SAMSUNG

ASIC Design Kit
for Synopsys' Applications
USER'S Guide

Revision 4.0

ELECTRONICS

Table of Contents

Chapter 1 Overview

Chapter 2 Installation

Chapter 3 Synthesis

Table of Contents (Continued)

Chapter 3 Synthesis (Continued)

Chapter 4 Appendix

List of Figures

1 OVERVIEW

This guideline describes the processes to follow when using Synopsys and Samsung Electronics Co., Ltd.(SEC) Design Kits to design an ASIC.

INTRODUCTION

For designers to use Synopsys' Design Compiler with SEC Synopsys synthesis technology libraries to synthesize modules and integrate them into ASICs, a "seamless" methodology is required. In addition, designers need a structured approach to integrate Design Compiler and SEC Design Kits to fully achieve functional and performance goals of an ASIC and in a fast turnaround time.

This document presents an overall flow for using SEC Design Kits and Synopsys tools, and describes in greater detail synthesis, optimization, and backannotation issues.

OVERALL FLOW

As ASICs become denser and more complex, the use of high-level design methodologies allows you to take advantage of the benefits of high-level design, resulting in high quality designs in less time. Systems should be defined and verified behaviorally first and then transported to an implementation phase. This implementation phase starts from the RTL code written so that a quality design can be easily synthesized. Note that the style of coding can limit the portability or reusability of a high-level description.

Before using high-level synthesis tools, you should decide on path timing, die area, clocking scheme, logic partitioning, and the testability structure. Planning at this design definition stage can have a significant impact on the gate-level netlist, simulation time, die cost, and ultimately the quality of the product.

The flow shown in Figure 1.1 illustrates a sequence in which SEC Design Kits and Synopsys tools are used. In some situations, a number of iterations may be needed, depending on the RTL code, the design, and timing constraints. Note that this overall flow aims mainly at obtaining testable synchronous designs, because the synthesis processes, produce synchronous structures. To produce asynchronous modules, you should accord such items with appropriate design attributes and timing constraints.

In the most straight forward case, a single pass would suffice for each tool, but trade-offs and interdependencies often affect the data obtained in a previous procedure. For example, after placement and routing, the wire lengths in the file are used for delay calculation. These delays are obtained from an SDF file created by the SEC In-House delay calculator (CubicDelay). This delay data is backannotated into Synopsys Design Compiler for reoptimization. Sometimes critical decisions require manual changes if the constraints are not satisfied and the probability of convergence is remote. In such cases, the RTL code should be re-written, or re-partitioned, or one should resort to early trial and error runs to get a rough estimate.

This flow carries the assumption that the high-level code is synthesizable and partitioned into modules. Currently, static timing analysis is used to verify whether timing constraints have been met at various iterative steps. Most of the logic and test synthesis steps are verified prior to fine-tuning the paths using in_place optimization, floorplanning, and layout. For sign-off purposes, the final structural verification is performed using a full-timing gate-level simulation engine such as VSS, Verilog-XL and so on.

