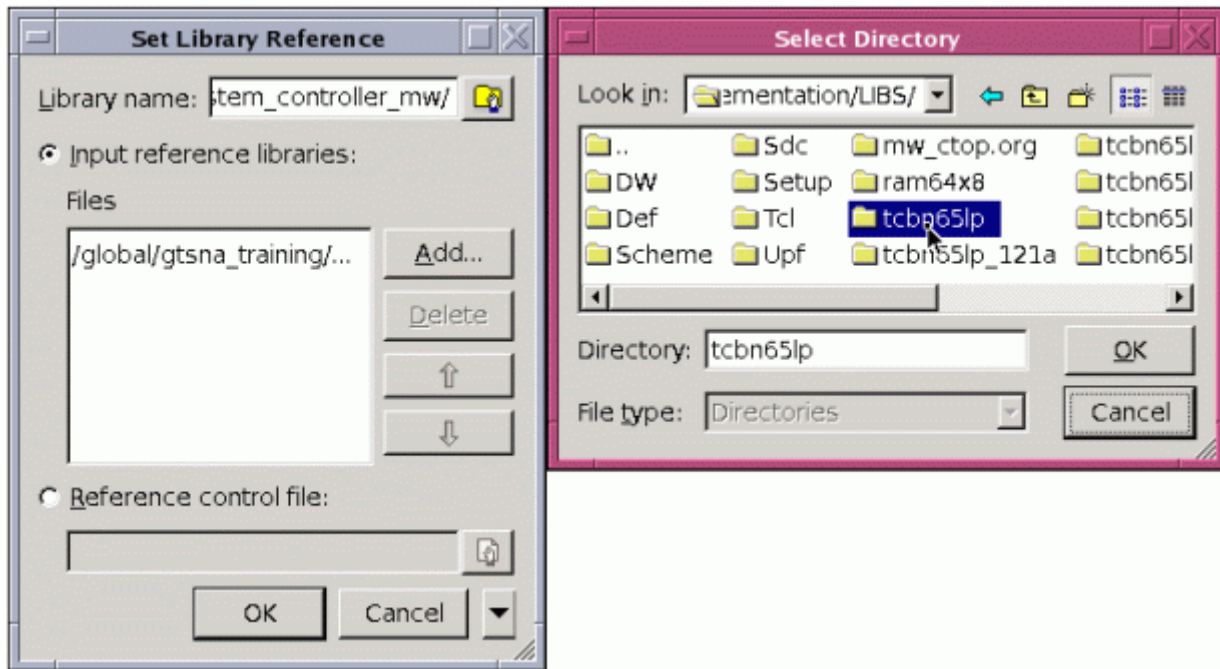
```
icc_shell> set_mw_lib_reference \
           -reference_control_file my_refs_A /mw/LIBS/mw_lib_A
```

You specify an absolute or relative path to the reference control file as well as to the Milkyway design library. The file format is described in the next section, “[Milkyway Reference Control File](#).”

You can also use the IC Compiler main GUI window to specify the reference libraries. If a Milkyway library is currently open, close it using File > Close Library. Then choose File > Set Library Reference. The Set Library Reference dialog box has a browse button that lets you

browse for the directory paths to design library and reference libraries, as shown in [Figure 7-3](#).

*Figure 7-3 File > Set Library Reference and Select Directory Dialog Boxes*



To generate a report showing a list of the reference libraries defined for a particular design library, use the `report_mw_lib` with the `-mw_reference_library` option. For example,

```
icc_shell> report_mw_lib -mw_reference_library mw_lib_A
/mw/LIBS/mw_lib_B
/mw/LIBS/mw_lib_C
```

If you do not specify which Milkyway design library to report, the currently open Milkyway library is reported.

---

## Milkyway Reference Control File

You can specify the Milkyway reference libraries associated with a Milkyway design library by using an ASCII text file called a reference control file. The `set_mw_lib_reference` command invokes this file when you use the `-reference_control_file` option.

This is the syntax of the reference control file:

```
LIBRARY path_to_mw_design_library_directory
REFERENCE path_to_mw_reference_library_directory
REFERENCE path_to_mw_reference_library_directory
REFERENCE path_to_mw_reference_library_directory
...
```

For example,

```
LIBRARY /mw/LIBS/mw_lib_A
REFERENCE /mw/LIBS/mw_lib_B
REFERENCE /mw/LIBS/mw_lib_C
```

In this example, `mw_lib_B` and `mw_lib_C` are reference libraries of design library `mw_lib_A`. Cells contained in the reference libraries can be used as instances to build cells contained in the design library.

Specify absolute, not relative, paths to the Milkyway design and reference libraries.

To have the IC Compiler tool generate a reference control file automatically from an existing Milkyway design library that already has reference libraries defined, use the `write_mw_lib_files` command with the `-reference_control_file` option. For example, to generate a reference control file called `my_refs_A` for the library `mw_lib_A`, use the following command:

```
icc_shell> write_mw_lib_files -reference_control_file \
            -output my_refs_A /mw/LIBS/mw_lib_A
```

This can be done with the `mw_lib_A` library either open or closed. However, you must specify the absolute or relative path to the design library.

---

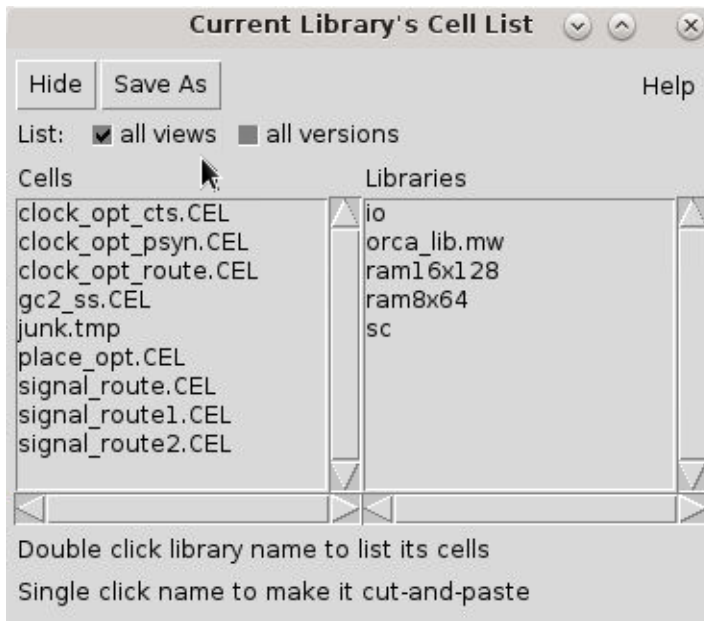
## Listing All Cells and Their Physical Views

You can list all the cells in a Milkyway reference library with their physical views in the Milkyway Environment GUI. Open the Milkyway reference library and view the information inside it as follows:

1. Choose Library > Open in the menu. The Open Library dialog box appears.
2. Enter a library name or browse to a library, and click OK. The library is loaded into memory.

3. Choose Library > Show Cells. The Current Library's Cell List dialog box appears, listing all the cells in the library.
4. Click the “all views” option to list the views for each cell in the library.

Figure 7-4 The “all views” Option Lists the Views for Each Cell



## Opening and Closing Milkyway Libraries

In the IC Compiler tool, you need to open a Milkyway design library before you can read in a cell for editing and save the edited cell. You open a Milkyway design library with the `open_mw_lib` command. For example,

```
icc_shell> open_mw_lib /mw/LIBS/mw_lib_A
```

Specify an absolute or relative path to the Milkyway design library directory.

By default, the specified library is opened with read/write access, and any associated reference libraries are available for read access only.

To open the Milkyway design library for read access only, use the `-readonly` option:

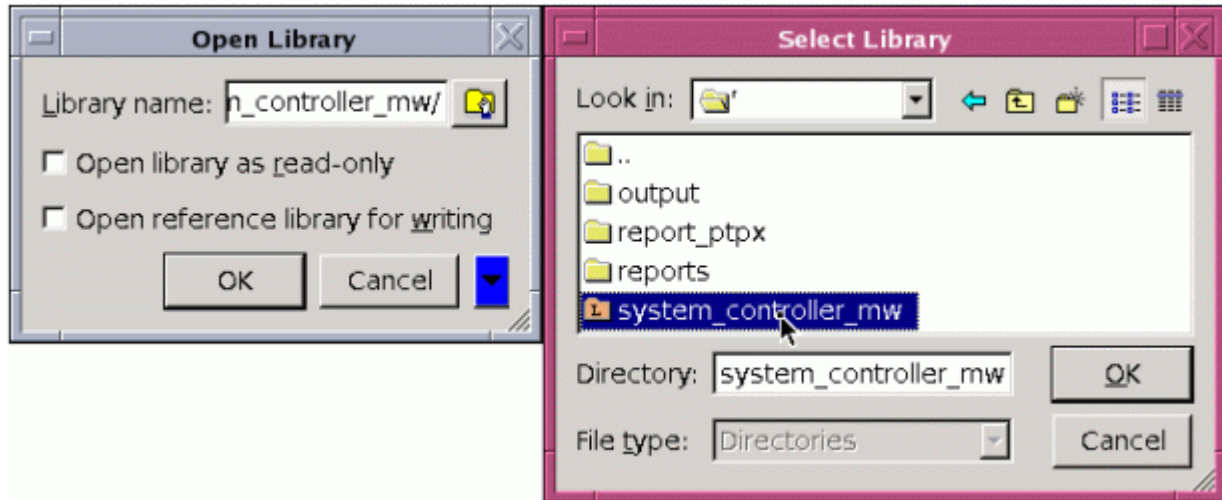
```
icc_shell> open_mw_lib -readonly /mw/LIBS/mw_lib_A
```

To open the Milkyway design library for read/write access and make the associated reference libraries also available with read/write access, use the `-write_ref` option:

```
icc_shell> open_mw_lib -write_ref /mw/LIBS/mw_lib_A
```

You can also use the IC Compiler main GUI window to open the Milkyway design library. Choose File > Open Library. The Open Library dialog box has check boxes for selecting the read/write access options and a browse button that lets you browse for the directory path to the design library. See [Figure 7-5](#).

