

5

Working With Libraries

This chapter contains the following sections:

- [Selecting a Semiconductor Vendor](#)
- [Understanding the Library Requirements](#)
- [Specifying Libraries](#)
- [Loading Libraries](#)
- [Listing Libraries](#)
- [Reporting Library Contents](#)
- [Specifying Library Objects](#)
- [Directing Library Cell Usage](#)
- [Removing Libraries From Memory](#)
- [Saving Libraries](#)

Selecting a Semiconductor Vendor

One of the first things you must do when designing a chip is to select the semiconductor vendor and technology you want to use. Consider the following issues during the selection process:

- Maximum frequency of operation
- Physical restrictions
- Power restrictions
- Packaging restrictions
- Clock-tree implementation
- Floorplanning
- Back-annotation support
- Design support for libraries, megacells, and RAMs
- Available cores
- Available test methods and scan styles

Understanding the Library Requirements

Design Compiler uses these libraries:

- Technology libraries
- Symbol libraries
- DesignWare libraries

The following sections describe these libraries.

Technology Libraries

Technology libraries contain information about the characteristics and functions of each cell provided in a semiconductor vendor's library. Semiconductor vendors maintain and distribute the technology libraries.

Cell characteristics include information such as cell names, pin names, area, delay times, and pin loading. The technology library also defines the conditions that must be met for a functional design (for example, the maximum transition time for nets). These conditions are called design rule constraints.

In addition to cell information and design rule constraints, technology libraries specify the operating conditions and wire load models specific to that technology.

Design Compiler requires the technology libraries to be in .db format. In most cases, your semiconductor vendor provides you .db format libraries. If you are provided only with library source code, see the Library Compiler documentation for information about generating technology libraries.

Design Compiler uses technology libraries to

- Implement the design function

The technology libraries that Design Compiler maps to during optimization are called target libraries. The target libraries contain the cells used to generate the netlist and definitions for the design's operating conditions.

The target libraries used to compile or translate a design become the local link libraries for the design. Design Compiler saves this information in the design's `local_link_library` attribute. See “Working With Attributes” in Chapter 6 for information about attributes.

- Resolve cell references

The technology libraries that Design Compiler uses to resolve cell references are called link libraries. In addition to technology libraries, link libraries can also include design files. The link libraries contain the descriptions of cells (library cells as well as subdesigns) in a mapped netlist.

Link libraries include both local link libraries (`local_link_library` attribute) and system link libraries (`link_library` variable).

See “Linking Designs” in Chapter 6 for more information about resolving references.

- Calculate timing values and path delays

The link libraries define the delay models used to calculate timing values and path delays. See the Library Compiler documentation for information about the various delay models.

Symbol Libraries

Symbol libraries contain definitions of the graphic symbols that represent library cells in the design schematics. Semiconductor vendors maintain and distribute the symbol libraries.

Design Compiler uses symbol libraries to generate the design schematic. You must have Design Analyzer to view the design schematic.

When you generate the design schematic, Design Compiler performs a one-to-one mapping of cells in the netlist to cells in the symbol library.

DesignWare Libraries

A DesignWare library is a collection of reusable circuit-design building blocks (components) that are tightly integrated into the Synopsys synthesis environment.

DesignWare components that implement many of the built-in HDL operators are provided by Synopsys. These operators include +, -, *, <, >, <=, >=, and the operations defined by if and case statements.

Additional DesignWare libraries can be developed by users at their sites by using DesignWare Developer, or they can be licensed from Synopsys or from third parties. To use licensed DesignWare components, you need a license key.

Specifying Libraries

You use `dc_shell` variables to specify the libraries used by Design Compiler. Table 5-1 lists the variables for each library type as well as the typical file extension for the library.

Table 5-1 Library Variables

Library Type	Variable	Default Value	File Extension
Target Library	<code>target_library</code>	<code>{"your_library.db"}</code>	<code>.db</code>
Link Library	<code>link_library</code>	<code>{"*", "your_library.db"}</code>	<code>.db</code>
Symbol Library	<code>symbol_library</code>	<code>{"your_library.sdb"}</code>	<code>.sdb</code>
DesignWare Library	<code>synthetic_library,</code>	<code>{}</code>	<code>.sldb</code>

Using a Library Search Path

You can specify the library location by using either the complete path or only the file name. If you specify only the file name, Design Compiler uses the search path defined in the `search_path` variable to locate the library files. By default, the search path includes the current working directory and `$SYNOPSYS/libraries/syn`. Design Compiler looks for the library files, starting with the leftmost directory specified in the `search_path` variable, and uses the first matching library file it finds.

For example, assume that you have technology libraries named `my_lib.db` in both the `lib` directory and the `vhdl` directory. Given the following search path,

```
search_path = {lib vhdl} + search_path
```

Design Compiler uses the `my_lib.db` file found in the `lib` directory, because it encounters the `lib` directory first.

You can use the `which` command to see which library file Design Compiler finds (in order).

```
dc_shell> which my_lib.db
{/usr/lib/my_lib.db, "/usr/vhdl/my_lib.db"}
```

Specifying Technology Libraries

Specify the same value for the target library and the link library (except when performing technology translation). For the link library, you should also specify the asterisk character (*), which specifies that Design Compiler should also search the designs in memory when resolving cell references. If the `link_library` variable has no asterisk, the designs loaded in memory are not searched. As a result, designs might not be found during linking and might become unresolved.

When specifying the files in the `link_library` variable, consider that Design Compiler searches these files from left to right when resolving references and stops searching when it finds a reference. In the following example, the designs in memory are searched before the `lsi_10k` library:

```
link_library = {"*" lsi_10k.db}
```

See “Linking Designs” in Chapter 6 for more information about resolving references.

