cādence®

# Encounter® RTL Compiler Synthesis Flows

**Product Version 9.1**
**July 2009**

# Contents

# Preface

This preface contains the following sections:

## About This Manual

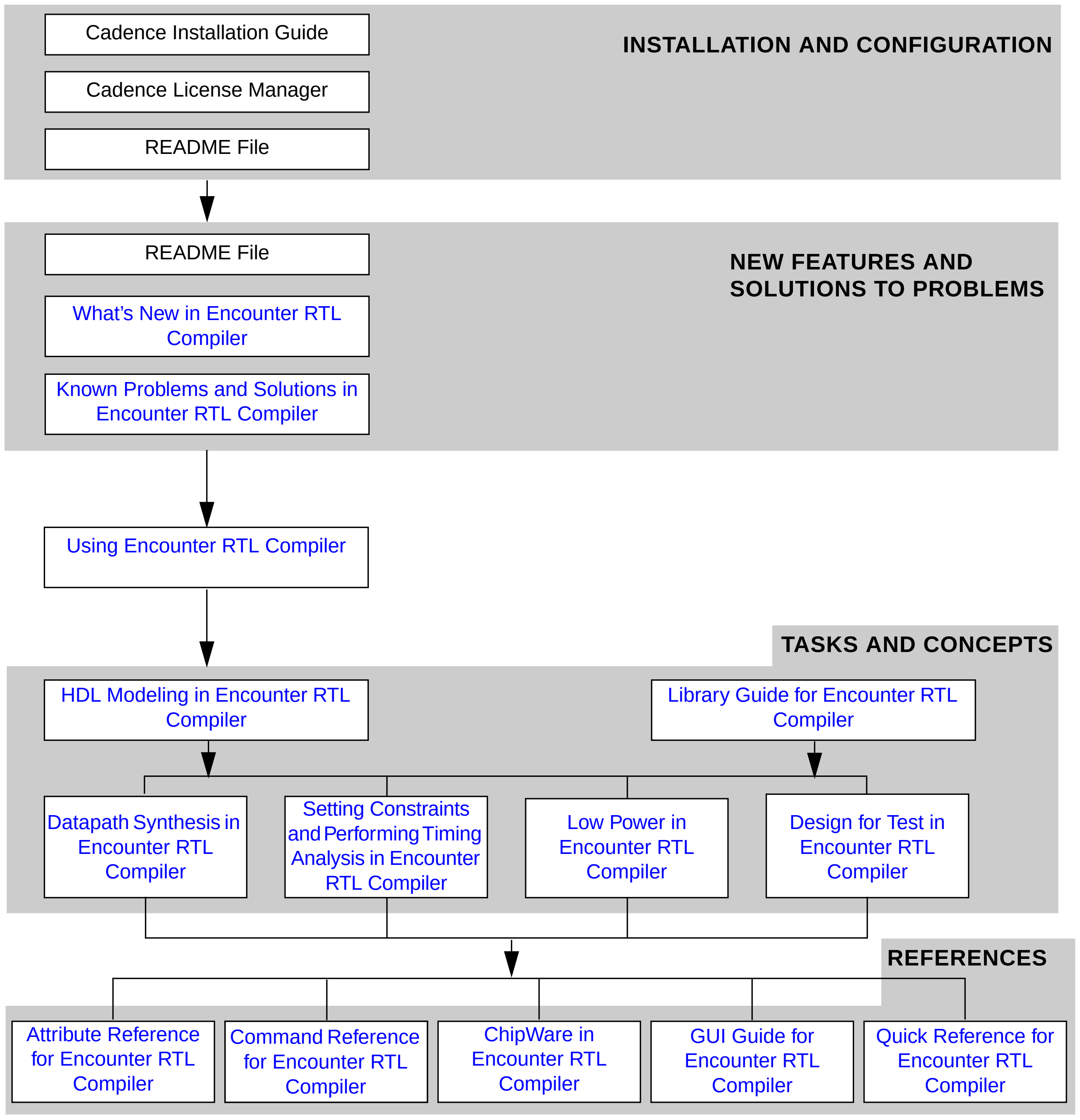
## Additional References

The following sources are helpful references, but are not included with the product documentation:

- TclTutor, a computer aided instruction package for learning the Tcl language: http://www.msen.com/~clif/TclTutor.html.
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std.1364-1995)
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1364-2001)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1987)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1993)

**Note:** For information on purchasing IEEE specifications go to http://shop.ieee.org/store/ and click on *Standards.*

## How to Use the Documentation Set


Cadence Installation Guide
Cadence License Manager
README File
INSTALLATION AND CONFIGURATION
README File
What's New in Encounter RTL Compiler
Known Problems and Solutions in Encounter RTL Compiler
NEW FEATURES AND SOLUTIONS TO PROBLEMS
Using Encounter RTL Compiler
TASKS AND CONCEPTS
HDL Modeling in Encounter RTL Compiler
Library Guide for Encounter RTL Compiler
Datapath Synthesis in Encounter RTL Compiler
Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler
Low Power in Encounter RTL Compiler
Design for Test in Encounter RTL Compiler
REFERENCES
Attribute Reference for Encounter RTL Compiler
Command Reference for Encounter RTL Compiler
ChipWare in Encounter RTL Compiler
GUI Guide for Encounter RTL Compiler
Quick Reference for Encounter RTL Compiler

## Reporting Problems or Errors in Manuals

The Cadence Online Documentation System, CDSDoc, lets you view, search, and print Cadence product documentation. You can access CDSDoc by typing `cdsdoc` from your Cadence tools hierarchy.

Clicking the *Feedback* button lets you send e-mail directly to Cadence Technical Publications. Use it if you find:

- An error in the manual
- An omission of information in a manual
- A problem displaying documents

## Customer Support

Cadence offers live and online support, as well as customer education and training programs.

### Cadence Online Support

The Cadence® online support website offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give you step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, service request tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

For more information on Cadence online support go to:

http://support.cadence.com

### Other Support Offerings

- **Support centers**—Provide live customer support from Cadence experts who can answer many questions related to products and platforms.
- **Software downloads**—Provide you with the latest versions of Cadence products.
- **Education services**—Offers instructor-led classes, self-paced Internet, and virtual classroom.

- **University software program support**—Provides you with the latest information to answer your technical questions.

For more information on these support offerings go to:

http://www.cadence.com/support

## Messages

From within RTL Compiler there are two ways to get information about messages.

- Use the `report messages` command.

  For example:

  ```
  rc:/> report messages
  ```

  This returns the detailed information for each message output in your current RTL Compiler run. It also includes a summary of how many times each message was issued.

- Use the `man` command.

  **Note:** You can only use the `man` command for messages within RTL Compiler.

  For example, to get more information about the "TIM-11" message, type the following command:

  ```
  rc:/> man TIM-11
  ```

If you do not get the details that you need or do not understand a message, either contact Cadence Customer Support to file a PCR or email the message ID you would like improved to:

rc_msg_improvement@cadence.com

## Man Pages

In addition to the Command and Attribute References, you can also access information about the commands and attributes using the man pages in RTL Compiler. Man pages contain the same content as the Command and Attribute References. To use the man pages from the UNIX shell:

1. Set your environment to view the correct directory:

   ```
   setenv MANPATH $CDN_SYNTH_ROOT/share/synth/man
   ```

2. Enter the name of the command or attribute that you want either in RTL Compiler or within the UNIX shell. For example:

   - `man check_dft_rules`
   - `man cell_leakage_power`

You can also use the `more` command, which behaves like its UNIX counterpart. If the output of a manpage is too small to be displayed completely on the screen, use the `more` command to break up the output. Use the spacebar to page forward, like the UNIX `more` command.

```
rc:/> more man synthesize
```

## Command-Line Help

You can get quick syntax help for commands and attributes at the RTL Compiler command-line prompt. There are also enhanced search capabilities so you can more easily search for the command or attribute that you need.

**Note:** The command syntax representation in this document does not necessarily match the information that you get when you type `help` *`command_name`*. In many cases, the order of the arguments is different. Furthermore, the syntax in this document includes all of the dependencies, where the help information does this only to a certain degree.

If you have any suggestions for improving the command-line help, please e-mail them to:

synthesis_help@cadence.com

### Getting the Syntax for a Command

Type the `help` command followed by the command name.

For example:

```
rc:/> help path_delay
```

This returns the syntax for the `path_delay` command.

## Getting the Syntax for an Attribute

Type the following:

```
rc:/> get_attribute attribute_name * -help
```

For example:

```
rc:/> get_attribute max_transition * -help
```

This returns the syntax for the `max_transition` attribute.

## Searching for Attributes

You can get a list of all the available attributes by typing the following command:

```
rc:/> get_attribute * * -help
```

You can type a sequence of letters after the `set_attribute` command and press `Tab` to get a list of all attributes that contain those letters.

```
rc:/> set_attr li
ambiguous "li": lib_lef_consistency_check_enable lib_search_path libcell
liberty_attributes libpin library library_domain line_number
```

## Searching For Commands When You Are Unsure of the Name

You can use help to find a command if you only know part of its name, even as little as one letter.

- If you only know the first few letters of a command you can get a list of commands that begin with that letter.

  For example, to get a list of commands that begin with “ed”, you would type the following command:

  ```
  rc:/> ed* -h
  ```

- You can type a single letter and press `Tab` to get a list of all commands that contains that letter.

  For example:

  ```
  rc:/> c <Tab>
  ```

This returns the following commands:

```
ambiguous "c": cache_vname calling_proc case catch cd cdsdoc change_names
check_dft_rules chipware clear clock clock_gating clock_ports close cmdExpand
command_is_complete concat configure_pad_dft connect_scan_chains continue
cwd_install ...
```

- You can also type a sequence of letters and press `Tab` to get a list of all commands that contain those letters.