Figure 7-5 File > Open and Select Library Dialog Boxes



To find out the name of the currently open Milkyway design library, use the `current_mw_lib` command.

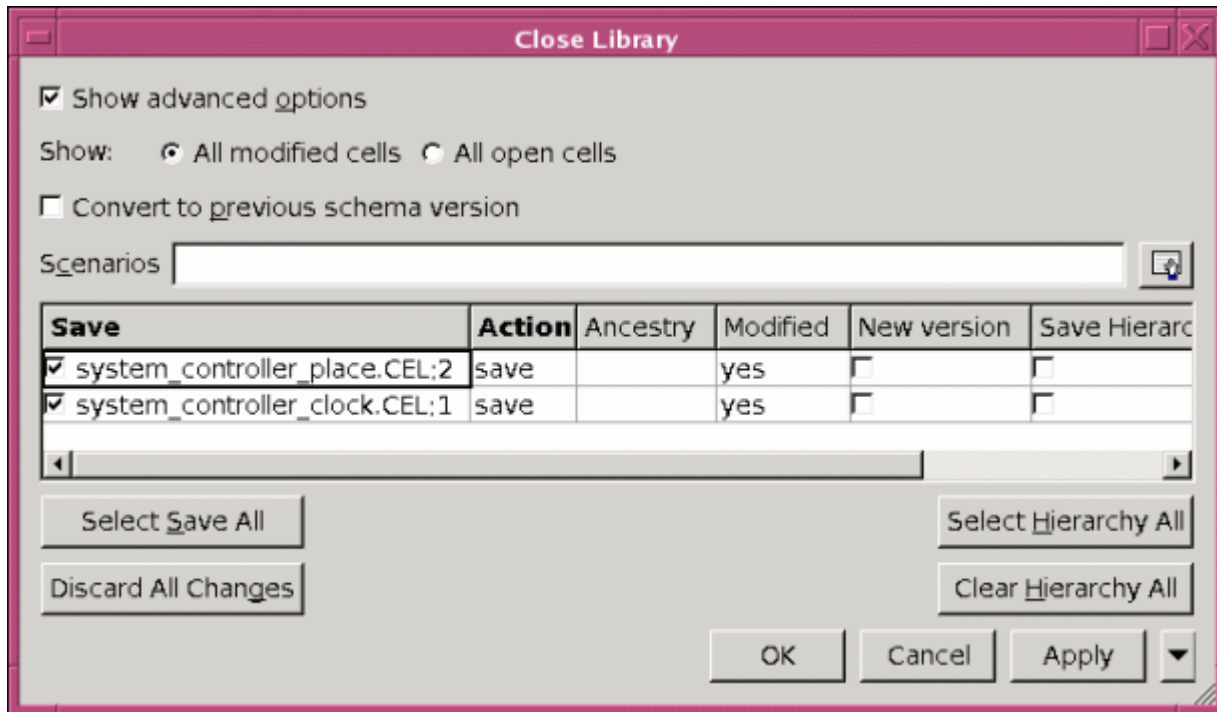
To close an open Milkyway design library, use the `close_mw_lib` command. Before using this command, you should save any edited cells. By default, this command closes the library and discards any changes made to cells in the library. To ensure that all edited cells are saved, you can use the `-save` option:

```
icc_shell> close_mw_lib -save
```

Alternatively, you can use the IC Compiler main window GUI to close the library. Choose File > Close Library, and the Close Library dialog box prompts you to either save or discard any open cells. It also offers some advanced options for saving open cells and lower-level hierarchical cells, as shown in [Figure 7-6](#).



Figure 7-6 File &gt; Close Library Dialog Box Advanced Options



For more information about the available options for closing a Milkyway design library, see the man page for the `close_mw_lib` command.

## Opening and Closing Milkyway Cells

Opening a cell displays a graphical view of the cell layout in a GUI window and makes the cell available for editing. You open a cell with the `open_mw_cel` command. You can open any number of cells contained in the Milkyway design library that is currently open.

For example, to open the cell called `system_controller_clock` in the currently open Milkyway design library, use the following command:

```
icc_shell> open_mw_cel system_controller_clock
```

By default, the cell is opened with read/write access. To open a cell for read access only, use the `-readonly` option:

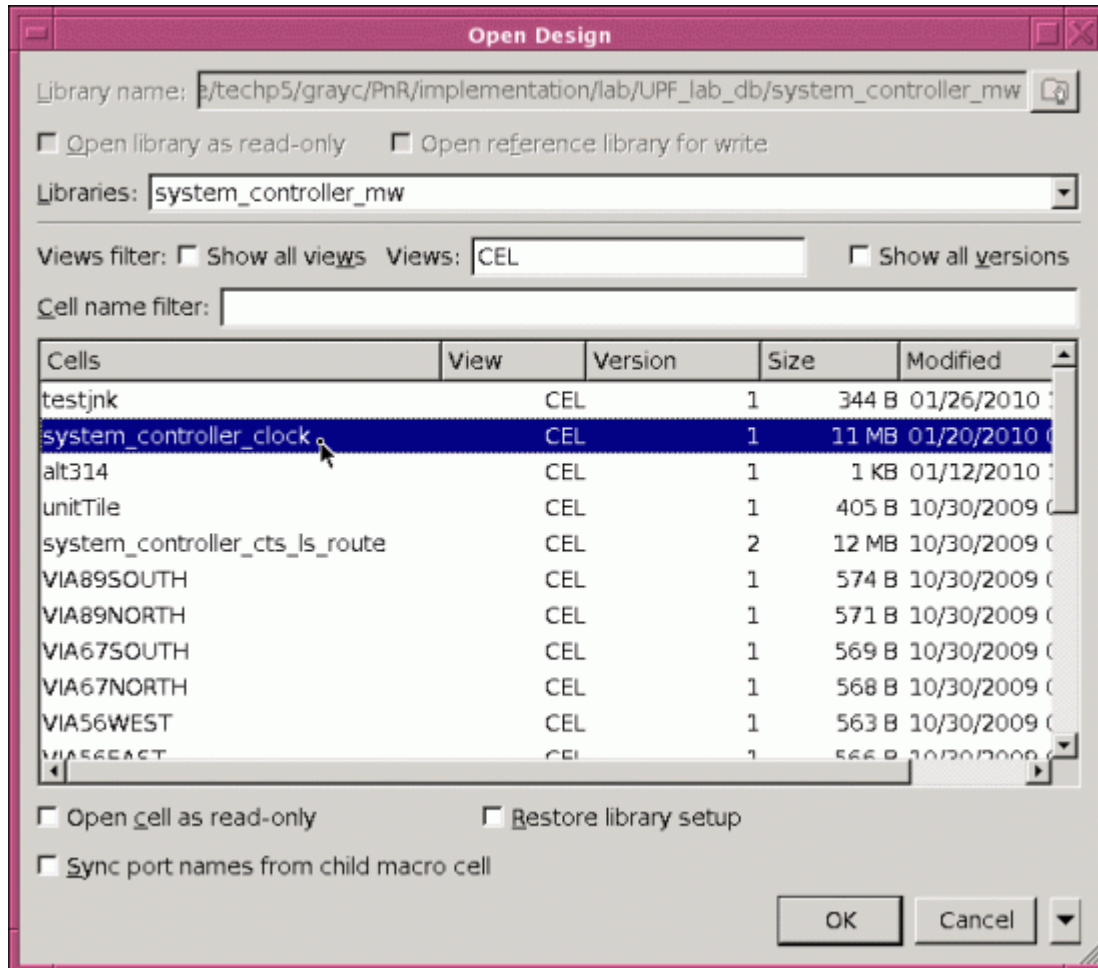
```
icc_shell> open_mw_cel -readonly system_controller_clock
```

If the Milkyway design library containing the cell is not already open, you can open both the library and cell at the same time with a single command:

```
icc_shell> open_mw_cel -library system_controller_mw \
            system_controller_clock
```

You can also use the IC Compiler main GUI window or any design window to open a cell. Choose File > Open Design. The Open Design dialog box lets you choose the Milkyway design library if a library is not already open, and lets you choose from a list of cells in the selected library, as shown in [Figure 7-7](#).