Specifying DesignWare Libraries

You do not need to specify the standard synthetic library, `standard.sldb`, that implements the built-in HDL operators. The software automatically uses this library.

If you are using additional DesignWare libraries, you must specify these libraries, using the `synthetic_library` variable (for optimization purposes) and the `link_library` variable (for cell resolution purposes).

For more information about using DesignWare libraries, see the *DesignWare User Guide*.

Loading Libraries

Design Compiler uses binary libraries (`.db` format for technology libraries and `.sdb` format for symbol libraries), and automatically loads these libraries when needed.

If your library is not in the appropriate binary format, use the `read_lib` command to compile the library source. The `read_lib` command requires a Library-Compiler license.

To manually load a binary library, use the `read` command.

```
dc_shell> read my_lib.db
dc_shell> read my_lib.sdb
```

Listing Libraries

Design Compiler refers to a library loaded in memory by its name. The library statement in the library source defines the library name.

To list the names of the libraries loaded in memory, use the `list_libs` command.

```
dc_shell> list_libs
my_lib      my_symbol_lib
1
```

To list the path and file name information along with the names, use the `list -libraries` command (dcsh mode only).

```
dc_shell> list -libraries
Library      File          Path
-----
my_lib       my_lib.db     /synopsys/libraries
my_symbol_lib my_lib.sdb    /synopsys/libraries
```

Reporting Library Contents

Use the `report_lib` command to report the contents of a library. The `report_lib` command can report the following data:

- Library units
- Operating conditions
- Wire load models
- Cells (including cell exclusions, preferences, and other attributes)

Specifying Library Objects

Library objects are the vendor-specific cells and their pins.

The Design Compiler naming convention for library objects is

[file:]library/cell[/pin]

file:

The file name of a technology library followed by a colon (:). If you have multiple libraries loaded in memory with the same name, you must specify the file name.

library/

The name of a library in memory, followed by a slash (/).

cell

The name of a library cell.

/pin

The name of a cell's pin.

For example, to set the `dont_use` attribute on the AND4 cell in the `my_lib` library, enter

```
dc_shell> set_dont_use my_lib/AND4
```

To set the `disable_timing` attribute on the Z pin of the AND4 cell in the `my_lib` library, enter

```
dc_shell> set_disable_timing find(pin, my_lib/AND4/Z)
```

Directing Library Cell Usage

When Design Compiler maps a design to a technology library, it selects components (library cells) from that library. You can influence the choice of components (library cells) by

- Excluding cells from the target library
- Specifying cell preferences

Excluding Cells From the Target Library

Use the `set_dont_use` command to exclude cells from the target library. Design Compiler does not use these excluded cells during optimization.

This command affects only the copy of the library that is currently loaded into memory and has no effect on the version that exists on disk. However, if you save the library, the exclusions are saved and the cells are permanently disabled.

For example, to prevent Design Compiler from using the high-drive inverter `INV_HD`, enter

```
dc_shell> set_dont_use MY_LIB/INV_HD
Performing set_dont_use on library cell 'MY_LIB/INV_HD'.
1
```

Use the `remove_attribute` command to reinclude excluded cells in the target library.

```
dc_shell> remove_attribute MY_LIB/INV_HD dont_use
Performing remove_attribute on library cell 'MY_LIB/INV_HD'.
1
```

Specifying Cell Preferences

Use the `set_prefer` command to indicate preferred cells. You can issue this command with or without the `-min` option.

Use the command without the `-min` option if you want Design Compiler to prefer certain cells during the initial mapping of the design.

- Set the preferred attribute on particular cells to override the default cell identified by the library analysis step. This step occurs at the start of compilation to identify the starting cell size for the initial mapping.
- Set the preferred attribute on cells if you know the preferred starting size of those complex cells or cells with complex timing arcs (such as memories and banked components).

You do not normally need to set the preferred attribute as part of your regular compile methodology because the library analysis step determines a good starting cell automatically.

Because nonpreferred gates can be chosen to meet optimization constraints, the effect of preferred attributes might not be noticeable after optimization.

For example, to set a preference for the low-drive inverter `INV_LD`, enter

```
dc_shell> set_prefer MY_LIB/INV_LD
Performing set_prefer on library cell 'MY_LIB/INV_LD'.
1
```

Use the `remove_attribute` command to remove cell preferences.

```
dc_shell> remove_attribute MY_LIB/INV_LD preferred  
Performing remove_attribute on library cell 'MY_LIB/INV_LD'.  
1
```

Use the `-min` option if you want Design Compiler to prefer fewer (but larger area) buffers or inverters when fixing hold-time violations. Normally, Design Compiler gives preference to smaller cell area over the number of cells used in a chain of buffers or inverters. You can change this preference by using the `-min` option, which tells Design Compiler to minimize the number of buffers or inverters by using larger area cells.

For example, to set a `hold_preferred` attribute for the inverter IV, enter

```
dc_shell> set_prefer -min class/IV  
Performing set_prefer on library cell 'class/IV'.  
1
```

Use the `remove_attribute` command to remove the `hold_preferred` cell attribute.

```
dc_shell> remove_attribute class/IV hold_preferred  
Performing remove_attribute on library cell 'class/IV'.  
{ "class/IV" }
```

Removing Libraries From Memory

The `remove_design` command removes libraries from `dc_shell` memory. If you have multiple libraries with the same name loaded into memory, you must specify the path as well as the library name. Use the `list -libraries` command to see the path for each library in memory (dcsh mode only).

Saving Libraries

The `write_lib` command saves (writes to disk) a compiled library in Synopsys database, EDIF, or VHDL format.