  For example:

  ```
  rc:/> path_ <Tab>
  ```

  This returns the following commands:

  ```
  ambiguous "path_": path_adjust path_delay path_disable path_group
  ```

## Documentation Conventions

### Text Command Syntax

The list below defines the syntax conventions used for the RTL Compiler text interface commands.

| | |
|---|---|
| `literal` | Nonitalic words indicate keywords you enter literally. These keywords represent command or option names. |
| *`arguments and options`* | Words in italics indicate user-defined arguments or information for which you must substitute a name or a value. |
| \| | Vertical bars (OR-bars) separate possible choices for a single argument. |
| `[ ]` | Brackets indicate optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one. |
| `{ }` | Braces indicate that a choice is required from the list of arguments separated by OR-bars. Choose one from the list. `{ argument1 \| argument2 \| argument3 }` |
| **{ }** | Braces, used in Tcl commands, indicate that the braces must be typed in. |

| | |
|---|---|
| `...` | Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, `[argument]...`), you can specify zero or more arguments. If the three dots are used without brackets (`argument...`), you must specify at least one argument. |
| # | The pound sign precedes comments in command files. |

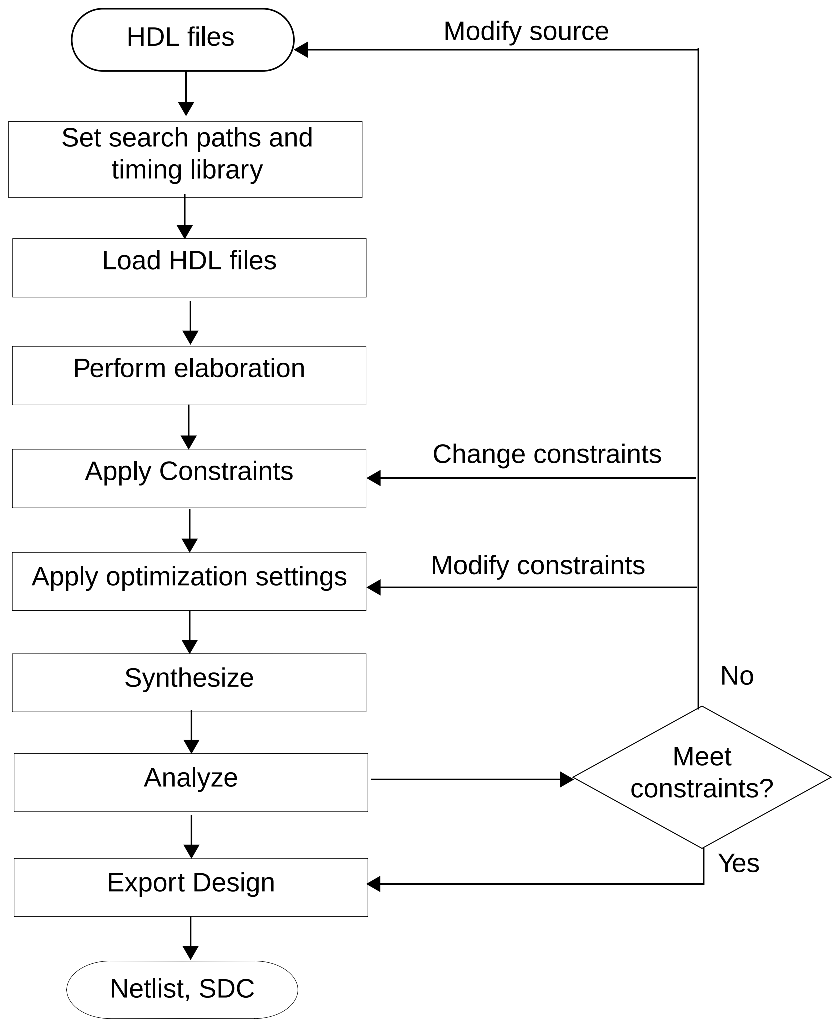
1

# Getting Started with the Generic Flow

## Overview

Figure 1-1 on page 18 shows the generic RTL Compiler work flow. The term “generic” merely illustrates that whatever flow you use, you will most likely use most or all of the steps in the generic flow. This section briefly and sequentially describes all the tasks within the work flow.

**Figure 1-1 Generic RTL Compiler Work Flow**



## Tasks

## Starting RTL Compiler

The `rc` command starts RTL Compiler from the UNIX environment. The syntax of the `rc` command is:

```
rc [-32] [-64] [-cmdfile string] [-execute command] [-files script_file]
[-logfile log_file_name][-lsf_cpu integer] [lsf_queue queue_name] [-no_custom]
[-unique} [-nologfile] [queue] [-use_license RTL_Compiler_L | RTL_Compiler_Ultra
|RTL_Compiler_Verification | First_Encounter_GXL | SOC_Encounter_GXL | FE_GPS |
SOC_Encounter_GPS | Virtuoso_Digital_Implem | Virtuoso_Digital_Implement] [-vdi]
[-version]
```

To invoke RTL Compiler:

➤ Type the following command at the UNIX prompt to launch RTL Compiler in 32-bit mode with the `RTL_Compiler_Ultra` license:

```
unix>  rc -32
```

Alternatively, you can just type the `rc` command because the 32-bit mode is the default mode (regardless of the platform):

```
unix> rc
```

➤ You can specify the logfile name with the `-logfile` option. The default name is `rc.log` if there is no other logfile in the directory from which RTL Compiler is launched.

```
unix> rc -logfile pov.log
```

Do not use the UNIX `tee` command and pipe (`|`) to specify your logname: doing so would not allow you to use the `control-c` key sequence to gracefully exit a process like incremental optimization.

➤ Type the following command if you want the launching of RTL Compiler to fail if no `RTL_Compiler_Ultra` license is available:

```
unix> rc -use_license RTL_Compiler_Ultra
```

If you specify multiple licenses, only the last one will be used.

➤ The following commands have the same effect. Therefore, you should use one or the other and not both in conjunction:

```
unix> rc -use_license Virtuoso_Digital_Implem
```

is the same as:

```
unix> rc -vdi
```

If you specify the `-vdi` option, RTL Compiler will first try to use the `Virtuoso_Digital_Implem` license. If that license is unavailable, it will then use the `Virtuoso_Digital_Implement` license.

➤ Type the following command at the UNIX prompt to launch RTL Compiler in 64-bit mode:

```
unix> rc -64
```

The initial splash screen will tell you whether you are in 64 or 32 bit mode.

**Note:** You can set the `CDS_AUTO_64BIT` environment variable to `ALL` (`setenv CDS_AUTO_64BIT ALL`) to launch not only RTL Compiler, but all Cadence tools in 64 bit. You will not need to specify the `-64` option if you use this variable.

➤ Type the following command to simultaneously invoke RTL Compiler as a background process and execute a script:

```
unix> rc < script_file_name &
```

➤ Type the following command to simultaneously invoke RTL Compiler, execute script, and exit if any problems are encountered with the script:

```
unix> rc -files script_file_name < /dev/null
```

➤ Type the following command to simultaneously set the script search path and invoke RTL Compiler:

```
unix> rc -execute “set_attribute script_search_path pathname”
```

➤ Type the following command to simultaneously set the a Tcl variable, invoke RTL Compiler, and launch a script:

```
unix> rc -files script_file_name -execute “set variable_name value”
```

➤ RTL Compiler supports super-threading on LSF. Use the `-lsf_cpus` option to specify the number of processes to send to LSF and the `-lsf_queue` option to specify a particular LSF queue:

```
unix> rc -lsf_cpus 4 -lsf_queue teagan_queue
```

If the `super_thread_servers` and `bsub_options` attributes are specified within a RTL Compiler session, they will override the `-lsf_cpus` and `-lsf_queue` options. In the following example, two processes will be sent to LSF and the `stormy queue` used:

```
unix> rc -lsf_cpus 4 -lsf_queue teagan_queue
...
rc:/> set_attribute super_thread_servers {lsf lsf}
rc:/> set_attribute bsub_options stormy_queue
```

*Tip*

You can abbreviate the options for the `rc` command as long as there are no ambiguities with other options. In the following example, `-ver` would imply the `-version` option:

```
unix> rc -ver
```

Just using `rc -v` would not work because there is more than one option that starts with the letter “v.”

## Generating Log Files

By default, RTL Compiler generates a log file named `rc.log`. The log file contains the entire output of the current RTL Compiler session. You can set the level of verbosity in the log file with the `information_level` attribute, as described in Setting Information Level and Messages on page 23.

You can customize the log file name while invoking RTL Compiler or during the synthesis session. The following examples simultaneously customize the log file name and execute a script file. RTL Compiler will overwrite any log file with the same name.

➤ Start RTL Compiler with the `-logfile` option:

```
unix> rc -f script_file_name -logfile log_file_name
```

➤ Start RTL Compiler as a background process and write out the log file:

```
unix> rc < script_file_name > log_file_name &
```

➤ Suppress the generation of any log file by using the `-nologfile` option when invoking RTL Compiler.

```
unix> rc -f script_file_name -nologfile
```

➤ Customizes the log file within an RTL Compiler session through the `stdout_log` attribute:

```
rc:/> set_attribute stdout_log log_file_name
```

➤ If a log file already exists, the new log file will have the number “`1`” appended to it.

## Generating the Command File

By default, RTL Compiler generates a command history file named `rc.cmd`, which contains a record of all the commands that were issued in a particular session. This file is created in addition to the log file.

To customize the command file name, use the `command_log` attribute within a RTL Compiler session. The following example changes the default name of `rc.cmd` to `rc_command_list.txt`:

```
rc:/> set_attribute command_log rc_command_list.txt
```

If a command file already exists, the new command file will have the number “`1`” appended to it.

## Setting Information Level and Messages

You can control the amount of information RTL Compiler writes out in the output logfiles.

➤ To specify the verbosity level, type the following command:

```
rc:/> set_attribute information_level value /
```

where `value` is an integer value between `0` (minimum) and `9` (maximum). The recommended level is `6`. The `information_level` attribute is a root-level attribute. Therefore, like all root level attributes, it needs to be set on the root level ("`/`") like the above example.

*Tip*

For analysis and debugging, set the information level to `9`.

## Specifying Explicit Search Paths

You can specify the search paths for libraries, scripts, and HDL files. The default search path is the directory in which RTL Compiler is invoked.

To set the search paths, type the following `set_attribute` commands:

```
rc:/> set_attribute lib_search_path path /
rc:/> set_attribute script_search_path path /
rc:/> set_attribute hdl_search_path path /
```

where *`path`* is the full path of your target library, script, or HDL file locations.

The slash ( "`/`" ) in these commands refers to the root-level RTL Compiler object that contains all global RTL Compiler settings.

## Setting the Target Technology Library

After you set the library search path, you need to specify the target technology library for synthesis using the `library` attribute.

➤ To specify a single library:

```
rc:/> set_attribute library lib_name.lbr /
```

RTL Compiler will use the library named `lib_name.lbr` for synthesis. RTL Compiler can also accommodate the `.lib` (Liberty) library format. In either case, ensure that you specify the library at the root-level ("`/`").

**Note:** If the library is not in a previously specified search path, specify the full path, as follows:

```
rc:/> set_attribute library /usr/local/files/lib_name.lbr
```

➤ To specify a single library compressed with gzip:

```
rc:/> set_attribute library lib_name.lbr.gz /
```

➤ To append libraries:

```
rc:/> set_attribute library {{lib1.lib lib2.lib}}
```

After `lib1.lib` is loaded, `lib2.lib` is appended to `lib1.lib`. This appended library retains the `lib1.lib` name.

## Setting the Appropriate Synthesis Mode

RTL Compiler has two modes to synthesize the design. The synthesis mode is determined by the setting of the `interconnect_mode` attribute:

- `wireload` (default) indicates to use wire-load models to drive synthesis
- `ple` indicates to use Physical Layout Estimators (PLEs) to drive synthesis

  PLE uses physical information, such as LEF libraries, to provide better closure with back-end tools.

  **Note:** When you read in LEF files, the `interconnect_mode` attribute is automatically set to `ple`.

## Loading the HDL Files

Use the `read_hdl` command to read HDL files into RTL Compiler. When you issue a `read_hdl` command, RTL Compiler reads the files and performs syntax checks.

➤ To load one or more Verilog files:

- ❑ You can read the files sequentially:

  ```
  read_hdl file1.v
  read_hdl file2.v
  read_hdl file3.v
  ```

- ❑ Or you can load the files simultaneously:

  ```
  read_hdl { file1.v file2.v file3.v }
  ```

*Caution*

***Your files may have extra, hidden characters (for example, line terminators) if they are transferred from Windows/Dos to the UNIX environment using the dos2unix command. Be sure to eliminate them because RTL Compiler will issue an error when it encounters these characters.***

For more information on loading HDL files, see *Loading Files*.