Figure 7-7 File > Open Design Dialog Box

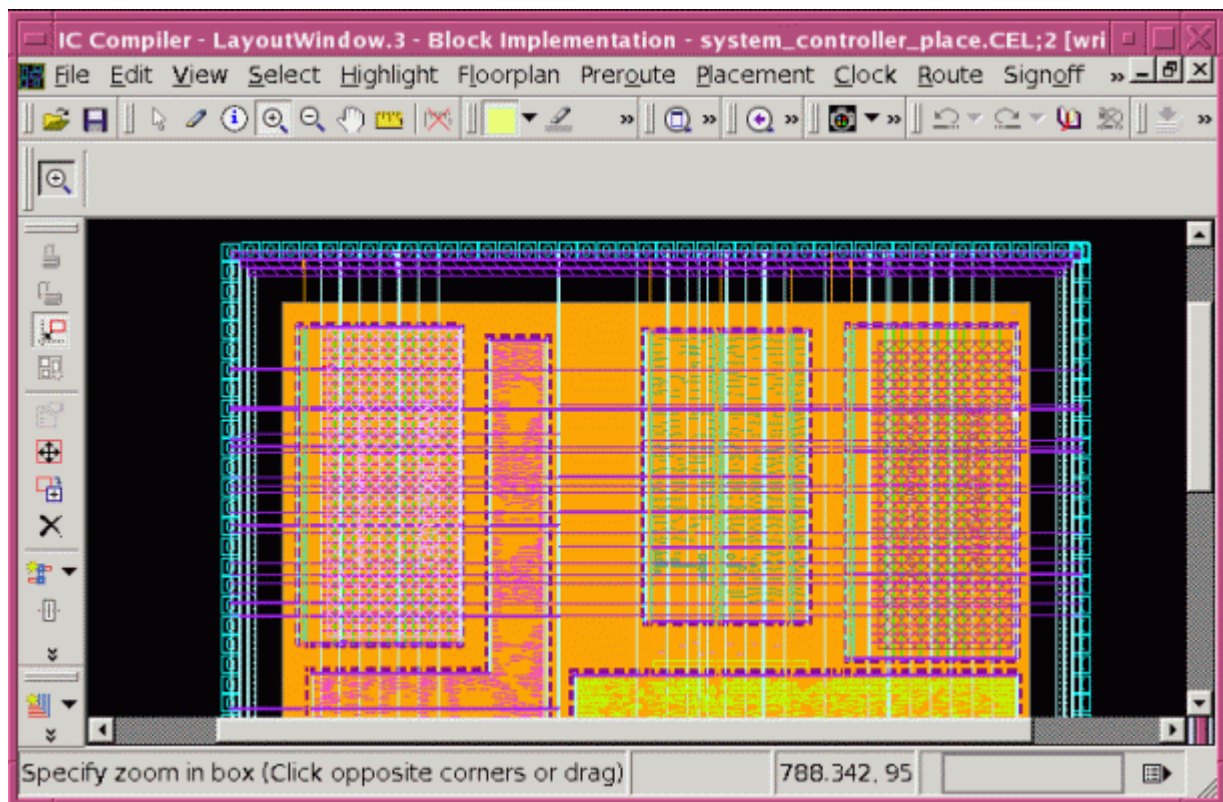


For more information about the available options for opening a design, see the man page for the `open_mw_cel` command.

To find out the name of the cell currently open for editing, use the `current_mw_cel` command or look at the title bar of the cell layout window.

You can open multiple cells contained in the currently open Milkyway library. Each open cell is displayed in its own GUI window like the one shown in [Figure 7-8](#).

Figure 7-8 Cell Layout Window Example



One of the open cells is considered the current cell for editing. To find out which cell is the current cell, use the `current_mw_cel` command. For example,

```
icc_shell> current_mw_cel
{system_controller_clock}
```

To change the current cell, simply click in the GUI window containing the cell you want to edit, bringing that window to the foreground. Alternatively, you can use the `current_mw_cel` command to specify the desired current cell:

```
icc_shell> current_mw_cel system_controller_route
{system_controller_route}
```

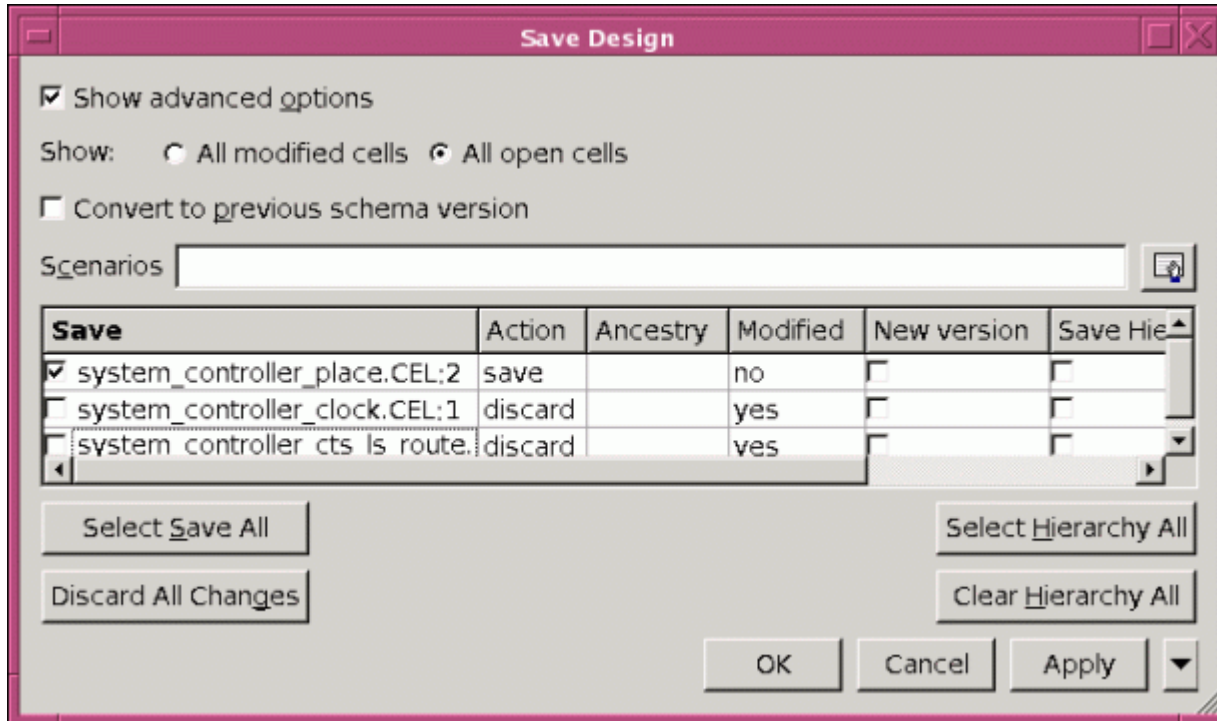
The current cell is the one affected by commands such as `place_opt`, `clock_opt`, and `route_opt`.

To save an edited cell explicitly, use the `save_mw_cel` command. For example, to save the current cell into the currently open Milkyway library, use the following command:

```
icc_shell> save_mw_cel
Information: Saved design named system_controller_route.CEL;2. UIG-5
1
```

Alternatively, you can use the IC Compiler GUI to save the cell. Choose File > Save Design, and the Save Design dialog box offers some advanced options for saving different versions of cells and saving lower-level hierarchical cells, as shown in [Figure 7-9](#).

Figure 7-9 File > Save Design Dialog Box Advanced Options



For more information about the available options for saving a Milkyway cell, see the man page for the `save_mw_cell` command.

To close an open cell, use the `close_mw_cell` command. Before using this command, you should save the edited cell. By default, this command closes all open cells and discards any changes made to them. To ensure that all edited cells are saved, you can use the `-save` option:

```
icc_shell> close_mw_cell -save
```

Alternatively, you can use the IC Compiler GUI to close the cell. Choose File > Close Design, and the Close Design dialog box prompts you to save or discard any open cells. It also offers some advanced options for saving open cells and lower-level hierarchical cells, similar to the Save Design dialog box.

For more information about the available options for closing a Milkyway cell, see the man page for the `close_mw_cell` command.

---

## Creating New Milkyway Libraries and Cells

In the IC Compiler tool, you can easily create new, empty Milkyway design libraries for storing designs by using the `create_mw_lib` command. You can also copy existing Milkyway libraries with the `copy_mw_lib` command and then modify the contents of the copied library. The `split_mw_lib` command splits a hierarchical design library into separate design libraries, one for each top-level cell, so that different teams can work on those blocks independently.

Within a given Milkyway library, you can create new cells by using the `create_mw_cel` command or by copying existing cells with the `copy_mw_cel` command.