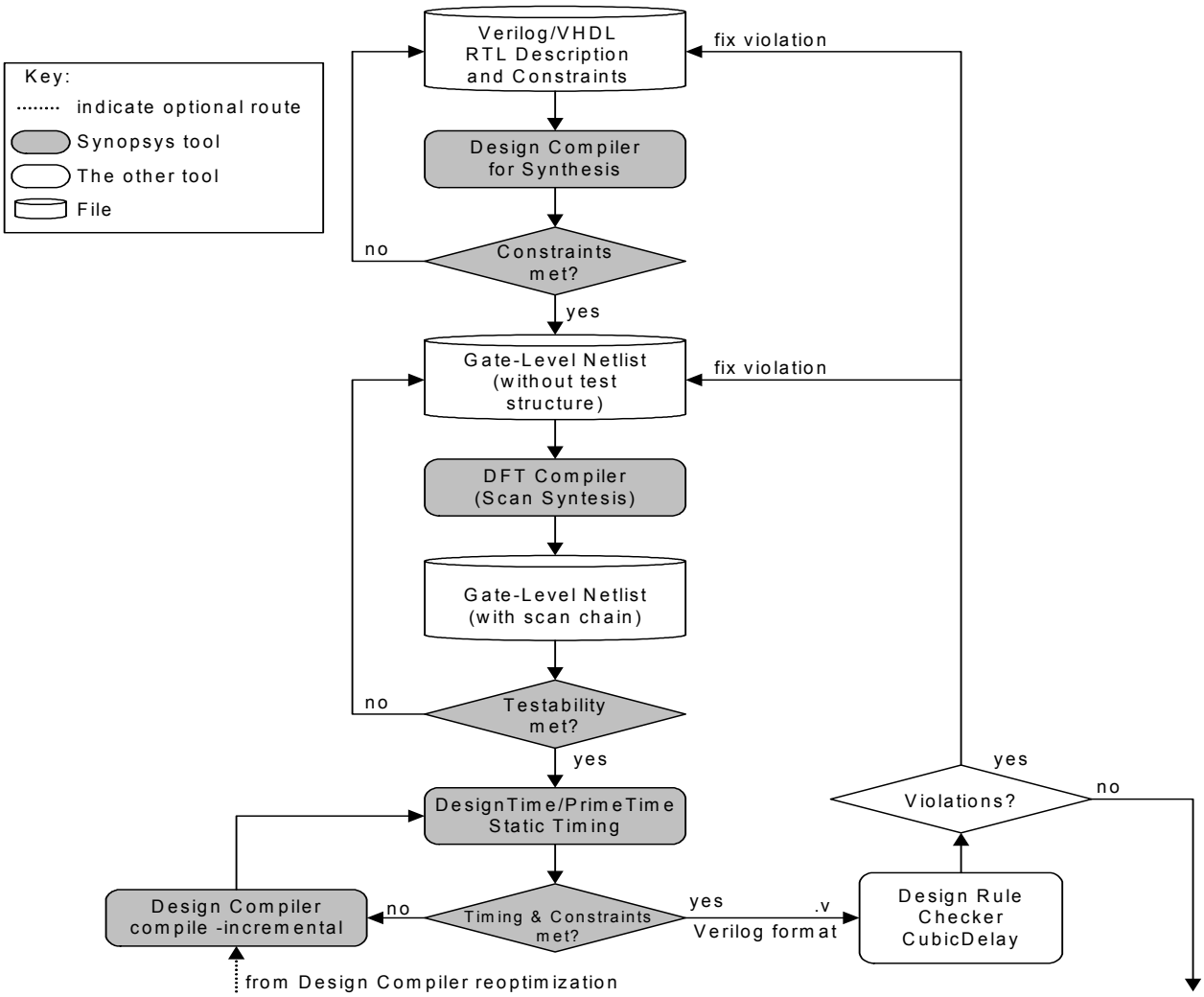

Figure 1-1. Overall SEC/Synopsys ASIC Design Flow

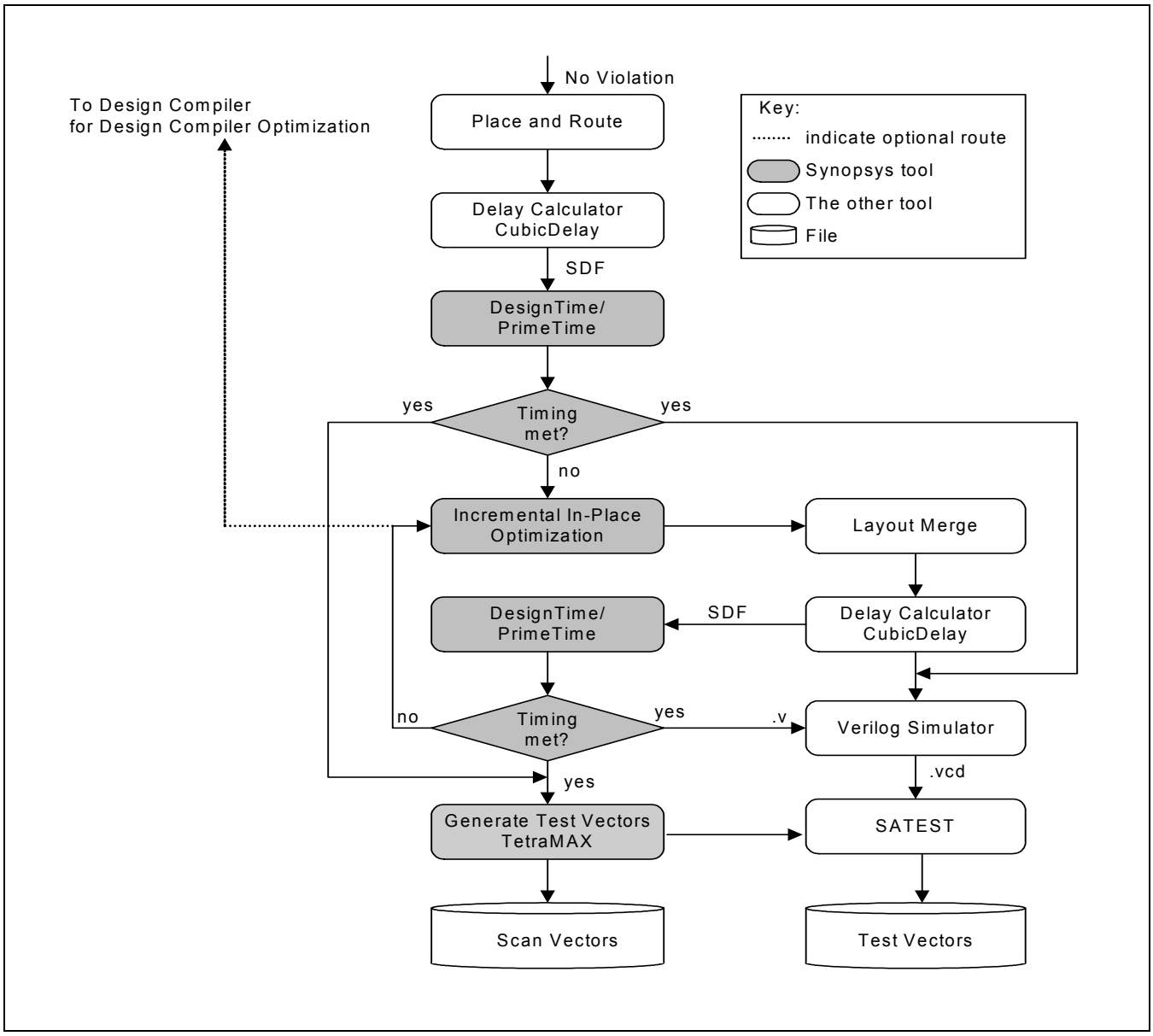

Figure 1-1. Overall SEC/Synopsys ASIC Design Flow (Cont.)

2 INSTALLATION

This chapter describes how to install SEC Synopsys Design Kit. The installation is illustrated with 150 normal Design Kit and the OS platform of SunOS 5.8. You can follow the same procedures for other SEC Synopsys Design Kit installation.

INSTALLING SEC SYNOPSYS DESIGN KIT

You can download the following directory from SEC IMS(IP Management System) :

— sec040901_0000_STD150E_regular_DK_Synopsys_N/

Run install program, sainstall.DK in download directory.

```
%sainstall.DK
```

If you execute ‘sainstall.DK’ with no argument, ‘sainstall.DK’ will automatically find the tar-gziped Design Kit file(s), and install it(them) interactively at proper path.

```
%sainstall.DK sec150e_synopsys.tar.gz
```

If you execute ‘sainstall.DK’ with argument(s), ‘sainstall.DK’ will install it(them) interactively at proper path.

```
*********************************************************
**  SAMSUNG ASIC Design Kit installation is completed! **
*********************************************************
```

After you have completed the Synopsys Design Kit installation, you should set the following variables:

```
%setenv SEC_SYNOPSYS <SEC Design Kit directory>/sec150e_synopsys
%setenv SEC_SYNOPSYS_AUX <SEC Design Kit directory>/sec150e_synopsys/aux
%set path = ($SEC_SYNOPSYS/bin/sparcOS5 $path)
```

For more information, please refer to

"*<SEC Design Kit directory>/*sec150e_synopsys/readme/README"

Now, you are ready to exercise the SEC Synopsys Design Kit. The next section describes the contents of the Design Kit.

SEC Synopsys Design Kit Structure

Figure 2.1 shows the SEC Synopsys design kit structure.