## Performing Elaboration

Elaboration is only required for the top-level design. The `elaborate` command automatically elaborates the top-level design and all of its references. During elaboration, RTL Compiler performs the following tasks:

- Builds data structures
- Infers registers in the design
- Performs high-level HDL optimization, such as dead code removal
- Checks semantics

**Note:** If there are any gate-level netlists read in with the RTL files, RTL Compiler automatically links the cells to their references in the technology library during elaboration. You do not have to issue an additional command for linking.

At the end of elaboration, RTL Compiler displays any unresolved references (immediately after the key words `Done elaborating`):

```
Done elaborating '<top_level_module_name>'.
Cannot resolve reference to <ref01>
Cannot resolve reference to <ref02>
Cannot resolve reference to <ref03>
...
```

After elaboration, RTL Compiler has an internally created data structure for the whole design so you can apply constraints and perform other operations.

For more information on elaborating a design, see *Elaborating the Design*.

## Applying Constraints

After loading and elaborating your design, you must specify constraints. The constraints include:

- Operating conditions
- Clock waveforms
- I/O timing

You can apply constraints in several ways:

- Type them manually in the RTL Compiler shell.
- Include a constraints file.
- Read in SDC constraints.

*Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler* gives a broader overview on constraints.

## Applying Optimization Constraints

In addition to applying design constraints, you may need to use additional optimization strategies to get the desired performance goals from synthesis.

With RTL Compiler, you can perform any of the following optimizations:

- Remove designer-created hierarchies (ungrouping)
- Create additional hierarchies (grouping)
- Synthesize a sub-design
- Create custom cost groups for paths in the design to change the synthesis cost function

For example, the timing paths in the design can be classified into the following four distinct cost groups:

- Input-to-Output paths (I2O)
- Input-to-Register paths (I2C)
- Register-to-Register (C2C)
- Register-to-Output paths (C2O)

For each path group, the worst timing path drives the synthesis cost function. For more information on optimization strategies and related commands, see *Defining Optimization settings*.

## Performing Synthesis

After the constraints and optimizations are set for your design, you can proceed with synthesis by issuing the `synthesize` command.

➤ To synthesize your design using the `synthesize` command, type the following command:

```
rc:\> synthesize -to_mapped
```

For details on the `synthesize` command, see *Performing Synthesis*.

After synthesis, you will have a technology-mapped gate-level netlist.

## Analyzing the Synthesis Results

After synthesizing the design, you can generate detailed timing and area reports using the various `report` commands:

- To generated a detailed area report, use `report area`.
- To generate a detailed gate selection and area report, use `report gates`.
- To generate a detailed timing report, including the worst critical path of the current design, use `report timing`.

For more information on generating reports for analysis, see *Generating Reports* and “Analysis Commands” in the *Command Reference for Encounter RTL Compiler*.

## Writing Out Files for Place and Route

The last step in the flow involves writing out the gate-level netlist, SDC, or Encounter configuration file for processing in your place and route tool. For more information on this topic, see *Interfacing to Place and Route*.

**Note:** By default, the `write` commands write output to `stdout`. If you want to save your information to a file, use the redirection symbol (>) and a filename.

➤ To write the gate-level netlist, use the `write_hdl` command.

Because `write_hdl` directs the output to `stdout`, use file redirection to create a design file on disk, as shown in the following example:

```
rc:/> write_hdl > design.v
```

This command writes out the gate-level netlist to a file called `design.v`.

➤ To write out the design constraints, use the `write_script` command, as shown in the following example:

```
rc:/> write_script > constraints.g
```

This command writes out the constraints to the file `constraints.g`.

➤ To write the design constraints in SDC format, use the `write_sdc` command, as shown in the following example:

```
rc:/> write_sdc > constraints.sdc
```

This command writes the design constraints to a file called `constraints.sdc`.

**Note:** Because some place and route tools require different structures in the netlist, you may need to make some adjustments either before synthesis or before writing out the netlist. For more information about these issues, see *Interfacing to Place and Route*.

➤ To write the Encounter configuration file, use the `write_encounter` command:

```
rc:/> write_encounter design design_name
```

## Exiting RTL Compiler

There are three ways to exit RTL Compiler when you finish your session:

- Use the `quit` command.
- Use the `exit` command.
- Use the `Control-c` key combination twice in succession to exit the tool immediately.

## Summarizing the Generic Flow

```
set_attribute library library_name
read_hdl filename
elaborate
read constraints
synthesize -to_mapped
write_hdl
read_def def_file
synthesize -to_mapped
predict_qos
write_encounter design
```

2

# Path Adjust Flows

## Overview

The following figures show two Path Adjust flows: the RTL Compiler Path Adjust flow and the Encounter Path Adjust flow.

Figure 2-1 on page 32 shows the RTL Compiler Path Adjust Flow. It is called the RTL Compiler Path Adjust Flow because it starts from within RTL Compiler and uses its `export_critical_endpoints` command. For more information about the `export_critical_endpoints` command, refer to the Command Reference for Encounter RTL Compiler.

**Figure 2-1 RTL Compiler Path Adjust Flow**

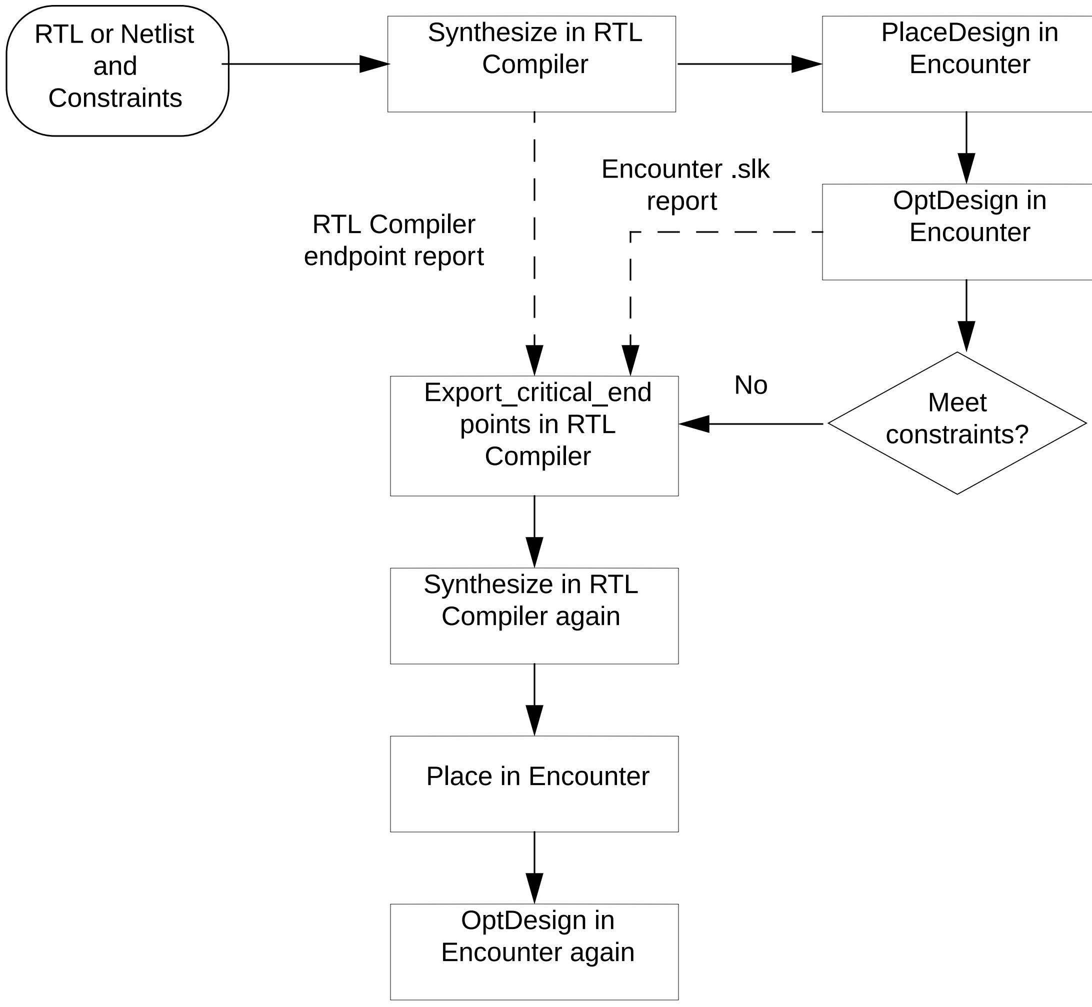
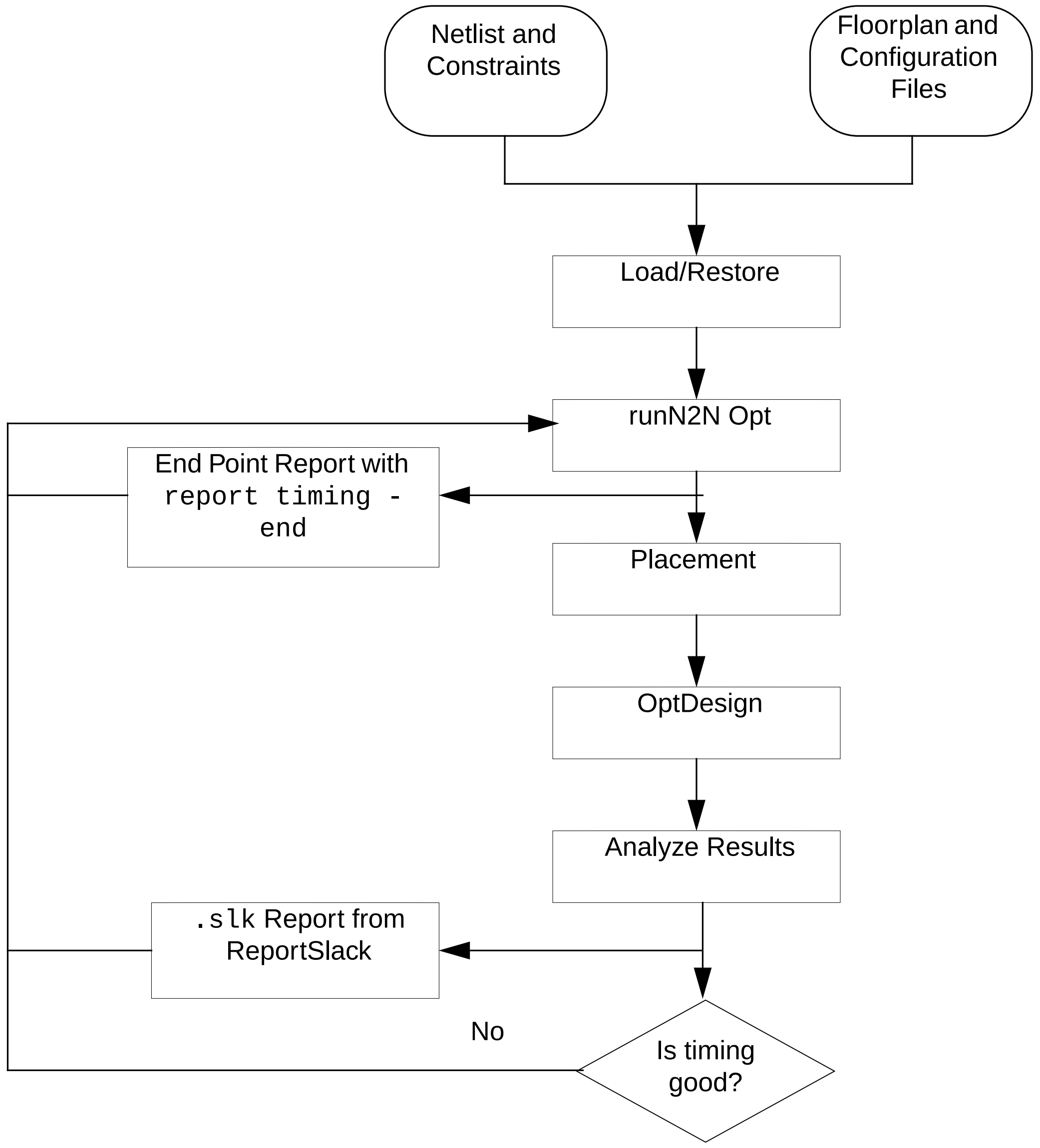