---

### Creating and Copying Milkyway Libraries

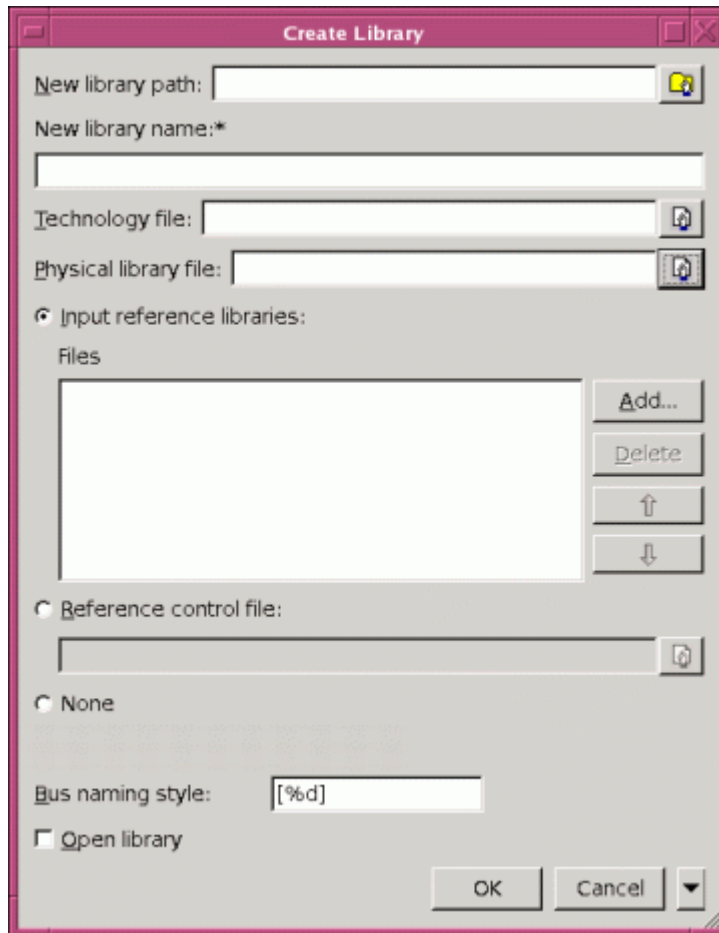
Before you can create a new Milkyway library for storing designs, you must have a technology file (.tf file) to specify the basic parameters for the library. The cells you add to the library must be consistent with this technology file.

To create a new Milkyway library called `my_lib_2` in the `/mw/LIB` path with the technology file `my_techfile.tf`, use the following command:

```
icc_shell> create_mw_lib -technology my_techfile.tf mw_lib_2
```

Alternatively, in the IC Compiler GUI, choose File > Create Library. This opens the Create Library dialog box, which displays the options for creating the new library, as shown in [Figure 7-10](#).

Figure 7-10 File &gt; Create Library Dialog Box



The `create_mw_lib` command and the Create Library dialog box have options to specify the name of the new Milkyway library, the name of the associated technology file, the names of the associated Milkyway reference libraries, and the bus naming style. For details, see the man page for the `create_mw_lib` command.

To copy an existing Milkyway design library, use the `copy_mw_lib` command. For example,

```
icc_shell> copy_mw_lib -from /mw/LIBS/mw_lib_A /mw/LIBS/mw_lib_A_bak
```

If you omit the `-from` option, the currently open library is copied and then closed.

Alternatively, in the IC Compiler GUI, choose File > Copy Library. The Copy Library dialog box lets you browse for the library to copy and browse for the directory in which to store the copied library.



## Creating and Copying Milkyway Cells

To create a new, empty cell in the currently open Milkyway design library, use the `create_mw_cel` command. For example,

```
icc_shell> create_mw_cel my_new_cell
```

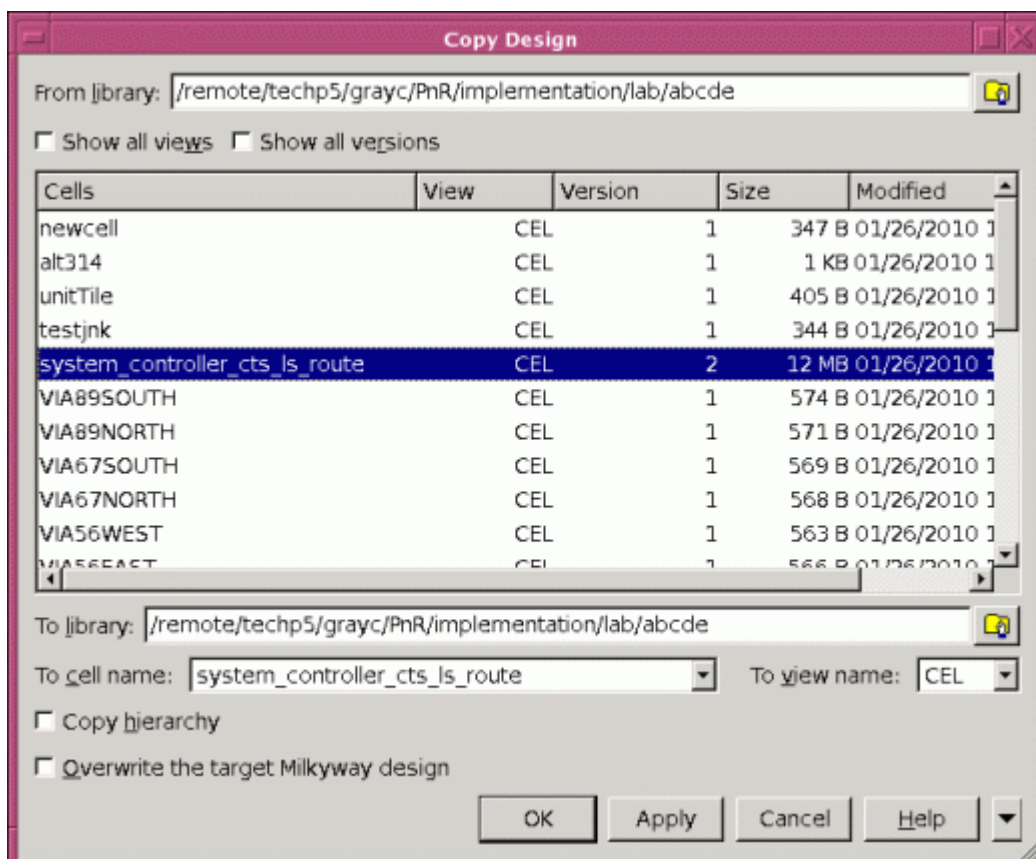
To copy a cell, you must have read permission from the source Milkyway library and read/write permission for the target Milkyway library. Use the `copy_mw_cel` command.

For example, the following command copies `cell_A` to `cell_B` in the currently open Milkyway design library:

```
icc_shell> copy_mw_cel -from cell_A -to cell_B
```

Alternatively, in the IC Compiler GUI, choose **File > Copy Design**. This opens the Copy Design dialog box, which displays the options for copying a cell, as shown in [Figure 7-11](#).

Figure 7-11 File > Copy Design Dialog Box



The `copy_mw_cel` command and the Copy Design dialog box have options to specify the name of the existing cell to copy, the name of the new cell to be created, the source and destination libraries, and whether to copy the lower-level hierarchical cells as well as the specified cell. For details, see the man page for the `copy_mw_cel` command.

---

## Splitting a Milkyway Design Library

The `split_mw_lib` command splits a hierarchical design stored in a library into separate design libraries, one for each cell in the hierarchy of the design, so that different teams can work on those cells independently. Each library is assigned a new name by adding a prefix to the cell name.

For example, to split the hierarchical cell called `chip_top` contained in a design library called `myMWlib` into a different library for each cell in the design hierarchy, use the following command:

```
icc_shell> split_mw_lib -from_library myMWlib chip_top
```

The command has options to specify the source library directory, the destination directory, and the prefix used to create the new library names. There is also an option for checking the potential results without actually performing the library split. For details, see the man page for the `split_mw_lib` command.

---

## Reading and Writing GDSII and OASIS in the IC Compiler Tool

You can read GDSII and OASIS data into the Milkyway database by using the `read_stream` command in the IC Compiler tool. Similarly, you can write design data in GDSII and OASIS format by using the `write_stream` command in the IC Compiler tool.

---

### read\_stream

The `read_stream` command in the IC Compiler tool lets you read design data in GDSII or OASIS format into the IC Compiler tool and save that data in Milkyway cell format. You can use this command to convert library cell information in GDSII or OASIS format into a form that can be used in the IC Compiler tool.

The `set_read_stream_options` command lets you set several data stream-in options, such as how to handle cells that already exist in the Milkyway library and how to handle cell boundary layers. You can also specify the following:

- A cell-type definition file, which identifies the function of each cell: standard cell, pad cell, filler cell, macro, and so on. If this file is not provided, all cells are assumed to be



standard cells. For information about the file format, see [“Cell-Type Definition File” on page 3-3](#).

- A layer mapping file, which identifies the corresponding layer names in the GDSII or OASIS file versus the Milkyway database. If this file is not provided, the layer names defined in the GDSII or OASIS data stream are retained in the Milkyway database. For information about the file format, see [“Layer Mapping File: GDSII or OASIS to Milkyway” on page 3-5](#).

After you set the stream-in options, you can view the option settings with the `report_read_stream_options` command.