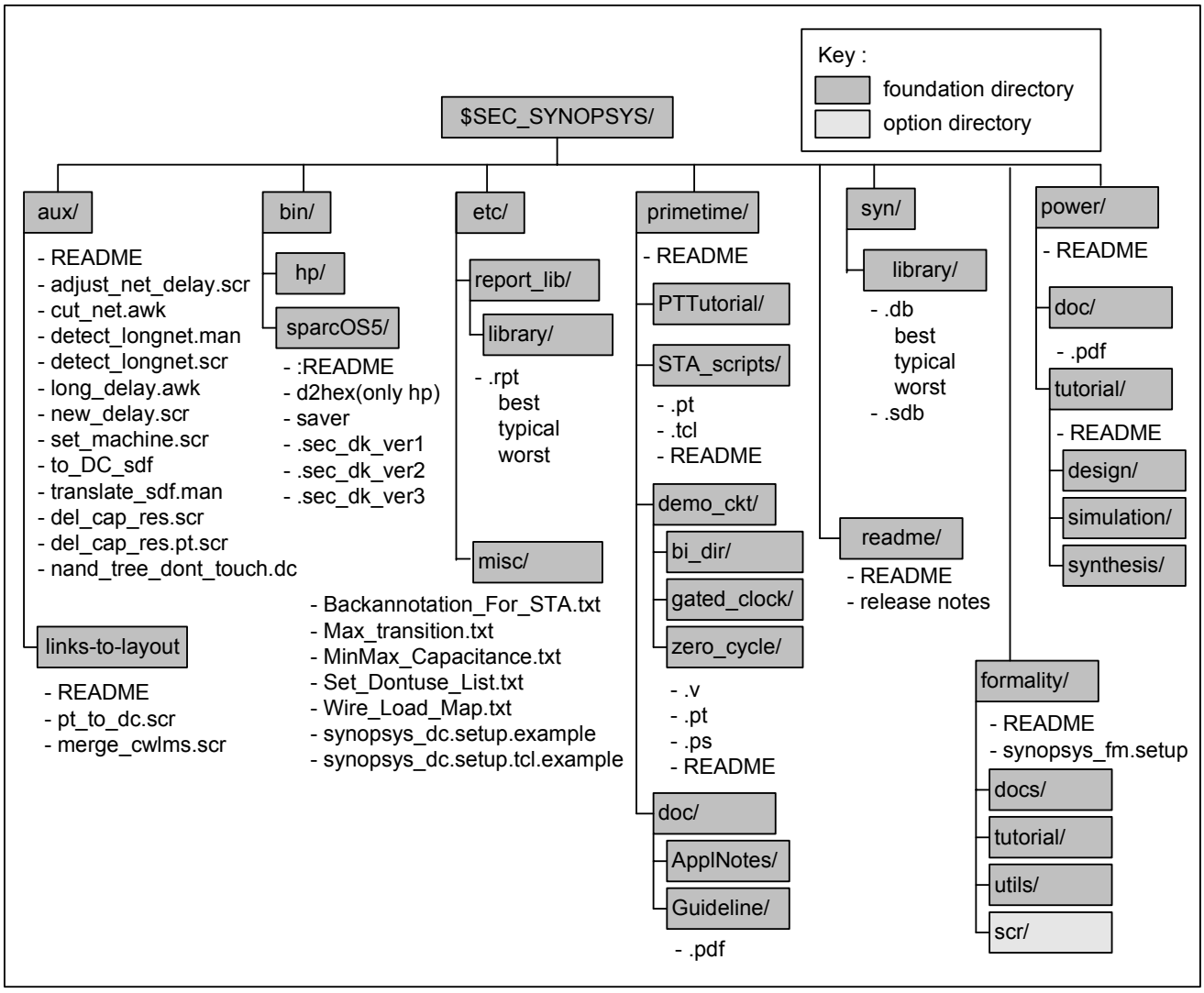

Figure 2-1. Synopsys Design Kit Structure

The structure in Figure 2.1 contains the following directories and files:

$SEC_SYNOPSYS:

— Synopsys Design Kit root directory (for instance, sec150e_synopsys)

$SEC_SYNOPSYS/aux

With the scripts in this directory, you can find nets whose delay is larger than 2 ns (detect_longnet.scr). If you want to modify delay values of these nets, you can annotates modified delays to the netlist (new_delay.scr).

See section Overruling Pre-layout Interconnection Delay for detailed procedures about using these scripts. Or refer to files README and detect_longnet.man files in this directory.

You can find the following files in this directory:

— README, adjust_net_delay.csh, ctc_net_r_zero.awk, ctc_net_r_zero.dc, cut_net.awk, detect_longnet.man, detect_longnet.scr, do_propagated_clock.dc, long_delay.awk, nand_tree_dont_touch.dc, new_delay.scr, set_ctc.man, set_machine.scr, to_DC_sdf, translate_sdf.man, del_cap_res.scr, del_cap.res.pt.scr,

$SEC_SYNOPSYS/aux/links-to-layout

This directory is available for using the LTL (Links-To-Layout) flow. you can find the following files in this directory:

— README, merge_cwlms.scr, pt_to_dc.scr

$SEC_SYNOPSYS/ etc

This directory includes report files generated by Synopsys Library Compiler, and files which include matters that require special attention such as the Synopsys setup file for Design Kit users.

$SEC_SYNOPSYS/etc/report_lib

It includes report files generated by the Synopsys Library Compiler's command, report_lib, for each library. The report files contain the cell list, cell attributes, operating conditions, and the library version.

$SEC_SYNOPSYS/etc/misc

This directory includes several files that contain useful information for synthesis and static timing analysis.

Backannotation_For_STA.txt describes the needed matters to interface CubicDelay when you run post-layout static timing analysis using Design Time. See 3.11 Back-annotation for STA for further information.

Max_transition.txt describes the maximum transition time of a technology library that is not described in the report file.

MinMax_Capacitance.txt describes min/max_capacitance of a technology library.

Set_Dontuse_List.txt describes the list of cells that have 'dont_use' attribute and describes how to remove the attribute.