RTL or Netlist and Constraints
Synthesize in RTL Compiler
PlaceDesign in Encounter
Encounter .slk report
OptDesign in Encounter
RTL Compiler endpoint report
Export_critical_end points in RTL Compiler
No
Meet constraints?
Synthesize in RTL Compiler again
Place in Encounter
OptDesign in Encounter again

Figure 2-2 on page 33 shows the Encounter Path Adjust Flow. It is called the Encounter Path Adjust Flow because it starts from within Encounter and uses its `runN2NOpt` command.

**Figure 2-2 Encounter Path Adjust Flow**


Netlist and Constraints
Floorplan and Configuration Files
Load/Restore
runN2N Opt
End Point Report with `report timing -end`
Placement
OptDesign
Analyze Results
`.slk` Report from ReportSlack
No
Is timing good?

## Tasks

- RTL Compiler Path Adjust Flow on page 34
- Encounter Path Adjust Flow on page 36

### RTL Compiler Path Adjust Flow

Timing closure between RTL Compiler and Encounter uses an automated path adjust flow. The path adjust flow is for those scenarios in which you want to improve QoS after a you have taken your design through a RTL Compiler synthesis run, placement and optimization in Encounter, and the timing goals are not met.

1. Take the timing endpoint report generated from the RTL Compiler synthesis run. The following example generates a report named `rc_endpoint.rpt`:

   ```
   report timing -end > rc_endpoint.rpt
   ```

2. Use the slack report (`.slk`) from the Encounter run. The following example generates a report named `top.slk`:

   ```
   reportSlacks -outfile top.slk
   ```

3. The endpoint reports (`rc_endpoint.rpt` and `top.slk` in the examples above) can be passed to the `export_critical_endpoints` RTL Compiler command to generate a path adjust file. The following example creates a path adjust file called `pathadjust.tcl`:

   ```
   export_critical_endpoints -rc_file rc_endpoint.rpt -fe_file top.slk > \
       pathadjust.tcl
   ```

4. Run a second RTL Compiler synthesis run using the path adjust file (`pathadjust.tcl` in the example above) as a additional set of constraint on the original RTL or netlist.

5. Load the newly created netlist into Encounter and take it through placement and optimization for better QoS.

Again, Figure 2-1 on page 32 illustrates the steps above.

All exceptions created by the automatic path adjust flow will have the prefix `RCFE_PA`. The following RTL Compiler command will give all the exceptions created by the automatic path adjust flow:

```
rc:\> find / -exception RCF_PA*
```

**Note:** The path adjust file that was created using `-rtl` option of the `export_critical_endpoints` will not be applicable on the RTL if the netlist has undergone hierarchy changes due to grouping or ungrouping.

### Example 2-1 Example Script Starting from RTL

```
set_attribute library libraries
read_hdl RTL files
elaborate
read_sdc constraint files
export_critical_endpoints  -rc_file rc_endpoint.rpt -fe_file top.slk -rtl > \
    rtl_pathadjust.tcl
source rtl_pathadjust.tcl   ?------------------Apply the pathadjust file created
synthesize -to_generic -effort level
synthesize -to_map -effort level
rm [find / -exception RCFE_PA*] ?------------Remove the created before reporting.
report timing
write_hdl > netlist.v
```

### Example 2-2 Example Script Starting from a Netlist

```
set_attribute library libraries
read_netlist RTL files
read_sdc <constraint files>
export_critical_endpoints  -rc_file rc_endpoint.rpt -fe_file top.slk  > \
    pathadjust.tcl
source pathadjust.tcl   ?------------------Apply the pathadjust file created
synthesize -to_generic -effort level
synthesize -to_map -effort level
rm [find / -exception RCFE_PA*] ?------------Remove the created before reporting.
report timing
write_hdl > netlist.v
```

## Encounter Path Adjust Flow

Encounter's `runN2NOpt` command supports the automatic path adjust flow from version 7.1. The `runN2NOpt` command launches RTL Compiler from Encounter and automatically runs a RTL Compiler synthesis session based on the options and RTL Compiler directives specified as options. The RTL Compiler endpoint report and Encounter slack report will be accepted as options to the `runN2NOpt` command and will enable the `export_critical_endpoints` command in the RTL Compiler flow script that is written out.

**Note:** For a detailed documentation & supported options/settings refer to SOCE documentation.

Figure 2-2 on page 33 illustrates the automatic path adjust flow using Encounter's `runN2NOpt` command. After an initial pass in Encounter or Encounter with `runN2Nopt` the RTL Compiler endpoint and Encounter slack reports can be loaded through the `runN2NOpt` command. This enables the `autopathadjust` flow during the netlist-to-netlist optimization performed in RTL Compiler from Encounter.

**Note:** If only the Encounter slack report is available and an initial RTL Compiler synthesis run was not performed, the `runN2NOpt` command will generate a RTL Compiler endpoint report automatically when it launches RTL Compiler. For third party netlists, run an incremental synthesis before generating an endpoint report using the `-incrFirst` *`effort`* option.

The following steps describe the Encounter Path Adjust Flow

1. Load or restore the Encounter session with the netlist, constraints, floorplan and configuration file.
2. Enable the `runN2NOpt` command with the automatic path adjust options and provide the RTL Compiler endpoint & `.slk` reports. In the following example the RTL Compiler endpoint report is `rc_endpoint.rpt` and the Encounter slack report is `top.slk`. The `-autoPathAdjust`'option enables the automatic path adjust flow and the `-backendReport` and `-frontEndReport` options accept the Encounter slack & RTL Compiler endpoint reports. respectively.

```
encounter> runN2NOpt -autoPathAdjust -backEndReport top.slk \
    -frontEndReport rc_endpoint.rpt
```

3. The `runN2NOpt` command loads the newly optimized netlist, which went through automatic path adjust into Encounter after a RTL Compiler synthesis session.
4. This netlist can be taken through placement and optimization for better QOS

Again, Figure 2-2 on page 33 illustrates the steps above.

If only the Encounter slack report is available and a initial RTL Compiler synthesis run was not performed, the `runN2NOpt` command can be specified in the following forms.

- The following example will create a RTL Compiler endpoint report automatically because only the Encounter slack report is specified:

  ```
  encounter> runN2NOpt -autoPathAdjust -backEndReport top.slk
  ```

- The following example will run a low effort incremental synthesis before creating a endpoint report for path adjust:

  ```
  encounter> runN2NOpt -autoPathAdjust -backEndReport top.slk -incrFirst low
  ```

The following table shows the `runN2NOpt` options for automatic path adjust and the corresponding `export_critical_endpoints` options that will be enabled.

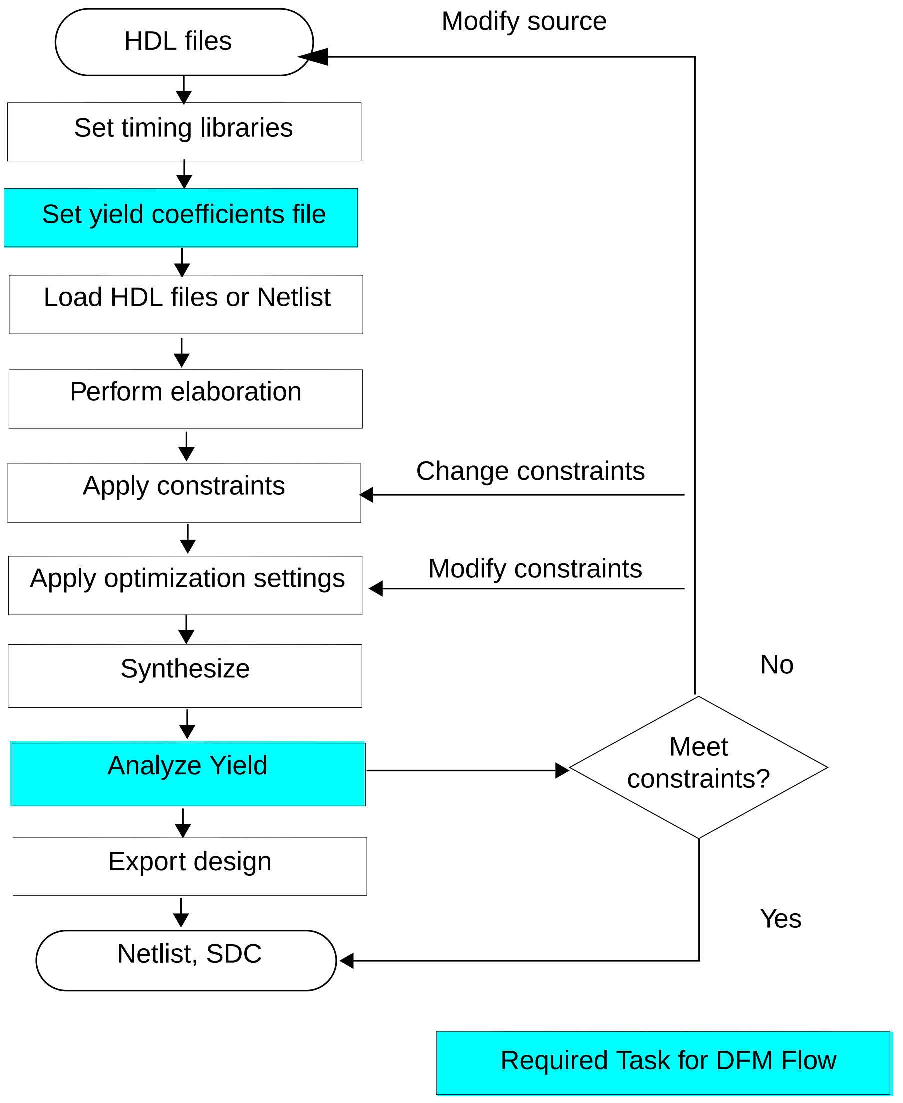
| **runN2NOpt** | **export_critical_endpoints** |
|---|---|
| `-autoPathAdjust` | n/a |
| `-fronEndReport` | `-rc_file` |
| `-backEncReport` | `-fe_file` |
| `-noGroup` | `-no_group` |
| `-group` | `-group` |
| `-percentageEndpoints` | `-percentage_of_endpoints` |

3

# Design For Manufacturing Flow

## Overview

RTL Compiler allows you to perform Design for Manufacturing (DFM) optimizations and discover yield information in the DFM flow. During synthesis, RTL Compiler estimates the probability for library cell failure and computes the overall impact on “defect-limited yield” for the whole design. While keeping this impact as a global cost function, RTL Compiler picks cells which have the best combination of timing, area, probability of cell failure, and power during logic structuring.



