

2

Getting a Quick Start

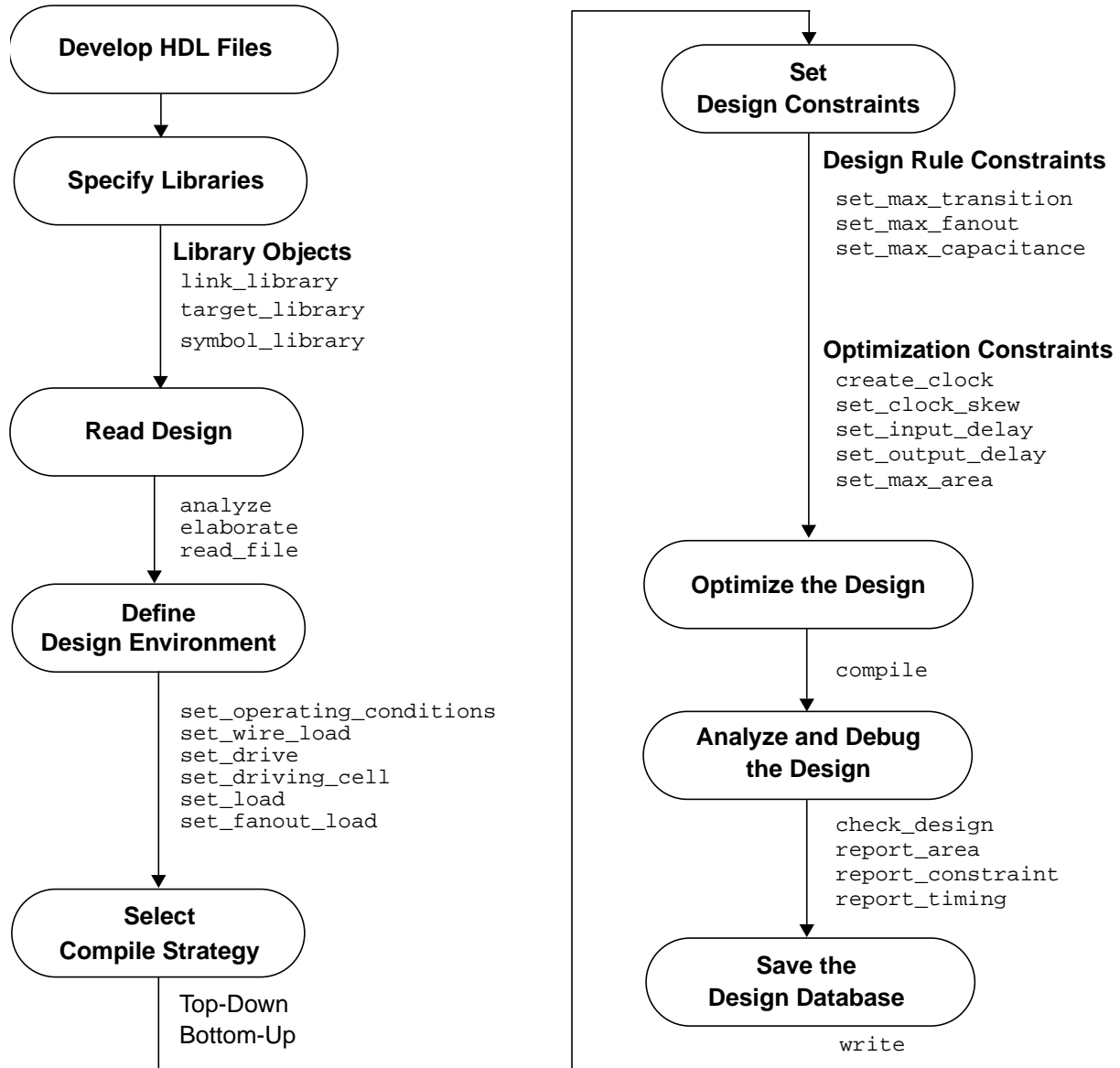
This chapter walks you through the basic synthesis design flow (shown in Figure 2-1). You use the same basic flow for both design exploration and design implementation.

The following sections each describe one step of the flow:

- Developing the HDL Files
- Starting the Shell Interface
- Specifying the Libraries
- Reading Designs
- Defining the Design Environment
- Selecting Your Compile Strategy
- Setting the Design Constraints

- [Optimizing Your Design](#)
- [Analyzing and Debugging Your Design](#)
- [Saving the Design Database](#)
- [Exiting the Shell Interface](#)
- [Design Compiler Session Example](#)

Figure 2-1 Basic Synthesis Design Flow



Developing the HDL Files

To achieve the best synthesis results, pay close attention to the following issues when developing the HDL files:

- Managing the design data
- Partitioning the design
- Coding the design

Managing the Design Data

To simplify data exchanges and data searches, develop data organization and revision control methods that all designers adhere to, such as

- Rules for file creation, maintenance, and deletion
- A file naming convention
- A hierarchical directory structure for design data

For more information, see Chapter 3, “Working With Design Files.”