Wire_Load_map.txt describes the area for each wire load.

synopsys_dc.setup.example(or *synopsys_dc.setup.tcl.example* for tcl mode) is a Synopsys environment file including naming rule, library setup, and variable definition to interface CubicDelay and third part tools. Users should use this file for the SEC design flow. See Appendix A for further information.

$SEC_SYNOPSYS/bin

This directory is divided into HP and SparcOS5 directories. Each directory includes utility programs saver.

A saver file is a simpler form of Samsung ASIC Version Viewer. It displays Design Kit's version.

You can find the following files in this directory:

— :README, d2hex(only hp), saver, .sec_dk_ver1, .sec_dk_ver2, .sec_dk_ver3.

$SEC_SYNOPSYS/primetime

This directory includes a PrimeTime setup file, documents, demo circuits, and an STA sign off tutorial for PrimeTime users.

$SEC_SYNOPSYS/primetime/doc

It includes PrimeTime application notes for SEC PrimeTime design kit users.

$SEC_SYNOPSYS/primetime/demo_ckt

Demo circuits and scripts are provided to demonstrate PrimeTime's analysis for design cases.
For each case, an application note is available in the directory $SEC_SYNOPSYS/primetime/doc/ApplNotes.

$SEC_SYNOPSYS/primetime/PTTutorial

For the PrimeTime STA Sign-off, a tutorial is available. This directory contains the tutorial. Read $SEC_SYNOPSYS/primetime/PTTutorial/README for more detailed information about the tutorial.

$SEC_SYNOPSYS/primetime/STA_scripts

This directory includes some useful sample scripts for at-speed, scan-test, and bist-sta.

$SEC_SYNOPSYS/syn

This directory includes the compiled db of the technology library.

Sub-directory includes the following multiple condition technology libraries. (for example, STD150E library)

— std150e_bst_135_n040.db : Best condition library [FF / 1.35V / -40°C]

— std150e_typ_120_p025.db : Typical condition library [NN / 1.20V / 25°C]

— std150e_wst_105_p125.db : Worst condition library [SS / 1.05V / 125°C]

Sub-directory includes also symbol library for Cadence's IC-Composer interface (_veri.sdb).

All libraries under directory syn were compiled using Library Compiler version 1999.10.

$SEC_SYNOPSYS/readme

This directory includes README file, Design Kit documents and release notes.

$SEC_SYNOPSYS/formality

This directory includes a Formality setup file, guidelines, utilities, and a tutorial for Formality users.

$SEC_SYNOPSYS/formality/docs

This directory includes guidelines.

$SEC_SYNOPSYS/formality/utils

This directory includes utilities for Formality users.

$SEC_SYNOPSYS/formality/tutorial

This directory includes demo circuits and scripts to demonstrate Formality's equivalence checking procedure for a design case.

$SEC_SYNOPSYS/formality/scr

This directory includes script file to treat the special IO cell as black-box for double equivalence checking. For more information, please refer to sec_fm_specialIO_guide.pdf in docs/ directory.

$SEC_SYNOPSYS/power

This directory includes the Power Compiler documents, and tutorials.

The compiled db of technology library in $SEC_SYNOPSYS/syn/*library_name* directory includes power characterization.

3 SYNTHESIS

This chapter describes synthesis, identifies synthesis inputs, and discusses the mechanics and impacts of synthesis. These methodologies use the capabilities of Synopsys' Design Compiler and the SEC Design Kits.

SYNTHESIS TOOLS

Synthesis is the process of transforming a circuit defined at one level of abstraction into one at a lower-level definition based on constraints. The transformation from a hardware description language (HDL), such as Verilog or VHDL, to a gate-level netlist is shown in Figure 3-1.