## Tasks

The tasks below list only those that are different from the generic flow or illustrate a new step.

- Specifying the Yield Coefficients Information on page 41
- Setting the DFM Flow on page 41
- Analyzing the Yield Information on page 41

### Specifying the Yield Coefficients Information

The yield coefficients file provides the probability for failure of each library cell. The cell failure rates are typically characterized by some library analysis methods based on the cell layout and fabrication yield characterization data.

To load the yield coefficients file, use the `read_dfm` command:

```
rc:/> read_dfm penny.dfm
```

RTL Compiler will annotate the defect probability of any matching cells between the coefficients file and the timing library.

If you have multiple coefficients file, use the `read_dfm` command for each file:

```
rc:/> read_dfm penny.dfm
rc:/> read_dfm flame.dfm
```

### Setting the DFM Flow

Before synthesis, you must set RTL Compiler into yield synthesis mode through the `optimize_yield` attribute.

```
rc:/> set_attribute optimize_yield true /
```

The default value of this attribute is `false`.

### Analyzing the Yield Information

After synthesizing the design to gates, find the yield cost and yield percentage for each instance with the `report yield` command:

```
rc:/> report yield

Instance     Cells   Cell Area            Cost     Yield %
```

```
---------------------------------------------------------
cpu            470         659    1.600606e-05     99.9984
  alu1         248         283    7.606708e-06     99.9992
  pcount1       65          92    2.215669e-06     99.9998
  ireg1         33          88    1.629471e-06     99.9998
  accum1        33          88    1.629471e-06     99.9998
  decode1       50          67    1.568901e-06     99.9998
```

The command shows the defect-limited yield impact for library cell defects.

To find the yield cost and yield percentage values for each library cell, use the `-yield` option of the `report gates` command:

```
rc:/> report gates -yield

Gate  Instances   Area       Cost       Yield    Library
------------------------------------------------------------
flopdrs        33  264.000  3.39278e-06  99.9997    tutorial
inv1          103   51.500   1.5022e-06  99.9998    tutorial
nand2         315  315.000  1.08311e-05  99.9989    tutorial
nor2           19   28.500  6.79989e-07  99.9999    tutorial
------------------------------------------------------------
total         470  659.000  1.64061e-05  99.9984

   Type    Instances   Area  Area %
------------------------------------
sequential        33 264.000   40.1
inverter         103  51.500    7.8
logic            334 343.500   52.1
------------------------------------
total            470 659.000  100.0
```

Chip will have other effects (for example, systematic and random) that also lower yield. If a given chip had zero defect-limited yield losses, it would still not reach 100% yield due to these other effects. In logic synthesis, RTL Compiler consider only the cell defects.

You must contact the fabrication facility to understand what percentage of failures are due to the cell based defect-limited yield effects. For example, `report yield` may estimate that the yield is 90% while the true yield may only be 80% due to other effects that are not currently modeled. Cell failure rates are given as a simple failure rate per instance of the cell used.

To find the total yield for the design, use the `yield` attribute with the `get_attribute` command. The following example finds the yield for the design `test`:

```
rc:> get_attribute yield [find . -design test]
0.999983594076
```

4

# Associating Dedicated Libraries with Different Blocks in the Design

## Overview

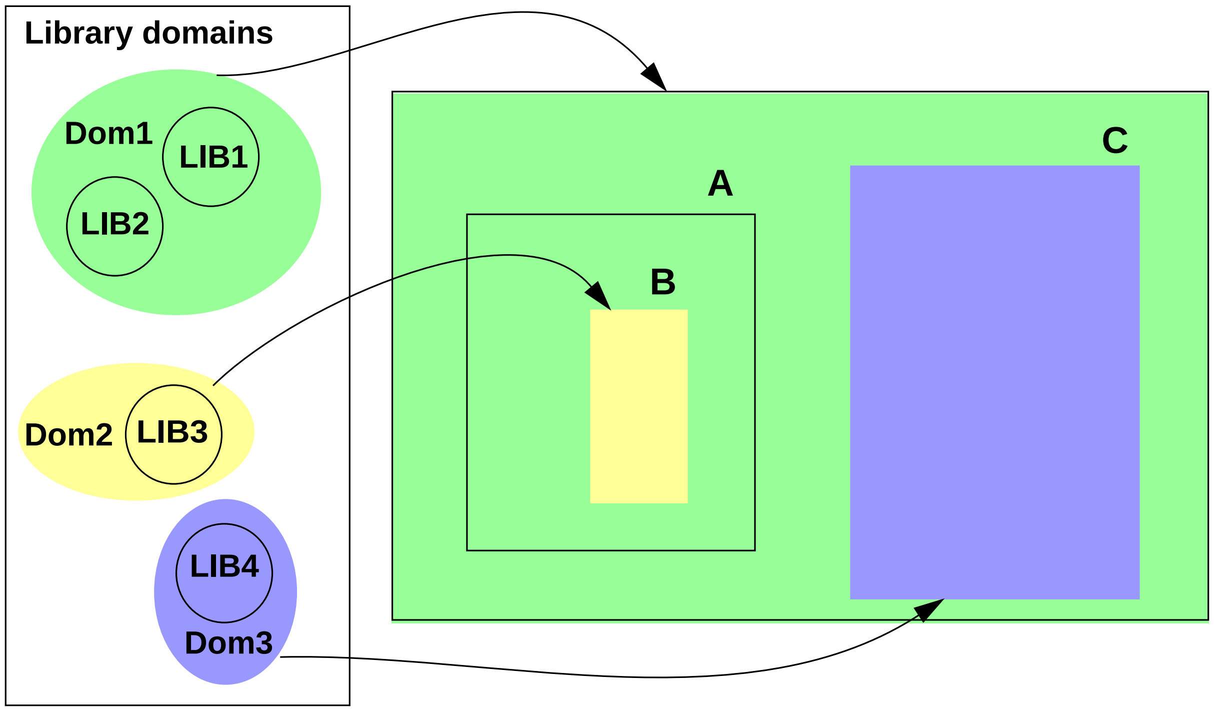
The generic synthesis flow uses the same libraries for the entire design.

You must define library domains to

- **Associate Dedicated Libraries with Different Portions of the Design**

  The design in Figure 4-1 uses three different library sets. You want to use libraries `LIB1` and `LIB2` for the top-level and for block A, library `LIB3` for block B and library `LIB4` for block C.

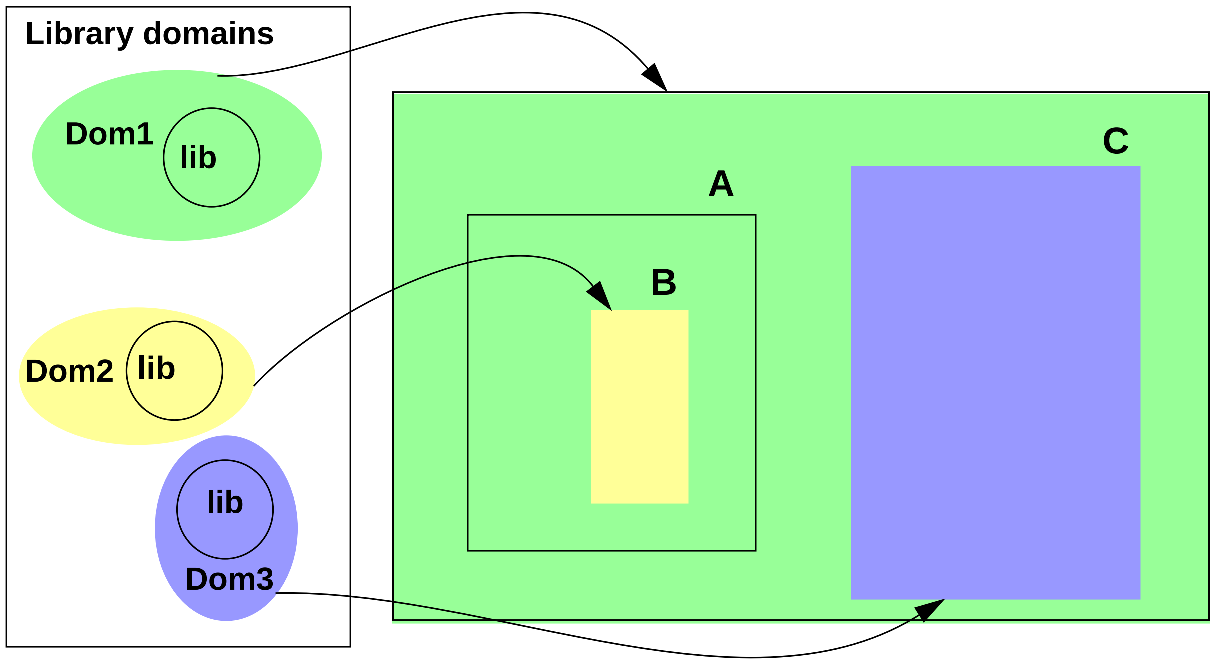
**Figure 4-1 Use of Dedicated Libraries in Single Supply Voltage Design**



- **Target Different Cells from Same Library for Different Portions of Design**

  The design in Figure 4-2 on page 47 uses the same library for the entire design, but you would like to use a limited set of cells to map block B and a different set of cells to map block C, and you allow the use of all cells for the top-level and block A.

**Figure 4-2 Use of Targeted Cells in Single Supply Voltage Design**



This chapter describes the top-down synthesis flow using multiple library domains. The flow is shown in Figure 4-3 on page 48. A script is shown in Example 4-1 on page 49.

Flow Steps describes the steps that are not part of the generic synthesis flow in detail.

