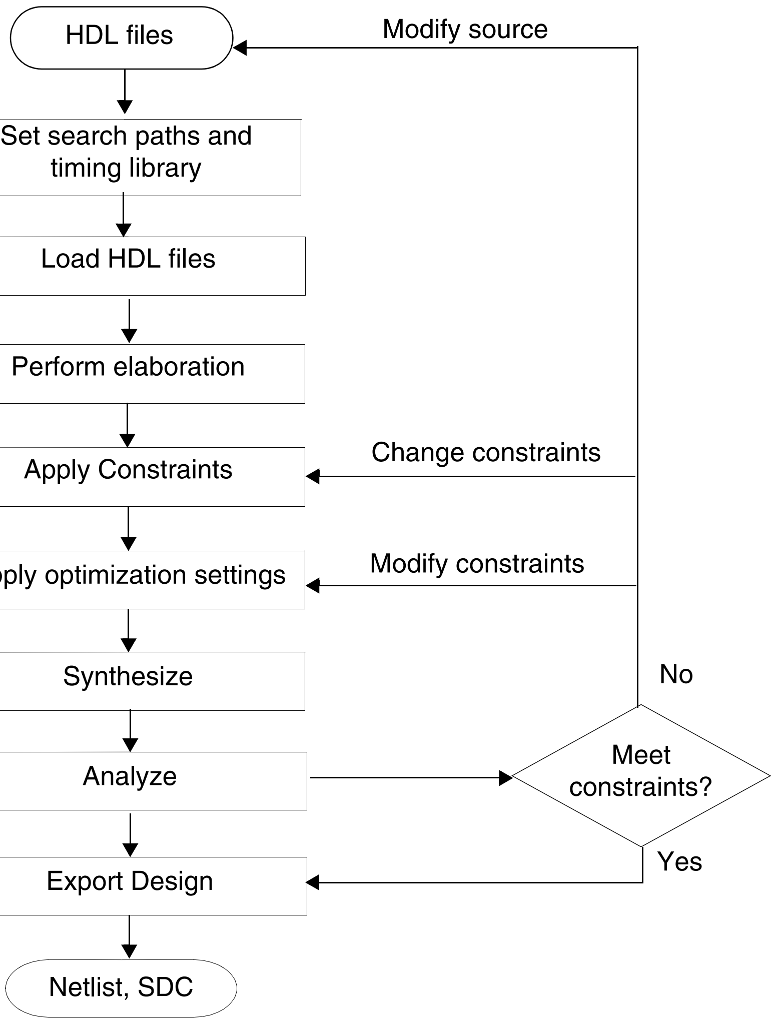
1

Getting Started with the Generic Flow

Overview

Figure 1-1 on page 18 shows the generic Genus 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 Genus Work Flow



Tasks

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 Genus is invoked.

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

```
genus@root:> set_db init_lib_search_path path
genus@root:> set_db script_search_path path
genus@root:> set_db init_hdl_search_path path
```

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

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:

```
genus@root:> set_db library lib_name.lib
```

Genus will use the library named `lib_name.lib` for synthesis. 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:

```
genus@root:> set_db library /usr/local/files/lib_name.lib
```

➤ To specify a single library compressed with gzip:

```
genus@root:> set_db library lib_name.lib.gz
```

➤ To append libraries:

```
genus@root:> set_db 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

Genus 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 Genus. When you issue a `read_hdl` command, Genus 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 }
  ```

- ❑ You can also read the design from simulation optionfiles:

  ```
  read_hdl -f optionfile
  ```

For more information on loading the design from optionfiles, see Reading Designs in Simulation Environment on page 59.

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 Genus will issue an error when it encounters these characters.

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

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 Genus shell.
- Include a constraints file.
- Read in SDC 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 Genus, 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.

Performing Synthesis

After the constraints and optimizations are set for your design, you can proceed with synthesis.

➤ To synthesize your design, type the following commands:

```
genus@root:> syn_generic
genus@root:> syn_map
```

For details on the synthesis commands, see *Command Reference*.

After these two steps, 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 generate 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 "Analysis and Report Commands" in the *Genus Command Reference*.

Exporting the Design

The last step in the flow involves exporting the design post synthesis to write out the gate-level netlist, design and SDC constraints.

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.

```
genus@root:> write_hdl > design.v
```

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

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

```
genus@root:> 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:

```
genus@root:> write_sdc > constraints.sdc
```

This command writes the SDC 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.

➤ To export all necessary files for the Innovus System, use the following command:

```
genus@root:> write_design -innovus design_name
```

Exiting Genus

There are three ways to exit Genus 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.

Recommended Flow

```
#general setup
#--------------
set_db init_lib_search_path path
set_db init_hdl_search_path path

#load the library
#-------------------------------
set_db library library_name

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

#specify timing and design constraints
#---------------------------------------
read_sdc sdc_file

#add optimization constraints
#-----------------------------
.....
#synthesize the design
#---------------------
syn_generic

syn_map

#analyze design
------------------
report_area
report_timing
report_gates

#export design
#-------------
write_hdl > dessign.vm
write_sdc > constraints.sdc
write_script > constraints.g

# export design for Innovus
#------------------------
write_design [-basename string] [-gzip_files] [-tcf]
[-innovus] [-hierarchical] [design]
```