To read in the GDSII or OASIS file, use the `read_stream` command. This is the command syntax:

```
read_stream
  [-lib_name lib_name]
  [-format gds | oasis]
  stream_file_name
```

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

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

---

## write\_stream

The `write_stream` command in the IC Compiler tool lets you write out design data in GDSII or OASIS format. The GDSII or OASIS data can then be used for mask creation or for input to external tools.

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 set individual options incrementally. You can report the option settings with the `report_write_stream_options` command. 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
```

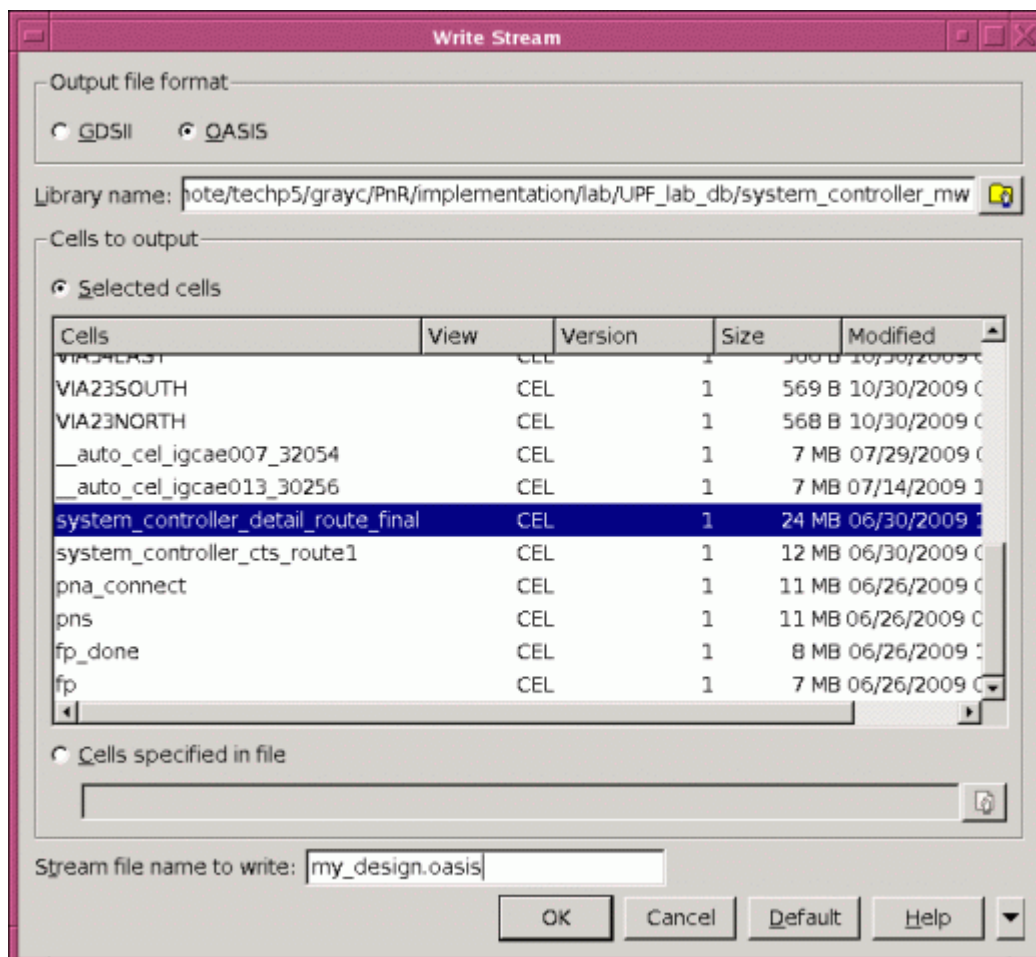
You must specify at least the output file name. The default format is GDSII, so to write the design in OASIS format, you must use the `-format oasis` option. For example,

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

By default, the command writes out all cells in the library that is currently open. 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.

Alternatively, in the IC Compiler GUI, choose **File > Export > Write Stream**. This opens the Write Stream dialog box, which displays the options for writing GDSII and OASIS, as shown in [Figure 7-12](#). The dialog box has the same options as the `write_stream` command. Click OK to generate the GDSII or OASIS file.

Figure 7-12 File > Export > Write Stream Dialog Box



## Milkyway Database Versions

The Milkyway database infrastructure is periodically updated to support new database features. Each new version of the database is called a “schema.” When a major new schema is released, existing Milkyway design information is updated to the new schema when read into the latest version of the tool.

[Table 7-1](#) shows the earliest IC Compiler and Milkyway Environment product version number corresponding to each new Milkyway schema.

*Table 7-1 IC Compiler and Milkyway Environment Versions and Milkyway Schemas*

IC Compiler and Milkyway Environment version	Milkyway schema
J-2014.09	8.1
H-2013.03	7.0
F-2011.09	6.0
D-2010.03-SP1	5.1 (compatible with schema 5.0)
D-2010.03	5.0
C-2009.06	4.0
B-2008.09	3.2
A-2007.12	2.0

When IC Compiler or Milkyway Environment version J-2014.09 reads Milkyway data stored under schema 7 or earlier, the tool automatically updates the data to schema 8.1. Milkyway data stored under schema 8.1 is still compatible with version I-2013.12-SP3 and later tools. For example, IC Compiler version I-2013.12-SP3 can read and update Milkyway data stored under schema 8.1. However, versions of tools earlier than that, such as IC Compiler versions I-2013.12 through I-2013.12-SP2, cannot read schema 8.1 data. Schema 8.1 data cannot be converted back to an earlier schema.

You can explicitly convert Milkyway cells and libraries from the previous to the current Milkyway schema by using the `convert_mw_lib` command. If you do not convert the data explicitly, when you open a Milkyway cell with the latest version of the tool, the tool performs an implicit conversion of the cell and issues a message about the conversion. You should save the converted cell before you close the library. Otherwise, the conversion will need to be performed again when you reopen the cell.

When you open a cell and an implicit update is performed, and you want to save the updated cell and keep a backup copy of the old cell, use the `save_mw_cel` command with the `-increase_version` option to save the updated design under a new cell version number. Alternatively, you can use the `-as` option and specify a different name under which to save the cell.

Using the `convert_mw_lib` command to convert all your data is recommended. This ensures complete conversion of your data and is more efficient than converting cells one at a time. To make a backup copy of your library before performing the conversion, run the version of the IC Compiler or Milkyway Environment tool under the previous version and use the `copy_mw_lib` command.

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

To convert all of the cells contained in the Milkyway design library called `mw_lib_A` to the latest schema, use the following command:

```
icc_shell> convert_mw_lib -all /mw/LIBS/mw_lib_A
```

All CEL views in the library are converted. FRAM views are not affected.

Only the cells contained in the specified library are converted. Lower-level cells referenced in the design but contained in other libraries are not converted.

To convert the cell named `cell_A` in library `mw_lib_A`, use the following command:

```
icc_shell> convert_mw_lib -cell_name cell_A /mw/LIBS/mw_lib_A
```

When you use the `-cell_name` option, all child soft macro cells referenced by the specified cell are converted along with that cell if those cells are contained in the same Milkyway library. If the reference cells are loaded into memory, the in-memory versions are updated but not implicitly saved into their respective reference libraries.

**Note:**

The `convert_mw_lib` command also works in the Milkyway Environment tool in Tcl mode. After a cell is converted by any Synopsys tool to the latest schema, the cell will work with all Synopsys tools of the same release.

# A

## Aserver, Amonitor, and Null Display

---

The following sections describe the software processes required by the Milkyway Environment tool and how to run the tool in batch mode:

- [AServer and AMonitor](#)
- [Null Display](#)

---

## 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, 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 your 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.

---

## Null Display

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 command:

```
% 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 command:

```
% 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, `NullXServer`. The version is stable and fast. It is the recommended version for running regressions. However, 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.

You can monitor the job with a VNC viewer, and you can write out a design layout to a .gif file. However, `Xvnc` is a third-party, free server, so 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.

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. However, it requires you to install a VNC server. Xvnc is a third-party, free server, so there is no guarantee that the server will be stable or fast.



# B

## DEF 5.6 and 5.7 Supported Syntax

---

Physical library information can be provided in Design Exchange Format (DEF), an industry-standard format for design-specific information. The Milkyway Environment tool supports the conversion of DEF version 5.6 and 5.7 data.

- [Supported DEF Version 5.6 and 5.7 Syntax](#)

## Supported DEF Version 5.6 and 5.7 Syntax

[Table B-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 B-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				The parser is a super set of 5.3, 5.4, 5.5, and 5.6.
<code>VERSION versionNumber ;</code>	Yes	No	Yes	
2) DIVIDERCHAR				This section is optional in DEF 5.6. Always assumed "/".
<code>DIVIDERCHAR "character" ;</code>	Yes	Yes	Yes	
3) BUSBITCHARS				This section is optional in DEF 5.6.
<code>BUSBITCHARS "delimiterPair" ;</code>	Yes	No	Yes	
4) DESIGN				The <code>read_def</code> command ignores the <code>designName</code> . The <code>designName</code> in <code>write_def</code> is the top module name.
<code>DESIGN designName ;</code>	Yes	No	Yes	
5) TECHNOLOGY				If the technology name is not NULL, it is set by <code>technologyName</code> ; if it is NULL, you can ignore <code>technologyName</code> .
<code>TECHNOLOGY technologyName ;</code>	Yes	No	Yes	

Table B-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
<b>6) UNITS</b>				
<pre> UNITS DISTANCE MICRONS DEFconvertFactor ; </pre>	Yes	No	Yes	<p>Milkyway determines the design <code>UNITS</code> by the units in the technology file and not by DEF. DEF maps the <code>UNIT</code> value to the corresponding Milkyway design library.</p> <p>The values 10,000 and 20,000 are supported as legal DEF <code>UNITS</code> in DEF 5.6.</p> <p>The <code>read_def</code> command uses the units to determine the conversion scale factor, but it does not change the units in the technology file.</p>
<b>7) HISTORY</b>				
<pre> HISTORY anyText ; </pre>	Yes	No	No	
<b>8) PROPERTY DEFINITIONS</b>				
<pre> PROPERTYDEFINITIONS </pre>	Yes	No	Yes	<p>This section is used to list all properties used in the design. You must define properties in the <code>PROPERTYDEFINITIONS</code> statement before you can refer to them in other sections of the DEF file.</p>
<pre> ObjectType propName propType [RANGE # #] [value   stringValue ; ? </pre>	Yes	Yes	Yes	
<pre> END PROPERTYDEFINITIONS </pre>	Yes	No	Yes	
<pre> ObjectType = { DESIGN   COMPONENT   NET   SPECIALNET   GROUP   ROW   COMPONENTPIN   REGION } </pre>	Yes	Yes	Yes	

Table B-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
PropType = { INTEGER   REAL   STRING }	Yes	Yes	Yes	
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.

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

DEF 5.6 and 5.7 Syntax	Parsed	Annotated	Written out	Comments
				As a rule, full contact array names must be changed as follows: If the contact array is central symmetrical, the name should be <i>viaName-XSize-YSize-ALL-xTimes-yTimes</i> ; otherwise, the name should be <i>viaName/left bottom right/top ALL xTimes yTimes</i> .
[ + VIARULE viaRuleName	Yes	Yes	Yes	<p>When you specify <code>DEFAULT</code> as the reserved via rule name, the via uses the previously defined <code>VIARULE GENERATE</code> rule with the <code>DEFAULT</code> keyword that exists for this routing-cutrouting-routing layer combination.</p> <p>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.</p> <p>If one of the following conditions occur, the <code>read_def</code> command exits:</p> <ul style="list-style-type: none"> <li>• The <code>viaRuleName</code> contact code is not found in the design library.</li> <li>• The contact code in the reference library comes from a <code>VIARULE GENERATE</code> statement.</li> <li>• 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.</li> </ul> <p>If this occurs, no via instance object will be created.</p>

Table B-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 <code>read_def</code> 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 <code>read_def</code> 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 <code>read_def</code> 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 <code>read_def</code> 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				This section is currently not supported.
[Styles	Yes	No	No	

Table B-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
<code>numStyles ;</code>	Yes	No	No	
<code>{- STYLE styleNum pt pt pt ... ;} ...</code>	Yes	No	No	
<code>END STYLES]</code>	Yes	No	No	
<b>15) NONDEFAULTRULES</b>				
<code>NONDEFAULTRULES numRules ;</code>	Yes	No	Yes	
<code>{- ruleName</code>	Yes	Yes	Yes	
<code>[+ HARDSPACING]</code>	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.
<code>{+ LAYER layerName</code>	Yes	Yes	Yes	
<code>WIDTH minWidth</code>	Yes	Yes	Yes	
<code>[DIAGWIDTH diagWidth]</code>	Yes	No	No	If used in DEF nondefault rule definitions, the <code>read_def</code> command produces a warning message and the <code>write_def</code> command cannot write it out.
<code>[SPACING minSpacing]</code>	Yes	Yes	Yes	
<code>[WIREEXT wireExt]]} ...</code>	Yes	Yes	Yes	
<code>[+ VIA viaName] ...</code>	Yes	Yes	Yes	
<code>[+ VIARULE viaRuleName] ...</code>	Yes	Yes	Yes	



*Table B-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
[+ MINCUTS cutLayerName numCuts] ...	Yes	No	No	If used in DEF nondefault rule definitions, the <code>read_def</code> command produces a warning message and the <code>write_def</code> command cannot write it out.
[+ PROPERTY {propName propVal} ...] ...	Yes	No	No	
END NONDEFAULTRULES	Yes	No	Yes	
16) REGIONS				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.
REGIONS numRegions ;	Yes	No	Yes	The <code>read_def</code> command adds the leaf cells into the move bound, but not the hierarchical cells.  The <code>write_def</code> command uses the move bound type to determine the appropriate DEF region type to write into the REGION section of the DEF file.  The <code>create_bounds</code> command is used to create the move bounds without a DEF file, and the <code>report_bounds -all</code> command is used to report all move bounds.
[ - regionName pt pt	Yes	Yes	Yes	
[ pt pt ]?	Yes	Yes	Yes	The <code>read_def</code> and <code>write_def</code> commands support multiple rectangles.

Table B-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 is 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 is 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	
<b>17) COMPONENTS</b>				
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 B-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				The PINS section supports multilayer 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.
PINS numPins ;	Yes	Yes	Yes	
[ [ - 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 B-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	
<b>19) PINPROPERTIES</b>				
PINPROPERTIES num ;	Yes	No	Yes	
[- {compName pinName   PIN pinName}	Yes	Yes	Yes	
[ + PROPERTY { propName propVal }... ]... ; ]...	Yes	Yes	Yes	
END PINPROPERTIES	Yes	No	Yes	
<b>20) BLOCKAGES</b>				
Blockage numblockages ;	Yes	No	Yes	

Table B-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 preroute route guides from signal route guides. Therefore, the <code>write_def</code> command does not output preroute route guides.
+ COMPONENT CompName	Yes	Yes	Yes	<code>read_def</code> annotates BLOCKAGE + COMPONENT as the keepout margin in the Milkyway database.
+ 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	Yes	
{ 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 B-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	Currently, the <code>write_def</code> command handles <code>PUSHDOWN</code> placement blockage the same as other placement blockages, and the <code>PUSHDOWN</code> keyword cannot be written out.
+ PUSHDOWN	Yes	Yes	No	
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, <code>UnsupportedRouteType</code> , is defined and used for OPC shapes.

Table B-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
{ Rect pt pt	Yes	Yes	Yes	
-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 <code>UnsupportedRouteType</code> . If it is not OPC, the route type is <code>MiscFillTrackRouteType</code> .