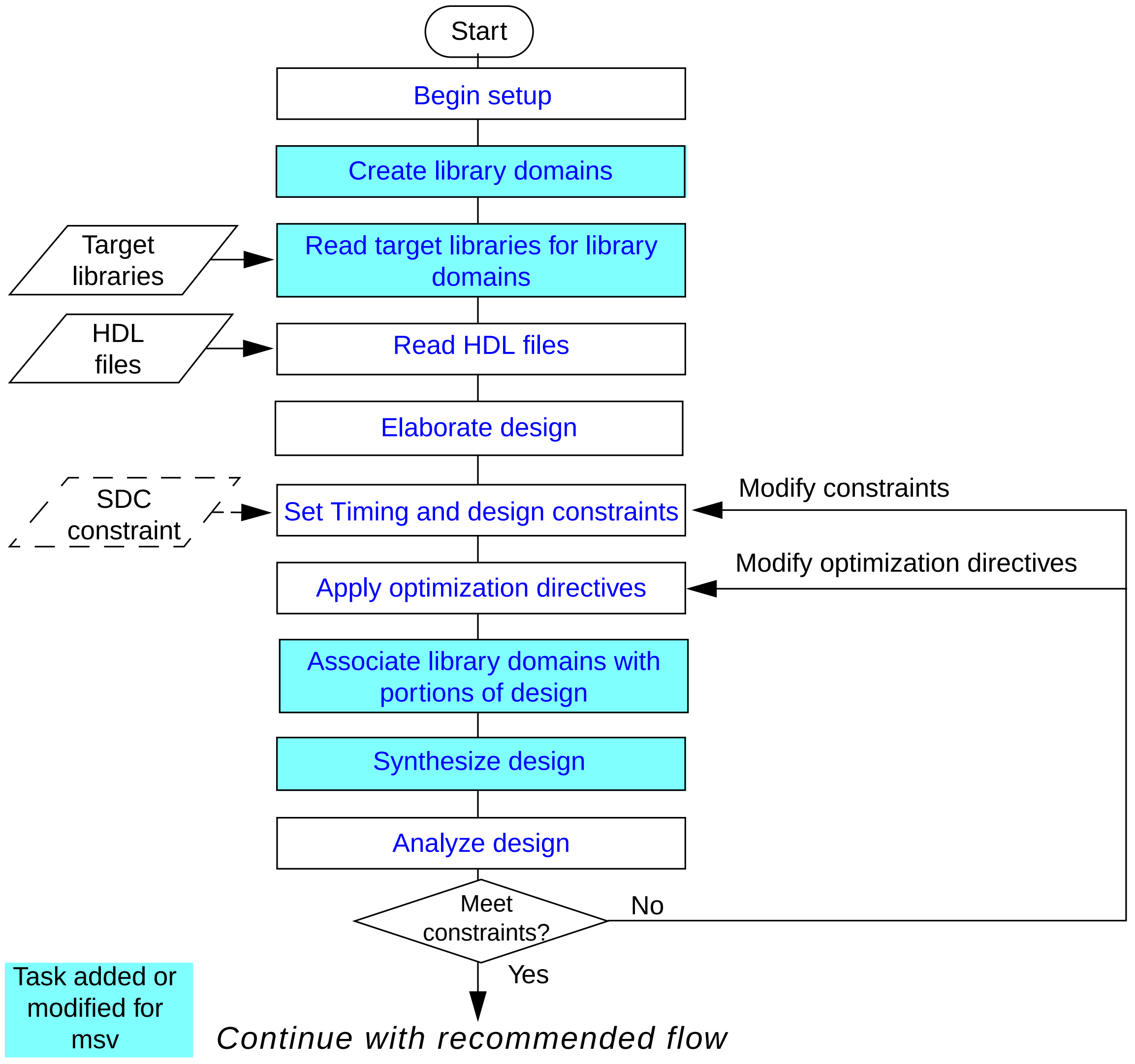
Library Domain Information in the Design Information Hierarchy shows where the library domain information is stored.

When using multiple library domains, you can also perform a what-if analysis to determine which configuration results in better timing. This analysis and some other tasks that are not part of the normal flow covered in Flow Steps are described in Additional Tasks.

*Important*

The flow described in this chapter assumes a *single* supply voltage design. RTL Compiler also uses library domains for *multiple* supply voltage designs. The multiple supply voltage flow is covered in *Low Power in Encounter RTL Compiler.*

**Figure 4-3 Top-Down Synthesis Flow with Multiple Library Domains**


Start
Begin setup
Create library domains
Target libraries
Read target libraries for library domains
HDL files
Read HDL files
Elaborate design
SDC constraint
Set Timing and design constraints
Modify constraints
Apply optimization directives
Modify optimization directives
Associate library domains with portions of design
Synthesize design
Analyze design
Meet constraints?
No
Yes
Task added or modified for msv
Continue with recommended flow

### Example 4-1 Script for Top-Down Synthesis Flow Using Multiple Library Domains

```
# general setup
#-------------
set_attributer lib_search_path ...
set_attribute hdl_search_path ..
# create library domains
#----------------------
create_library_domain domain_list
# specifiy the target libraries for each library domain
#-------------------------------------------------------
set_attribute library library_list1 [find /libraries -library_domain domain1] /
set_attribute library library_list2 [find /libraries -library_domain domain2] /
...
# load and elaborate the design
#-----------------------------
read_hdl design.v
elaborate
# specify timing and design constraints
#---------------------------------------
# specify the following constraints per library domain
#------------------------------------------------------
set_attr operating_conditions string [find /libraries -library_domain domain]
set_attr wireload_selection string [find /libraries -library_domain domain]
# set target library domain for top design
#-----------------------------------------
set_attribute library_domain library_domain design
# set target library domain for blocks
#-------------------------------------
edit_netlist uniquify subdesign
set_attribute library_domain library_domain subdesign
#synthesize the design
#--------------------
synthesize -to_mapped
# analyze design
------------------
report timing
report gates
# export design
#-------------
write_encounter design design] [-basename string] [-gzip_files]
[-reference_config_file file] [-preserve_avoid_cells]
[-ignore_scan_chains] [-floorplan {.def|.pdef|.fp}] [-lef file_list]
```

## Flow Steps

### Begin Setup

For more information on the setup, see

- Generating Log Files
- Generating the Command File
- Setting Information Level and Messages
- Specifying Explicit Search Paths

### Create Library Domains

A library domain is a collection of libraries. You can use library domains to associate *dedicated* libraries or library cells with portions of the design.

➤ To create library domains, use the `create_library_domain` command:

```
create_library_domain domain_list
```

**Note:** There is no limitation on the number of library domains you can create.

*Tip*

Create as many library domains

- ❑ As there are portions in the design for which you want to use dedicated libraries.

  In the example of Figure 4-1 on page 46, you want to use three different sets of libraries, which requires you to create three library domains.

- ❑ As the number of library cell sets that you want to use.

  In the example of Figure 4-2 on page 47, you want to use three different cell sets, so you need to create three library domains.

This command returns the directory path to the library domains that it creates. You need this information when loading in the target libraries.

To get meaningful timing results, all libraries *should* have been characterized for the same nominal operating conditions. If the libraries have different operating conditions, the nominal operating conditions of the last library will be used and thus the last library also determines the voltage of the library domain. For this flow the voltage of all library domains *must* be the same.

For example, to find the active operating condition of a specific domain, use

```
get_att active_operating_conditions [find / -library_domain lib*/*domains/domain]
```

To find the voltage of a domain, you need use the active operating condition of the domain:

```
get_att voltage [get_att active_operating_conditions \
[find / -library_domain lib*/library_domains/domain]]
```

*Tip*

You can always rename or remove a library domain. For more information on removing library domains, refer to Removing a Library Domain.

## Read Target Libraries for Library Domains

Next, you need to associate the libraries with the library domains.

To read in the libraries for a specific library domain, set the `library` attribute for the corresponding domain:

```
set_attribute library library_list [find /libraries -library_domain domain]
```

**Note:** There is no limitation on the number of libraries you can read in per domain.

*Tip*

When targeting different cells from the same library for different portions of design, you need to

**a.** Read in the same library for each library domain:

```
set_attribute library lib [find /libraries -library_domain dom1]
set_attribute library lib [find /libraries -library_domain dom2]
set_attribute library lib [find /libraries -library_domain dom3]
```

**b.** Exclude library cells.

Because library `lib` is used in three library domains, RTL Compiler treats it as if three different libraries were read in. This allows to exclude cells in one library domain, while you allow the use of them in a different library domain.

Within each library domain, you can exclude the library cells that the mapper should not use. To exclude a cell you can use the `avoid` attribute. For example:

```
set_attribute avoid true /lib*/library_domains/dom2/lib/libcells/AO22X2M
set_attribute avoid true [find /*/dom2/lib -libcell AO22X2M]
```

The first library domain for which you read in the libraries, becomes the default library domain.

➤ To change the default library domain, set the following attribute:

```
set_attribute default true desired_library_domain
```

For more information on the use of the default library domain, refer to Removing a Library Domain.

## Read HDL Files

For more information on reading HDL files, see Loading Files in *Using Encounter RTL Compiler.*

## Elaborate Design

For more information on elaborating a design, see Performing Elaboration in *Using Encounter RTL Compiler*.

## Set Timing and Design Constraints

Except for the following constraints, which *should* be set per library domain, all other (SDC and RC native) constraints are set as in the regular top-down flow.

```
set_attr operating_conditions string [find /libraries -library_domain domain]
set_attr wireload_selection string [find /libraries -library_domain domain]
```

For more information on setting design constraints, see Applying Constraints in *Using Encounter RTL Compiler*.

*Tip*

When you set the `force_wireload` attribute on a design or subdesign, make sure that the wireload model you set matches a wireload model defined for a library in the associated library domain.

## Apply Optimization Directives

For more information on optimization strategies and related commands, see Defining Optimization Settings in *Using Encounter RTL Compiler*.

## Associate Library Domains with Different Blocks of Design

To inform RTL Compiler about the special library use, associate the library domains with the design and blocks for which you want to use the dedicated libraries or dedicated library cells.

➤ To set the target library domain for the top design, specify the `library_domain` attribute on the design:

```
set_attribute library_domain library_domain design
```

➤ To set the target library domain for a subdesign, do the following

**a.** Uniquify the subdesign:

```
edit_netlist uniquify subdesign
```

See Example 4-2 for more information.

**b.** Specify the `library_domain` attribute on the subdesign:

```
set_attribute library_domain library_domain subdesign
```

**Note:** This attribute is hierarchical. It applies to all instances of the specified design or subdesign. So the order in which you specify the target domains is important. See Example 4-3 for more information.

When you *change* the library domain for a subdesign, RTL Compiler copies all attributes from the original mapped instances to the new mapped instances.

*Important*

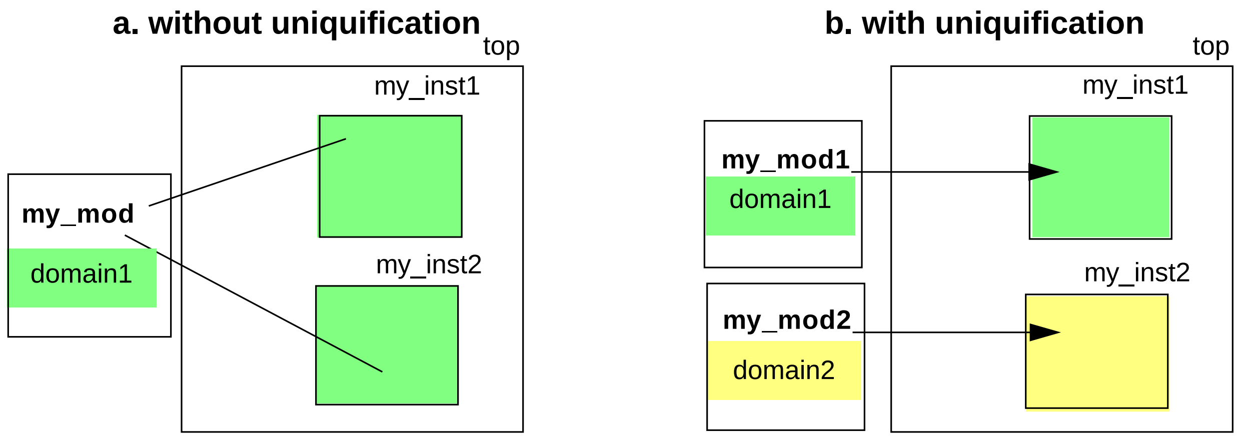
If you marked an instance *preserved*, the library domain that the instance is associated with will still be changed. However, the instance will still be marked preserved even though it will probably be pointing to another library cell in the new library domain. In other words, the `library_domain` attribute has a higher priority than the `preserve` attribute.