Partitioning the Design

Partitioning a design effectively can enhance synthesis results. Efficient partitioning can also help obtain optimal results by reducing compile time and simplifying constraint definition.

Keep the following in mind when you partition your design:

- To achieve the best synthesis results,

- Keep related combinational logic in the same VHDL entity or Verilog module.
- Merge resources in the same VHDL process or Verilog always block.
- Keep user-defined resources with the logic they drive.
- Separate blocks having different goals.
- Separate structural logic from random logic.
- To reduce compile time,
 - Eliminate glue logic at the top level.
 - Maintain a reasonable gate count for each block.
 - Isolate timing exceptions in the same block.
- To simplify constraint definition,
 - Register all outputs.
 - Partition the top level and isolate I/O pads.

Achieving the best synthesis results is the most important goal. It is easier to obtain good synthesis results if you arrange the logic correctly, both in modules and in the hierarchy.

The best approach to partitioning is to plan the partitioning before you write the HDL code. See “Partitioning for Synthesis” in Chapter 3 for information about partitioning strategies.

If the HDL code is already developed, you can use Design Compiler commands to rearrange the hierarchy without modifying the HDL code. See “Changing the Design Hierarchy” in Chapter 6 for more information.

Coding the Design

For easier maintenance, use meaningful and efficient names, a single naming style, and follow conventions for naming suffixes. In addition, a good HDL coding style can generate smaller and faster designs.

For coding style recommendations, see “HDL Coding for Synthesis” in Chapter 3.

Starting the Shell Interface

To invoke the `dc_shell` interface, enter the `dc_shell` command at the system prompt:

```
% dc_shell
```

To get information about the commands available in `dc_shell`, use the `list -commands` command (`dcsh`) or `help` command (Tcl). To get detailed information about commands, variables, or error messages, use the `man` command.

Specifying the Libraries

Design Compiler uses the following libraries:

Technology libraries

These libraries specify the semiconductor vendor’s set of cells and related information, such as cell names, cell pin names, delay times, and pin loading.

Specify the technology libraries by using the `link_library` and `target_library` variables.

Symbol libraries

These libraries define the symbols for schematic viewing in Design Analyzer.

Specify the symbol libraries by using the `symbol_library` variable.

DesignWare libraries

These libraries provide the implementations for many built-in HDL operators. You do not need to specify the standard DesignWare library.

See Chapter 5, “Working With Libraries,” for details about these libraries and how to specify them.

Reading Designs

Design Compiler can read RTL designs as well as gate-level netlists. Design Compiler commands, attributes, and constraints are directed toward a design and its objects.

Design Compiler supports Verilog and VHDL RTL designs. Use the `analyze` and `elaborate` commands to read RTL designs.

Design Compiler supports many formats for gate-level netlists. Use the `read_file` command to read gate-level netlists.

For more information, see “Reading Designs” in Chapter 6, “Working With Designs in Memory.”

Defining the Design Environment

The design environment is a set of attributes and constraints that model the environment surrounding the design being synthesized. Design Compiler uses these attributes and constraints to compute the effect of the design environment on the circuit performance.

The design environment includes the following items:

Operating conditions

The operating conditions include the operating temperature, supply voltage, and manufacturing process.

Wire load models

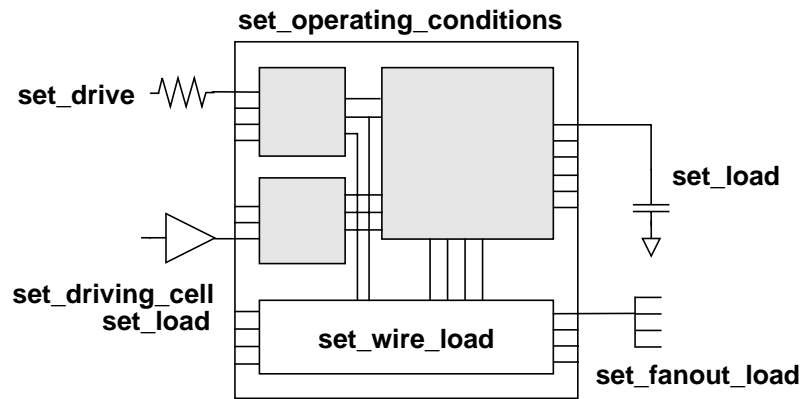
Wire load modeling estimates the effect of wire length and fanout on resistance, capacitance, and area to estimate wire delays.

System interface

The system interface includes the cells driving the design and the loads driven by the design.