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 <code>nameId</code> .
( compNameRegExpr pinName )	Yes	Yes	Yes	
+ SYNTHESIZED	Yes	No	No	Connectivity that is read in for <code>SYNTHESIZED</code> is ignored.
+ VOLTAGE volts	Yes	Yes	Yes	
+ SOURCE { NETLIST   DIST   USER   TIMING }	Yes	Yes	Yes	
+ FIXEDBUMP	Yes	Yes	Yes	
+ ORIGINAL netName	Yes	Yes	Yes	

Table B-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	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 { STEINER   BALANCED   WIREDLOGIC   TRUNK }	Yes	Yes	Yes	
+ ESTCAP wireCapacitance	Yes	Yes	Yes	
+ WEIGHT weight	Yes	Yes	Yes	
+ PROPERTY: { propName propVal }... ]	Yes	Yes	Yes	
SPECIALWIRING				<p>Using the <code>read_def -preserve_wire_ends</code> 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.</p> <p>You should use this option when the DEF file is from a third party tool after the <code>read_def</code> command. Then, the <code>read_def</code> 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.</p>



Table B-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
+ 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	
{+ COVER   + FIXED   +ROUTED}	Yes	Yes	Yes	
+ SHIELD shieldNetName	Yes	Yes	Yes	
{ layerName routeWidth   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	

Table B-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
<b>24) NETS</b>				
NET numNets	Yes	No	Yes	
- netName	Yes	Yes	Yes	The Name string is referenced by nameId.
{ compName pinName   PIN pinName [ + SYNTHESIZED ] }	Yes	Yes	Yes	
MUSTJOIN ( compName pinName )	Yes	Yes	Yes	
+ SHIELDNET shieldNetName	Yes	Yes	Yes	
+ VPIN vpinName [ LAYER layerName ] pt pt [ PLACED pt orient   FIXED pt orient   COVER pt orient ]	Yes	No	No	
+ SUBNET subnetName [ ( { compName pinName   PIN pinName   VPIN vpinName } ) ] [NONDEFAULTRULE ruleName ] [ regularWiring ] ...]	Yes	No	No	
+ XTALK class	Yes	Yes	Yes	
+ NONDEFAULTRULE ruleName	Yes	Yes	Yes	
+ SOURCE { DIST   NETLIST   TEST   TIMING   USER }	Yes	No	Yes	
+ FIXEDBUMP	Yes	No	Yes	
+ FREQUENCY frequency	Yes	No	Yes	

Table B-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
+ 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	
+ 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	

Table B-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
<b>25) SCANCHAINS</b>				
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 { fixedInComp   PIN } [ outPin ] ]	Yes	Yes	Yes	
[ + FLOATING {floatingComp [ ( IN pin ) ] [ ( OUT pin ) ]	Yes	Yes	Yes	
[ ( BITS numBits ) ]} ...]	Yes	Yes	Yes	
[+ ORDERED {fixedComp [ ( IN pin ) ] [ ( OUT pin ) ]	Yes	Yes	Yes	
[ ( BITS numBits ) ]} ...]	Yes	Yes	Yes	
[ + STOP { fixedOutComp   PIN } [ inPin ] ] ; ]...	Yes	Yes	Yes	
END SCANCHAINS	Yes	No	Yes	
<b>26) GROUPS</b>				
GROUPS NumOfGroups ;	Yes	No	Yes	
[ - groupName compNameRegExpr ...	Yes	Yes	Yes	

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

<b>DEF 5.6 and 5.7 Syntax</b>	<b>Parsed</b>	<b>Anno- tated</b>	<b>Written out</b>	<b>Comments</b>
[+ REGION regionName ]	Yes	Yes	Yes	
[ + PROPERTY { propName propVal }... ]... ; ]?	Yes	Yes	Yes	
END GROUPS ;	Yes	No	Yes	
<b>27) END DESIGN</b>				
END DESIGN	Yes	No	Yes	



# Index

---

## A

- ALF format 6-13
- AMonitor A-2
- antenna properties 5-31
  - diode cells 5-36
  - macros 5-33
  - standard cells 5-31
- AServer A-2

## B

- back-bias pins 5-2
- blockage areas (BPV) 5-17
- blockage merging (BPV) 5-17
- blockage types (BPV) 5-8
- blockage, pin, and via extraction (BPV) 5-4
- boundary, multiple-height 5-25
- boundary, place-and-route 5-22
- BPV (blockage, pin, and via extraction) 1-3, 5-4
  - blockage areas 5-17
  - blockage types 5-8
  - Cell Library > Set PR Boundary command 5-22
  - Check Wire Track dialog box 5-30
  - Define Wire Track dialog box 5-30
  - diagram 5-5

- extract pin by text options 5-9
- horizontal via placement 5-19
- IC Compiler commands 5-5
- internal pin exclusion 5-20
- macro extract blockage options 5-13
- macro extract pin by text options 5-15
- macro extract via options 5-16
- macros 5-12
- pin identification by text 5-16
- preserve system metal blockage 5-8
- real metal blockage 5-8
- route guides 5-8
- Set Multiple Height PR Boundary dialog box 5-25
- Set PR Boundary dialog box 5-24
- standard cells and pad cells 5-6
- system metal blockage 5-8
- text fall through options 5-17
- thin wire rules 5-8
- Verti Via Grid Offset 5-19
- via extraction options 5-12
- Wire Tracks > Check Pin on Track command 5-30
- Wire Tracks > Define Unit Tile Wire Tracks command 5-29

- Browse Cell dialog box 2-5
- Browse Library dialog box 2-4

## C

CEL view 1-3

    revision number 2-6

    title bar 2-6

    versus FRAM view 1-3

cell

    closing 2-7

    closing in IC Compiler 7-12

    opening 2-5

    opening in IC Compiler 7-9

    saving in IC Compiler 7-11

Cell > Close command 2-7

Cell > Make Macro Abstract command 5-12

Cell > Open command 2-5

Cell Library > Blockage, Pin & Via command 5-6

Cell Library > LEF In 4-5

Cell Library > Multi Height Cell command 5-25

Cell Library > Set PR Boundary command 5-22

Cell Library > Smash command 5-3

cell types 1-3

cell view window 2-6

cells 1-2

    copying in IC Compiler 7-15

    creating in IC Compiler 7-15

    flattening hierarchical 5-3

cell-type definition file, GDSII or OASIS 3-3

Check Wire Track dialog box 5-30

check\_library command 6-1

    cell group 6-6

    in Library Compiler 6-2

    in Milkyway Environment 6-2

    library group 6-6

    logic libraries 6-5

    overview 6-2

    overview diagram 6-3

    physical libraries 6-9

    pin and pg\_pin groups 6-6

    report examples 6-14

    set\_check\_library\_options command 6-4

    syntax 6-2

Close Library dialog box 7-9

close\_mw\_cel command 7-12

close\_mw\_lib command 7-8

commands

    Cell > Make Macro Abstract 5-12

    Cell > Open 2-5

    Cell Library > Blockage, Pin & Via 5-6

    Cell Library > LEF in 4-5

    Cell Library > Multi Height Cell command 5-25

    Cell Library > Stream In 3-2

    check\_library 6-1

    close\_mw\_cel 7-12

    close\_mw\_lib 7-8

    convert\_mw\_lib 7-19

    copy\_mw\_cel 7-15

    copy\_mw\_lib 7-14

    create\_macro\_fram 5-5

    create\_mw\_cel 7-15

    create\_mw\_lib 7-13

    current\_mw\_cel 7-11

    extract\_blockage\_pin\_via 5-5

    File > Close Library 7-9

    File > Copy Design 7-15

    File > Create Library 7-14

    File > Export > Write Stream 7-18

    File > Open Design 7-10

    File > Open Library 7-8

    File > Save Design 7-12

    File > Set Library Reference 7-5

    flatten\_cell 5-4

    Library > Create 2-15

    Library > Open 2-4

    open\_mw\_cel 2-8, 7-9

    open\_mw\_lib 2-8, 7-7

    Output > LEF Out 4-11

    Output > Stream Out 3-8

    printvar 2-11

    read\_def 4-14

    read\_gds 3-3, 4-7

    read\_oasis 3-15



- report\_mw\_lib 7-5
- save\_mw\_cel 7-11
- set\_app\_var 2-11
- set\_attribute 5-2
- set\_check\_library\_options 6-4
- set\_local\_link\_library 1-4
- set\_mw\_lib\_reference 7-4
- set\_write\_stream\_options 7-16, 7-17
- split\_mw\_lib 7-16
- Wire Tracks > Check Pin on Track 5-30
- Wire Tracks > Define Unit Tile Wire Tracks 5-29
- write\_def 4-15
- write\_gds 3-9
- write\_lef 4-12
- write\_mw\_lib\_files 7-6
- write\_oasis 3-17
- write\_stream 7-17

## CONN view 1-3

conventions for documentation 1-xi

convert\_mw\_lib command 7-19

Copy Design dialog box 7-15

copy\_mw\_cel command 7-15

copy\_mw\_lib command 7-14

Create Library dialog box 2-15, 7-14

create\_macro\_fram command 5-5

create\_mw\_cel command 7-15

create\_mw\_lib command 7-13

current\_mw\_cel command 7-11

customer support 1-xii

## D

.db files, importing pin attributes 5-36

.db libraries 1-4

DEF 4-12

- exporting 4-14

- import and export flow 4-17

- importing 4-13

- read\_def command 4-14

- recommended flows 4-16

- syntax versions 5.6 and 5.7 B-2

- Verilog and DEF input flow 4-18

- write\_def command 4-15

Define Wire Track dialog box 5-30

design library, Milkyway 7-2

design versus reference libraries 7-2

## E

electromigration constraints

- ALF format 6-13

- plib format 6-13

ERR view 1-3

extended Milkyway layers 2-16

external\_pin\_file variable 5-20

Extract Blockage dialog box 5-6

extract\_blockage\_pin\_via command 5-5

## F

File > Close Library command 7-9

File > Copy Design command 7-15

File > Create Library command 7-14

File > Export > Write Stream command 7-18

File > Open Design command 7-10

File > Open Library command 7-8

File > Save Design command 7-12

File > Set Library Reference command 7-5

FILL view 1-3

flatten hierarchical cells 5-3

flatten\_cell command 5-4

FRAM view 1-3

- creation (BPV) 5-4

- versus CEL view 1-3

## G

GDSII 3-1

- cell-type definition file 3-3