If RTL Compiler cannot find the corresponding cell in the libraries that belong to the new library domain, the instance becomes *unresolved*.

#### Example 4-2 Uniquifying a Subdesign before Associating the Library Domain

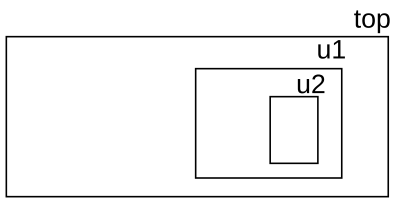
In the following design, module `top` has two instantiations of module `my_mod`. If you associate subdesign `my_mod` with library domain `domain1` before uniquifying subdesign `my_mod`, both instances `my_inst1` and `my_inst2` are associated with library domain `domain1.`This is illustrated in Figure 4-4. To associate instance `my_inst1` with library domain `domain1`, and instance `my_inst2` with library domain `domain2`, you first need to uniquify subdesign `my_mod,` and then associate both subdesigns with their domains individually.

#### Figure 4-4 Uniquifying a Subdesign before Associating the Library Domain



#### Example 4-3 Setting Target Library Domains

Assume a design `top` which has a subdesign `u1`. Subdesign `u1` has a subdesign `u2`.



Assume that all instances of u2 should be mapped using libraries from library domain `dom1`. All instances of u1, except for the instances of u2, should be mapped using libraries from library domain `dom2`. The instances in the remainder of the design should be mapped using libraries from library domain `dom1`.

To ensure the correct mapping, make the assignments in the following order:

```
set_attribute library_domain [find / -library_domain dom1] /designs/top
set_attribute library_domain [find / -library_domain dom2] [find / -subdesign u1]
set_attribute library_domain [find / -library_domain dom1] [find / -subdesign u2]
```

## Synthesize Design

After the constraints and optimizations are set for your design, you can proceed with synthesis by issuing the `synthesize` command.

➤ To synthesize your design using the `synthesize` command, type:

```
synthesize -to_mapped
```

**Note:** The design can be synthesized top down, without the need for manual partitioning.

The different portions of the design that are associated with different library domains will be mapped to the target libraries of those library domains and optimized.

For details on the `synthesize` command, see Synthesizing your Design in *Using Encounter RTL Compiler*.

## Analyze Design

After synthesizing the design, you can generate detailed timing and area reports using the various `report` commands.

For more information on generating reports for analysis, see Generating Reports in *Using Encounter RTL Compiler* and “Analysis Commands” in the *Command Reference for Encounter RTL Compiler*.

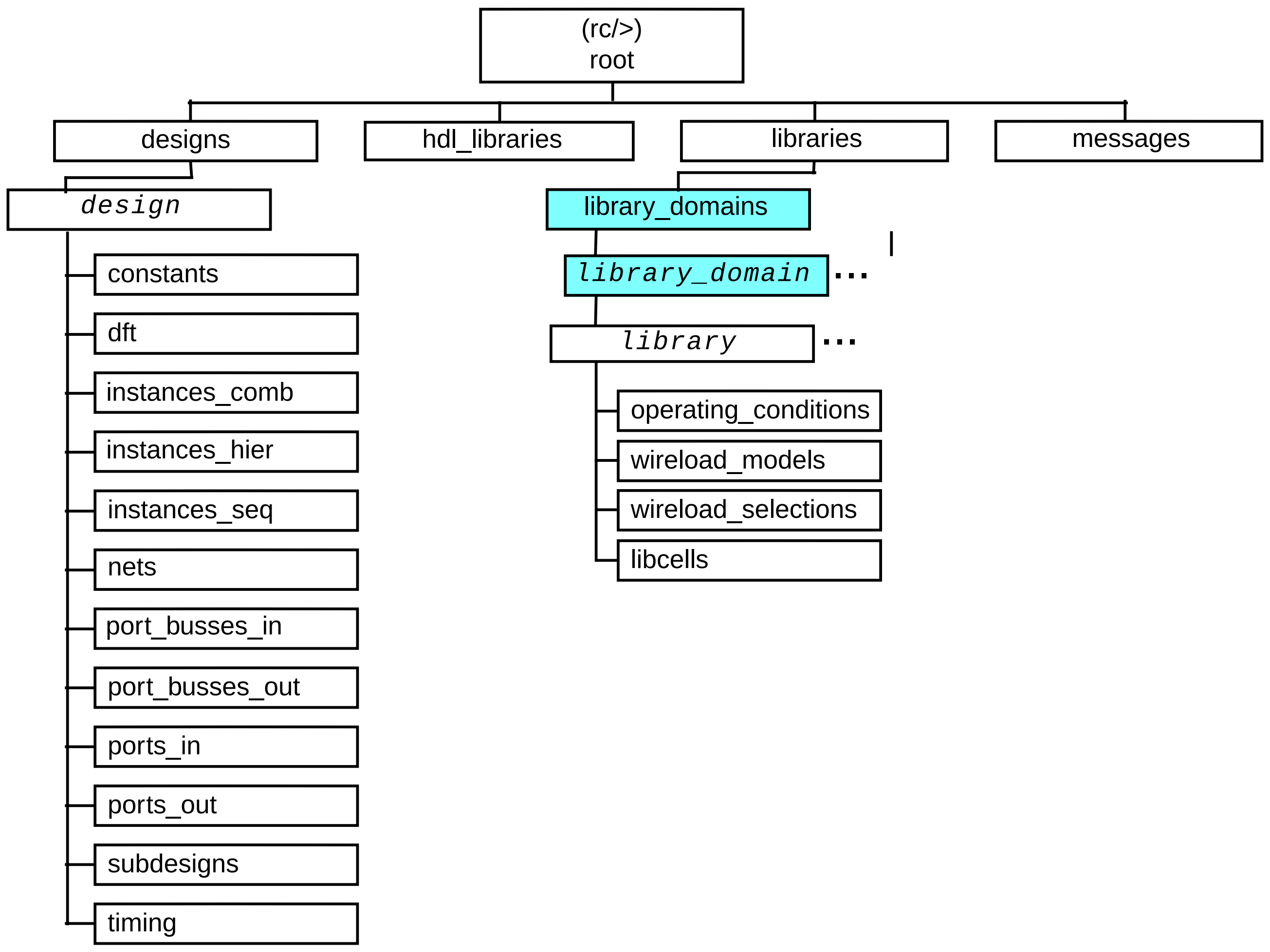
Most reports reflect information for the library domains. Gates Report

In the timing report, the domain information is added to the `Type` column.

## Library Domain Information in the Design Information Hierarchy

RTL Compiler stores the original design data along with additional information added by RTL Compiler in the design information hierarchy in the form of attributes. Figure 4-5 highlights where the library domain information is stored in the design information hierarchy.

**Figure 4-5 Design Information Hierarchy**



## Additional Tasks

### Removing a Library Domain

➤ To remove a library domain, use

```
rm [find /libraries -library_domain domain]
```

When you remove a library domain, RTL Compiler removes

- The libraries that are part of that library domain
- Any level-shifter group that was referring from or to this library domain

#### *Additional Notes*

- RTL Compiler links any instances in the subdesigns associated with a library domain that is being removed to the default library domain. While relinking, RTL Compiler copies all attributes from the original mapped instance to the new mapped instance.

  *Important*

  If you marked an instance *preserved*, the library domain that the instance is associated with will still be removed. However, the instance will still be marked preserved even though it will probably be pointing to another library cell in the default library domain. In other words, the `library_domain` attribute has a higher priority than the `preserve` attribute.

  If RTL Compiler cannot find the corresponding cell in the libraries that belong to the default library domain, the instance becomes *unresolved*.

- You can remove the library domain that is marked the default library domain, by first setting the `default` attribute on another library domain.

  In the following example, `dom_1` is the default library domain. You can only remove library domain `dom1`, after changing the `default` for example to library domain `dom_2`,

```
rc:/> get_attribute default [find /libraries -library_domain dom_1]
true
rc:/> set_attribute library typical.lib dom_2
Setting attribute of library_domain dom_2: 'library' = typical.lib
rc:/> set_attribute default true [find /libraries -library_domain dom_2]
Info    : Default library domain has been set. [LBR-109]
        : Default domain changed from: /libraries/library_domains/dom_1 to /
libraries/library_domains/dom_2

  Setting attribute of library_domain dom_2: 'default' = true
rc:/> rm [find /libraries -library_domain dom_1]
```

- You can only remove the default library domain if it is the only library domain that remains. If you remove the default library domain and a design was loaded, all instances become unresolved. In that case, none of the instances have timing, power or area information.

## Saving Information

➤ To save the information for a later synthesis session, use the following commands:

```
write_hdl > design.v
write_script > design.scr
write_sdc > design.sdc
```

These commands will save specific information such as library domain-associations with portions of the design.

## What If Analysis

You can check the effect on timing by using different library domains for some portions of the designs. The what-if analysis can be done before or after mapping, although you can get more meaningful results when you perform it after mapping.

*Important*

To do a what-if analysis, RTL Compiler expects that the libraries in the library domain you want to switch to, have the same set of cells with the same names as the libraries in the original library domain.

When you change the library domain assignment of a subdesign, RTL Compiler tries to rebind each cell in the subdesign by searching for a cell with the same name in a library in the new library domain. If RTL Compiler finds such cell, it uses the timing and power information of this cell from the new library domain to perform timing and power analysis and optimization. Otherwise, the original instance becomes an unresolved instance with the library cell name as its subdesign name.

### *What-If Analysis before Mapping Flow*

1. Setup.
2. Create library domains.
3. Load libraries
4. Read netlist.
5. Elaborate design.
6. Set constraints.
7. Set optimization directives.
8. Assign library domains to portions of design.
9. Report on timing (and power).

**What-if Analysis Steps**

10. Reassign the target library domain for a portion of the design.
11. Report on timing (and power).
12. Repeat steps 10 through 11 until you are satisfied with results.
13. Map design.
14. Continue flow.

### *What-If Analysis after Mapping Flow*

1. Setup.
2. Create library domains.
3. Load libraries
4. Read netlist.
5. Elaborate design.
6. Set constraints.
7. Set optimization directives.
8. Assign library domains to portions of design.
9. Map design.
10. Report on timing (and power).

**What-if Analysis Steps**

11. Save information (see Saving Information).
12. Reassign the target library domain for a portion of the design.
13. Either remap or do incremental synthesis.
14. Report on timing (and power).
15. Repeat steps 11 through 14 until you are satisfied with results.
16. Restore information depending on the what-if results.
17. Continue flow.

# 5
# Quality Analyzer Flow

## Overview

The RTL Compiler Quality Analyzer (RCQA) software performs signoff checks on the RTL at the RTL development stage before front end implementation, and on the structural netlist before back-end implementation.