Figure 2-2 shows the commands used to set these attributes and constraints. See “Defining the Design Environment” in Chapter 7 for more information about these commands.

Figure 2-2 Commands Used to Define the Design Environment



Selecting Your Compile Strategy

The basic strategies you can use to optimize a hierarchical design are

- Top-down compile
- Bottom-up compile
- Characterize, compile

You can use different strategies for different designs, hierarchy levels, and components in the design.

Each method has advantages and disadvantages that depend on your design goals. This manual describes the top-down and bottom-up compile strategies. The characterize, compile strategy is an advanced technique and is not discussed in this manual.

See “Selecting a Compile Strategy” in Chapter 7 for details about selecting a compile strategy.

Setting the Design Constraints

Design goals are communicated to Design Compiler as constraints. Design Compiler uses two types of constraints when optimizing your design:

Design rule constraints

These are implicit constraints, defined by the vendor-provided technology library. Design rule constraints set the requirements for correct design function, and they apply to any design using that library. You can make these constraints more restrictive than the library definitions.

Table 2-1 shows the commands used to define the most commonly used design rule constraints. For more information, see “Setting Design Rule Constraints” in Chapter 7.

Table 2-1 Commonly Used Design Rule Constraints

Constraint	Commands
Transition time	set_max_transition
Fanout load	set_max_fanout
Capacitance	set_max_capacitance

Optimization constraints

These are explicit constraints, which you define. Optimization constraints are measurable circuit characteristics, such as timing and area, that provide the optimization goals. They apply only to the design on which you are working. Specify realistic constraints to prevent excessive compile runtimes.

Table 2-2 shows the commands used to define the most commonly used optimization constraints. For more information, see “Setting Optimization Constraints” in Chapter 7.

Table 2-2 Commonly Used Optimization Constraints

Constraint	Commands
Clock definition	<code>create_clock</code> <code>set_clock_skew</code>
Port timing requirements	<code>set_input_delay</code> <code>set_output_delay</code>
Combinational path requirements	<code>set_min_delay</code> <code>set_max_delay</code>
Area requirement	<code>set_max_area</code>

Optimizing Your Design

Design Compiler uses the target technology libraries, design rule constraints, and optimization constraints to synthesize your design description into a technology-specific gate-level implementation. Use the `compile` command to invoke the optimization process.

See Chapter 8, “Optimizing Your Design,” for optimization techniques and details about the `compile` command.

Analyzing and Debugging Your Design

Design Compiler can generate reports which can help you analyze optimization results and debug problems. See Chapter 9, “Analyzing and Debugging Your Design,” for analysis and debugging techniques.

Saving the Design Database

Use the `write` command to save the design. Design Compiler supports many formats for saving designs. For more information, see “Saving Designs” in Chapter 6.

Exiting the Shell Interface

You can exit `dc_shell` at any time and return to the operating system. However, `dc_shell` does not automatically save designs when you exit.

Use the `write` command to save your design before exiting `dc_shell`.

```
dc_shell> write -hierarchy -output my_design.db
```

To exit `dc_shell`, do one of the following:

- Enter `quit`.
- Enter `exit`.
- Press `Ctrl-d`.

Design Compiler Session Example

Example 2-1 shows a script that performs a top-down compile run, using the basic synthesis flow. The script contains comments that identify each of the steps in the flow.

Example 2-1 Top-Down Compile Script

```
/* specify the libraries */
target_library = my_lib.db
symbol_library = my_lib.sdb
link_library = "*" + target_library

/* read the design */
read -format verilog Adder16.v

/* define the design environment */
set_operating_conditions WCCOM
set_wire_load "10x10"
set_load 2.2 sout
set_load 1.5 cout
set_driving_cell -cell FD1 all_inputs()
set_drive 0 clk

/* set the optimization constraints */
create_clock clk -period 10
set_input_delay -max 1.35 -clock clk {ain, bin}
set_input_delay -max 3.5 -clock clk cin
set_output_delay -max 2.4 -clock clk cout
set_max_area 0

/* map and optimize the design */
uniquify
compile

/* analyze and debug the design */
report_constraint -all_violators
report_area

/* save the design database */
```

```
write -format db -hierarchy -output Adder16.db
```

You could execute these commands in any of the following ways:

- Enter `dc_shell` and type all the commands.
- Enter `dc_shell` and execute the script file, using the `include` command (dcsh mode) or the `source` command (Tcl mode).

For example, assuming the script is in a file called `run.scr`, you execute the script file by entering the command

```
dc_shell> include run.scr
```

- Run the script from the UNIX command line by using the `-f` option of the `dc_shell` command.

For example, assuming the script is in a file called `run.scr`, you execute the script file from UNIX by entering the command

```
% dc_shell -f run.scr
```