- GDSII to Milkyway 3-2
- layer mapping file, stream in 3-5
- layer mapping file, stream out 3-10
- objects that can be streamed out 3-9
- writing from IC Compiler 7-16, 7-17
- ground ports 5-2

## H

- hierarchical cells, flattening 5-3
- hierarchy separator 2-16
- horizontal via placement (BPV) 5-19
- HSPICE 1-5

## I

### IC Compiler

- check\_library command 6-2
- Close Library dialog box 7-9
- close\_mw\_cel command 7-12
- close\_mw\_lib command 7-8
- convert\_mw\_lib command 7-19
- Copy Design dialog box 7-15
- copy\_mw\_cel command 7-15
- copy\_mw\_lib command 7-14
- Create Library dialog box 7-14
- create\_macro\_fram command 5-5
- create\_mw\_cel command 7-15
- create\_mw\_lib command 7-13
- creating new Milkyway cells 7-13
- creating new Milkyway libraries 7-13
- current\_mw\_cel command 7-11
- extract\_blockage\_pin\_via command 5-5
- File > Close Library command 7-9
- File > Copy Design command 7-15
- File > Create Library command 7-14
- File > Export > Write Stream command 7-18
- File > Open Design command 7-10
- File > Open Library command 7-8
- File > Save Design command 7-12
- libraries used 1-2

- library editing commands 1-7
- Milkyway database commands 1-7
- Open Design dialog box 7-10
- Open Library dialog box 7-8
- open\_mw\_cel command 7-9
- open\_mw\_lib command 7-7
- opening and closing Milkyway cells 7-9
- opening and closing Milkyway libraries 7-7
- reference control file 7-6
- report\_mw\_lib command 7-5
- Save Design dialog box 7-12
- save\_mw\_cel command 7-11
- Set Library Reference dialog box 7-5
- set\_mw\_lib\_reference command 7-4
- split\_mw\_lib command 7-16
- using Milkyway libraries 7-1
- Write Stream dialog box 7-18
- write\_mw\_lib\_files command 7-6
- writing GDSII 7-16, 7-17

IC Compiler writing OASIS 7-16, 7-17

internal pin exclusion (BPV) 5-20

introduction to library data preparation 1-1

## L

- layer mapping file, GDSII or OASIS stream in 3-5
- layer mapping file, GDSII or OASIS stream out 3-10
- layer mapping file, LEF in 4-9
- layers, extended 2-16
- LEF to Milkyway 4-3
  - advanced (manual) LEF input flow 4-7
  - automated LEF input flow 4-5
  - flow diagram 4-3
  - mixed input, LEF, GDSII, TF, CLF, PLIB 4-10
- lef\_layer\_tf\_number\_mapper.pl script 4-9
- LEF/DEF
  - DEF syntax versions 5.6 and 5.7 B-2
  - design exchange format (DEF) 4-12
  - exporting DEF 4-14

- importing DEF 4-13
  - introduction 4-2
  - layer mapping file, LEF in 4-9
  - LEF to Milkyway 4-3
  - Milkyway to LEF 4-11
  - Read Library flow 4-7
  - read\_def command 4-14
  - recommended DEF flows 4-16
  - write\_def command 4-15
  - write\_lef command 4-12
  - .lib libraries 1-4
  - Liberty libraries 1-4
  - Liberty NCX 1-5
  - libraries
    - cell types 1-3
    - cells 1-2
    - closing in IC Compiler 7-8
    - copying in IC Compiler 7-14
    - creating in IC Compiler 7-13
    - creating new 2-15
    - .db 1-4
    - design versus reference 7-2
    - IC Compiler library editing commands 1-7
    - in IC Compiler 1-2
    - .lib 1-4
    - Liberty 1-4
    - logic 1-4
    - Milkyway 1-2
    - opening in IC Compiler 7-7
    - physical versus logic 1-2
    - reference control file 7-6
    - reference (Milkyway) 7-2
    - splitting in IC Compiler 7-16
    - updating in IC Compiler 7-19
  - Library > Create command 2-15
  - Library > Open command 2-4
  - library cell preparation 5-1
    - blockage, pin, and via extraction 5-4
    - flatten hierarchical cells 5-3
    - FRAM view creation 5-4
    - identify power and ground ports 5-2
    - multiple-height boundary 5-25
    - place-and-route boundary 5-22
    - unit tile wire tracks 5-27
    - wire tracks 5-27
  - library checking 6-1
    - overview 6-2
    - special checks 6-7
    - specific checks 6-7
    - validating physical libraries 6-9
  - Library Compiler 1-4, 1-5
  - library data preparation
    - flow description 1-5
    - flow diagram, detailed 2-14
    - flow diagram, preparation and usage 1-6
  - library data preparation, introduction 1-1
  - library preparation
    - GDSII 3-1
    - LEF/DEF 4-1
    - OASIS 3-1
  - link\_library variable 1-4
  - local\_link\_library attribute 1-4
  - logic libraries 1-4
  - logic libraries, validating 6-5
  - logic versus physical libraries 1-2
- ## M
- Make Macro dialog box 5-12
  - man page command 2-9
  - merging blockages (BPV) 5-17
  - Milkyway
    - cell types 1-3
    - database model version 7-19
    - design library 7-2
    - libraries 1-2
    - library structure 7-2
    - reference libraries 7-2
    - updating database 7-19
  - Milkyway Environment 2-1, 2-10, 2-11
    - BPV (blockage, pin, and via extraction) 5-4
    - diagram 5-6

- Browse Cell dialog box 2-5
- Browse Library dialog box 2-4
- Cell > Close command 2-7
- Cell > Open command 2-5
- Cell Library > Stream In command 3-2
- cell view window 2-6
- cell-type definition file 3-3
- command entry methods 2-3
- Create Library dialog box 2-15
- flatten\_cell command 5-4
- GDSII to Milkyway 3-2
- GUI command entry 2-3
- layer mapping file, LEF in 4-9
- layer mapping file, stream in 3-5
- layer mapping file, stream out 3-10
- LEF to Milkyway 4-3
- Library > Create command 2-15
- Library > Open command 2-4
- Milkyway to GDSII 3-8
- Milkyway to LEF 4-11
- Milkyway to OASIS 3-15
- OASIS to Milkyway 3-14
- objects that can be streamed out 3-9
- online help 2-12
- Open Cell dialog box 2-5
- Output > LEF Out command 4-11
- Output > Stream Out command 3-8
- overview 1-9
- read\_def command 4-14
- read\_gds command 3-3, 4-7
- read\_oasis command 3-15
- Scheme command entry 2-11
- starting 2-2
- Stream In dialog box 3-2
- Stream Out Data File dialog box 3-8
- Tcl command entry 2-7
- Write LEF dialog box 4-11
- write\_def command 4-15
- write\_gds command 3-9
- write\_lef command 4-12
- write\_oasis command 3-17

- Milkyway to GDSII 3-8
- Milkyway to LEF 4-11
- Milkyway to OASIS 3-15
- multiple-height boundary 5-25

## N

- null display A-2

## O

- OASIS 3-1
  - cell type definition file 3-3
  - layer mapping file, stream in 3-5
  - layer mapping file, stream out 3-10
  - OASIS to Milkyway 3-14
  - objects that can be streamed out 3-9
  - writing from IC Compiler 7-16, 7-17
- online help, Milkyway Environment 2-12
- Open Cell dialog box 2-5
- Open Design dialog box 7-10
- Open Library dialog box 7-8
- open\_mw\_cel command 2-8, 7-9
- open\_mw\_lib command 2-8, 7-7

## P

- physical libraries 1-2
- physical versus logic libraries 1-2
- pin attributes, importing from .db file 5-36
- pin identification (BPV) 5-16
- PINASSIGN view 1-3
- place-and-route boundary 5-22
- port\_type attribute 5-2
- ports, power and ground 5-2
- power ports 5-2
- preserve system metal blockage 5-8
- printvar command 2-11

## R

- Read LEF dialog box 4-5
- read\_def command 4-14
- read\_gds command 3-3, 4-7
- read\_oasis command 3-15
- real metal blockage (BPV) 5-8
- reference control file 7-6
- reference libraries, Milkyway 7-2
  - reference control file 7-6
  - reporting 7-5
  - setting 7-4
- reference versus design libraries 7-2
- report\_mw\_lib command 7-5
- route guides (BPV) 5-8
- ROUTE view 1-3

## S

- Save Design dialog box 7-12
- save\_mw\_cel command 7-11
- search\_path variable 1-4
- server process A-2
- Set Library Reference dialog box 7-5
- Set Multiple Height PR Boundary dialog box 5-25
- Set PR Boundary dialog box 5-24
- set\_app\_var command 2-11
- set\_attribute command 5-2
- set\_check\_library\_options command 6-4, 6-5
- set\_local\_link\_library command 1-4
- set\_mw\_lib\_reference command 7-4
- set\_write\_stream\_options command 7-16, 7-17
- Smash dialog box 5-3
- SMASH view 1-3
- SolvNet
  - accessing 1-xii
  - documentation 1-x
- split\_mw\_lib command 7-16

- Stream In dialog box 3-2

- Stream Out Data File dialog box 3-8

- system metal blockage (BPV) 5-8

## T

- target\_library variable 1-4
- Tcl command entry 2-7
  - help 2-9
  - man page 2-9
  - script execution 2-10
  - source command 2-10
  - variables 2-11
- thin wire rules (BPV) 5-8

## U

- unit tile wire tracks 5-27
- updating data to latest Milkyway database
  - model version 7-19

## V

- validating logic libraries 6-5
- validating physical libraries 6-9
- variables
  - external\_pin\_file 5-20
  - link\_library 1-4
  - search\_path 1-4
  - target\_library 1-4
- variables, Milkyway 2-11
- version, Milkyway 7-19
- Verti Via Grid Offset (BPV) 5-19
- via placement (BPV) 5-19
- view types 1-3

## W

- Wire Tracks > Check Pin on Track command 5-30

Wire Tracks > Define Unit Tile Wire Tracks  
command 5-29

wire tracks, unit tile 5-27

Write LEF dialog box 4-11

Write Stream dialog box 7-18

write\_def command 4-15

write\_gds command 3-9

write\_lef command 4-12

write\_mw\_lib\_files command 7-6

write\_oasis command 3-17

write\_stream command 7-17