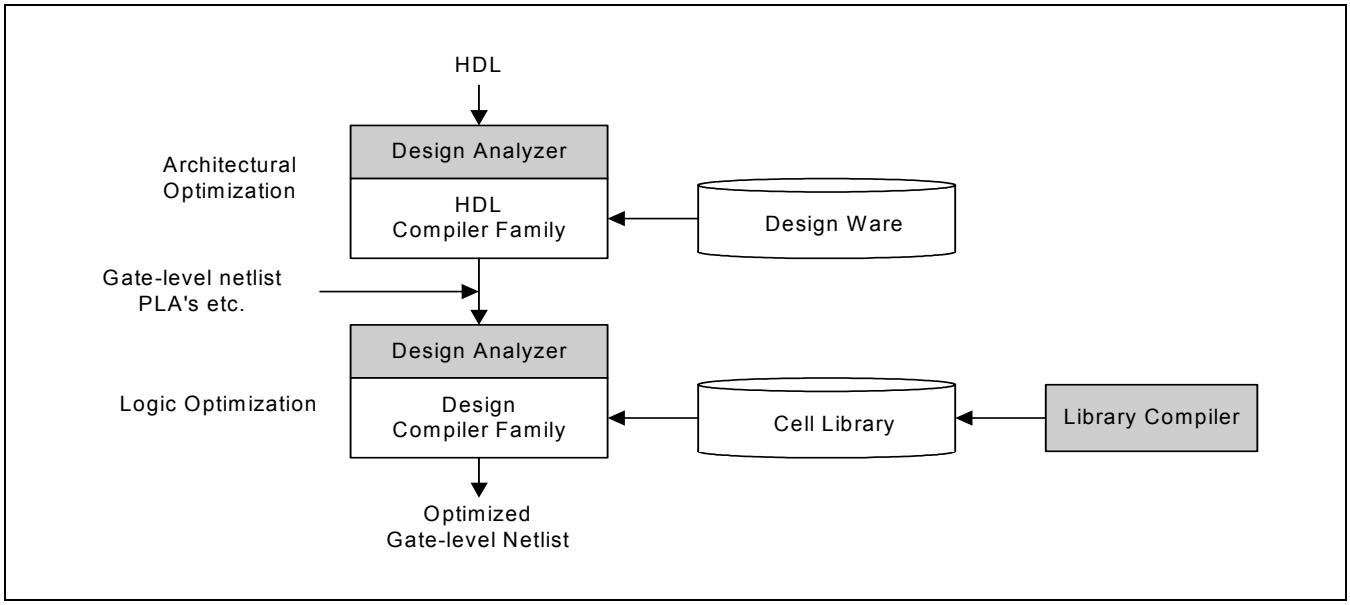

Figure 3-1. Logic Synthesis

Using Synopsys and SEC Design Kits, existing designs described in gate-level netlist can be reoptimized using gate-level optimization.

The tools and libraries represented in Figure3-1 are:

Design Analyzer™ - a graphical menu-based tool used to access the Synopsys tool set. It also analyzes the final netlist and schematic.

HDL Compiler Family - tools to optimize Verilog and VHDL designs at the architectural level. A design is optimized before the gate-level netlist is passed to Design Compiler. The architectural-level optimization steps include:

— Arithmetic optimization

— Timing and area-based resource sharing

— Common subexpression optimization

— Constraint-driven resource selection and parameterization

— Inference of synthetic parts (DesignWare)

Refer to VHDL Compiler Reference Manual and HDL Compiler for Verilog Reference Manual for more information on these steps.

- DesignWare - a library of operator-level components such as adders and multipliers. HDL Compiler selects the correct type of each component based on the HDL description and your area and timing goals.
- Design Compiler Family - tools to optimize designs at the gate level. You can define the designs in a variety of HDL formats and gate-level netlist formats. The optimization produces a gate-level netlist by using cells selected out of the target cell library.
- DesignTime (timing analyzer) - The Synopsys high-level synthesis, logic synthesis, test synthesis, and DesignTime tools all use the same embedded timing analyzer to compute the timing in a design. Commands can be issued to DesignTime in two ways. You can use either the menu interface (Design Analyzer) or the command-line interface (dc_shell) of Design Analyzer.
- PrimeTime - the Synopsys stand-alone full chip, gate-level static timing analyzer. Commands can be issued to PrimeTime in two ways. You can use commands in the command-line interface (pt_shell) or you can use the menu interface (primetime) of PrimeTime GUI.
- Cell Library - SEC's library of cells, such as AND, OR gate cells, used by Design Compiler.
- Library Compiler - tool to create cell libraries.

DESIGN PROCESS

It is assumed that you have a basic understanding of the requirements for synthesis, as well as some experience working with the Synopsys synthesis tools. It is also assumed that you have targeted the high-level code for a synchronous design, that the testability scheme is scan, and that the initial constraints are known.

Decisions about testability of large ASICs should be made at the beginning of the design process, because they affect the area and performance of the design. Therefore logic synthesis should also encompass the issues of embedding test techniques concurrently with refining design parameters.

Partitioning

Design management of large ASICs requires that you ensure the code is synthesizable and enough CPU and memory resources are available.

High-level partitioning plays an important role in the synthesis quality of a design. When partitioning a design, you should consider clock domains, leaf blocks, and hierarchy.

Clock Domains

You should partition the sub-chips into clock domains at as high a level as possible and apply detailed constraints at this level. The clock domain is sensitive to one edge of one clock.

Leaf Blocks

Leaf blocks are blocks that can be synthesized and optimized as a unit without any special hierarchical compile methodology.

Hierarchy

Hierarchy should be "pure?" down to leaf blocks. That is, there should be no mixture of RTL code and hierarchical instantiation. Within a leaf block, RTL code and hierarchical instantiation can be freely mixed.