For RTL signoff, you can use the RCQA software to confirm that the RTL and constraints are ready for synthesis, simulation, equivalency checking and test. With the RCQA software, you can efficiently complete as many design logical signoff checks as possible early in the design cycle and avoid downstream design flow iterations due to late checking, and avoid rerunning many of the checks past the RTL signoff point.

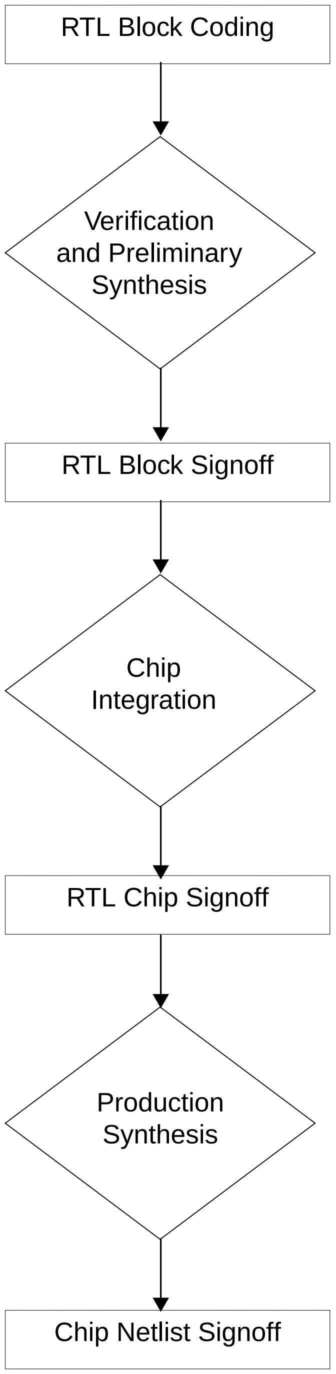
For Front-End signoff, you can use the RCQA software to ensure efficient handoff from front-end signoff to back-end implementation to confirm netlist and constraints are truly ready for back-end implementation, and avoid iterations from the back-end to the front-end due to late checking.

You can run the following signoff checks:

- Clock Domain Crossing Checks.
- Constraint (SDC) Checks
- Design For Test (DFT) Checks
- HDL Lint Checks
- Library Checks
- Low Power Checks
- Physical Checks

## RCQA Flow Stage Methodology

The following shows which recommended checks to use during which flow stage:



Checks: HDL, CDC, Library, DFT

Checks: HDL, CDC, Constraint, DFT, Low Power

Checks: HDL, CDC, Constraint, DFT, Low Power

Checks: Netlist, Constraint, DFT, Low Power, Physical

## HDL

These are done at the RTL block level and chip integration level. For RTL, this checks the RTL for any coding style, synthesis compatibility, simulation race conditions, and naming conventions. For netlists, this performs structural lint checks for the netlist.

HDL lint checks are recommended during the following flow stages:

- RTL Block Code
- RTL Block Signoff
- RTL Chip Signoff

*Important*

If the HDL lint checks do not pass for synthesizability checks, they must be fixed before running clock domain crossing, constraint, DFT, and low power checks.

**Note:** HDL lint checks might require a `.lib` file for missing modules.

## CDC

These are done for designs with multiple clocks that need clock domain crossing (CDC) checks. These checks are started with the RTL rather than netlist as problems are easier to find at the RTL stage. These checks flag errors in designs where clock domain crossings do not have appropriate clock synchronizer cells.

CDC checks are recommended during the following flow stages:

- RTL Block Code
- RTL Block Signoff
- RTL Chip Signoff
- Chip Netlist Signoff

## Netlist

Netlist checks detect errors in the netlists that can cause problems during the netlist implementation through the backend. Some of these errors are checking for multiple drivers on a net without use of tristate logic, unused output and inputs, correctness of syntax, and missing modules in the netlists.

Netlist checks are recommended during the Chip Netlist Signoff flow stage.

## Library

Technology libraries used for synthesis must be error free. Errors flag non-conformity to the liberty specification, such as unsupported timing arcs for synthesis, inconsistent operating conditions, and unsupported constructs.

Library checks are recommended during the following flow stages:

- Library Development
- ASIC Vendor Selection
- RTL Block Code

**Note:** Library checks should be completed before running synthesis or a full set of DFT checks on gate-level netlists

## Constraint

For RTL and netlists, these checks perform syntax checks on the SDC and basic structural checks on the design, and reports missing input/output delays, invalid from/to points of exceptions, and so on.

Constraint (SDC) checks are recommended during the following flow stages:

- RTL Block Signoff
- RTL Chip Signoff
- Chip Netlist Signoff

## DFT

Cadence recommends that DFT checks are done at every step of the design flow, and whenever there is a meaningful change to the design. These checks make sure that the resulting design is scannable, clock rules are not violated, and if there are conflicting off state requirements.

DFT checks are recommended during the following flow stages:

- RTL Block Code

- RTL Block Signoff
- RTL Chip Signoff
- Chip Netlist Signoff

## Low Power

These check designs that use low power techniques: multiple supply voltages (MSV), power shut off (PSO), and state retention power gating sequential cells. For RTL and netlists, these check for completeness and correctness of the CPF file provided for the RTL and performs low power implementation checks for cell types such as level shifters, isolation, and state retention.

CPF quality checks are recommended during the following flow stages:

- RTL Block Signoff
- RTL Chip Signoff
- Chip Netlist Signoff

**Note:** The results can be different for the RTL and the netlist for the same CPF file, for example, if there are test ports in the netlist that are not in the RTL.

## Physical

For physical libraries, the scaling factors used in back-end tools must be consistent with the technology libraries (`.lib`) used to create the netlists in the front-end tools. Physical design data checks perform LEF library and captable checks, library and LEF inconsistency checks, and DEF (floorplan, placement) checks.

Physical checks are recommended during the Chip Netlist Signoff flow stage.

## RCQA GUI-Based Methodology

The following shows a typical RCQA flow when using the graphical user interface (GUI).

1. Start the RCQA software.

   ```
   %> rc -qa
   ```

   This will start the software in GUI mode by default.

2. Create and load the configuration file.

   Use the Create Configuration File form (*File – Configuration File – Create*) to create in the configuration file, and use the Read RCQA Configuration File form (*File – Configuration File – Read*) to read it in.

3. Select and run rule checks for your level of signoff.

   Use the Signoff Checks form (*Tools – Signoff Checks*) to select and run the signoff checks.

4. Verify that the checks ran successfully in the shell window.

5. Generate a report of violations in the shell window

   Use the Report Checks form (*Tools – Report Checks*) to choose a report category, report type, and report severities.

6. Open the Signoff Checks Summary window to analyze the scope of violations

   Use the Signoff Checks Summary form (*Tools – Report Summary*) to view a pie chart graphically represents the distribution of errors, warnings, and info rule severity messages.

7. Open the Message Browser and select Filter.

   Use the Message Browser Filter form (*Tools – Message Browser*, click *Filter*) to view and hide individual occurrences of message IDs while navigating through the messages.

8. In the Message Browser Filter window, select a *Severity* of *Error*.

9. Go Back to the Message Browser window.

10. Select one rule violation instance.

11. Click on the link under *File information* to debug the rule violation.

12. Use additional diagnosis reports, if needed.

    Use the forms from the *Diagnosis* submenu (*Tools – Diagnosis).*

**13.** Launch Individual analysis engines, if needed.

Use the forms from the *Launch* submenu (*Tools – Launch).*

**14.** Open the Message Browser's File Editor and fix any issues.

Right-click on a highlighted file in the *Message* panel of the Message Browser and select *Edit File*.

**15.** Go Back to Message Browser and add rule instances and rule groups to the filter.

**16.** Go to Message Browser Filter Window apply the filter.

**17.** In the Message Browser Filter window, and select a *Severity* of *Warning* and repeat from step 7 on.

**18.** Choose the next rule violation instance and repeat from step 10 on.

**19.** In the Message Browser Filter window, and select a *Severity* of *Warning* and repeat from step 8 on.

**20.** Reload the configuration file and rerun rule checks to verify the fixes.

A

# Summary of the Flows

## Generic Flow

`set_attribute library` *library_name*
`read_hdl filename`
`elaborate`
*read constraints*
`synthesize -to_mapped`
`write_hdl`

## Physical Flow with predict_qos, LEF, and Capacitance Table

```
set_attribute library library_name
set_attribute lef_library {tech.lef cell.lef}
set_attribute cap_table_file cap_file
read_hdl filename
elaborate
read sdc
read_def def_file
synthesize -to_mapped
predict_qos
write_encounter design
```

## Physical Flow with predict_qos and Encounter Configuration File

```
read_encounter config config_file
read_def def_file
synthesize -to_mapped
predict_qos
write_encounter design
```

## Physical Flow without predict_qos

```
set_attribute library library_name
set_attribute lef_library {tech.lef cell.lef}
set_attribute cap_table_file cap_file
read_hdl filename
elaborate
read sdc
synthesize -to_mapped
write_hdl
```

## DFM Flow

```
set_attr library *library_name*
read_dfm *coefficients_file*
read_hdl *filename*
elaborate
*read sdc*
set_attribute optimize_yield true /
synthesize -to_mapped
report yield
report gates -yield
get_attribute yield
```

## Top-Down Synthesis Flow Using Multiple Library Domains

```
# general setup
#-------------
set_attributer lib_search_path ...
set_attribute hdl_search_path ..
```

```
# create library domains
#----------------------
create_library_domain domain_list
```

```
# specifiy the target libraries for each library domain
#-----------------------------------------------------
set_attribute library library_list1 [find /libraries -library_domain domain1] /
set_attribute library library_list2 [find /libraries -library_domain domain2] /
...
```

```
# load and elaborate the design
#-----------------------------
read_hdl design.v
elaborate
```

```
# specify timing and design constraints
#--------------------------------------
# specify the following constraints per library domain
#-----------------------------------------------------
set_attr operating_conditions string [find /libraries -library_domain domain]
set_attr wireload_selection string [find /libraries -library_domain domain]
```

```
# set target library domain for top design
#----------------------------------------
set_attribute library_domain library_domain design
```

```
# set target library domain for blocks
#------------------------------------
edit_netlist uniquify subdesign
set_attribute library_domain library_domain subdesign
```

```
#synthesize the design
#--------------------
synthesize -to_mapped
```

```
# analyze design
------------------
report timing
report gates
```

```
# export design
#-------------
write_encounter design design] [-basename string] [-gzip_files]
[-reference_config_file file] [-preserve_avoid_cells]
[-ignore_scan_chains] [-floorplan {.def|.pdef|.fp}] [-lef file_list]
```

## RCQA Flow

```
# read in the configuration file
read_config_file <file>
# run all signoff checks. If specifying a CPF file, run ‘signoff_checks power’
# before ‘signoff_checks dft’
signoff_checks hdl_lint
signoff_checks clock_domain_crossing
signoff_checks constraints
signoff_checks dft
signoff_checks library
signoff_checks power
signoff_checks physical
# report on all signoff checks
report checks
# go to GUI for analysis
```

For configuration file examples that specify the full path and search path for the library and RTL files, see Sample Configuration Files in *Using Encounter RTL Compiler Quality Analyzer*.