SYNTHESIS PROCESS

Design Compiler optimizes logic designs for speed and area. This optimization is performed for hierarchical combinational or sequential circuit design descriptions. From the goals you define for measurable circuit characteristics, Design Compiler synthesizes a new circuit and puts it in a target technology.

The flowchart in Figure3.2 contains the following Design Compiler synthesis process steps:

For more information about each step, please refer to Synopsys On-Line Documentation (SOLD).

1. Read in the design and all its subdesigns
2. Set design attributes on the top-level design.
3. Set realistic timing or area goals for the design.
4. Run check_design to verify the design. Identify and correct any errors.
5. Select the scan style implementation.
6. Optimize the design with Design Compiler.
7. After optimization, run area and constraint reports to determine if design goals are met.
8. Run dft_drcto identify testability violations in the design. Identify and correct any violations.
9. Reoptimize after modifying attributes or constraints if goals are not met or to explore design space.
10. Run additional reports and schematics to analyze results further.

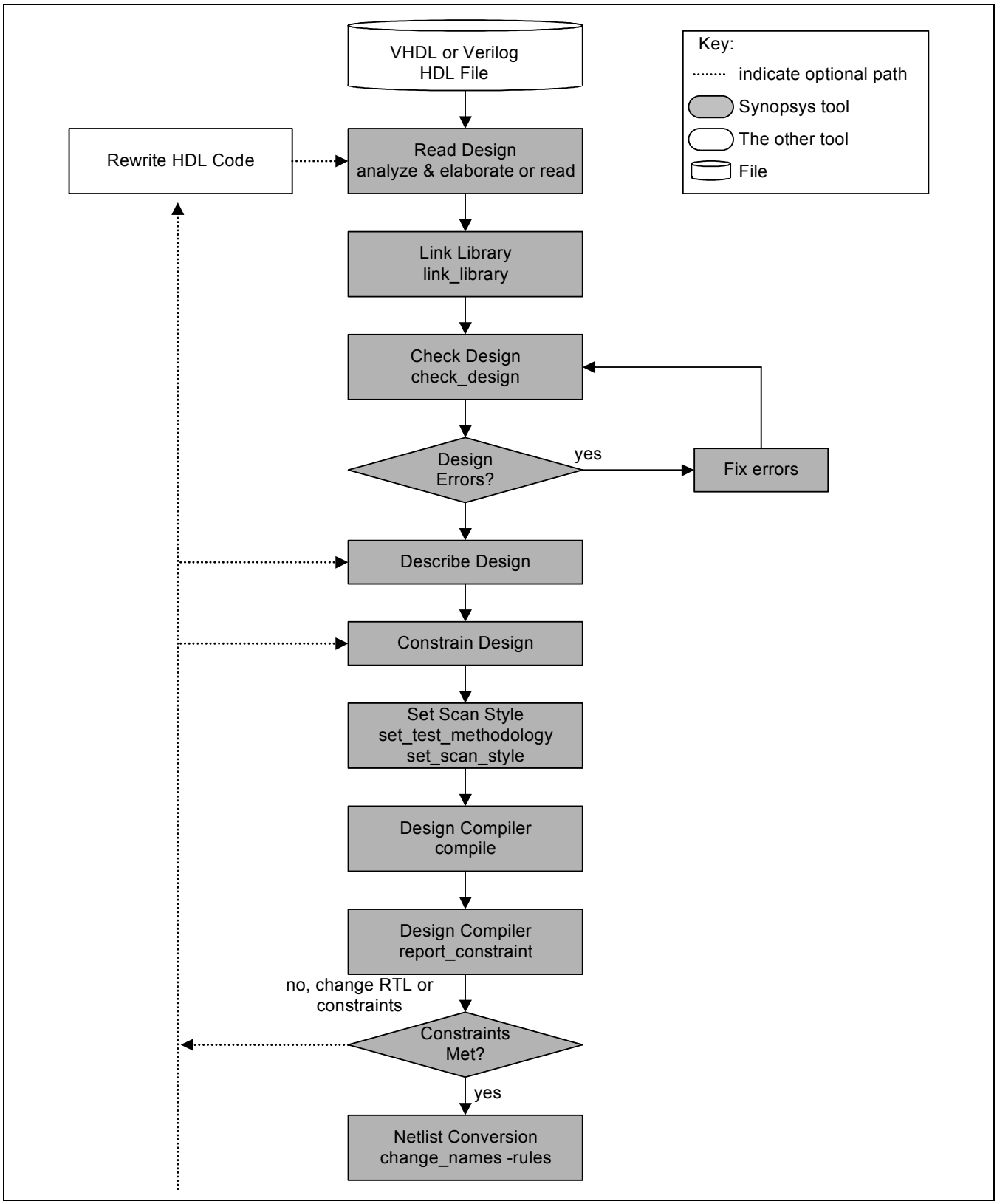

Figure 3-2. Synthesis Flowchart

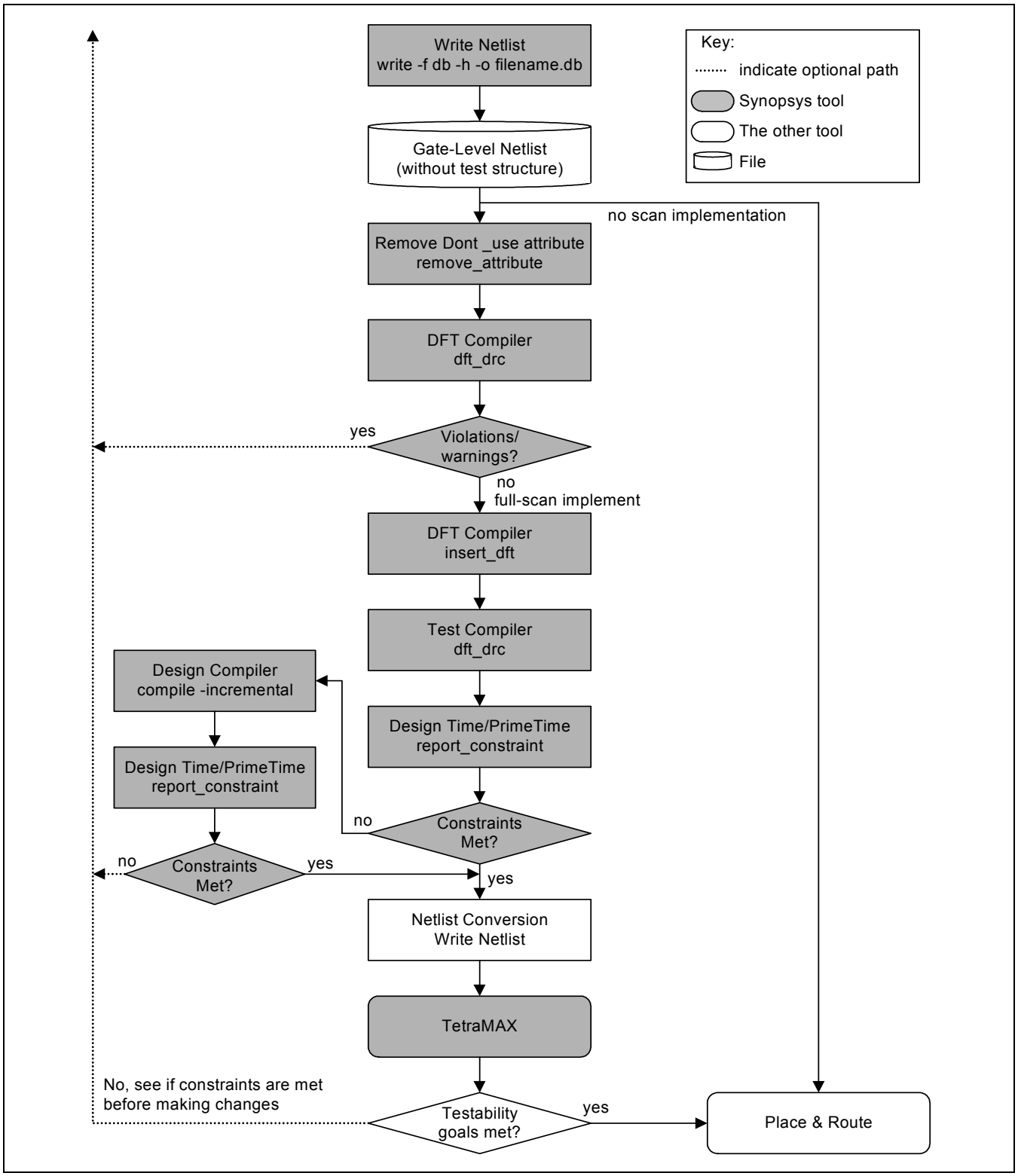

Figure 3-2. Synthesis Flowchart (Cont.)

EXAMPLE OF Design Compiler AND DFT Compiler SESSION

The section is a detailed example of a Design Compiler and DFT Compiler session for an SEC design.
To synthesize a design into a given technology, you could:

1. Set the following UNIX variables before using SEC Synopsys Design Kit: (for example, STD150E library)

```
%setenv SEC_SYNOPSYS <SEC Design Kit directory>/sec150e_synopsys"
%setenv SEC_SYNOPSYS_AUX <SEC Design Kit directory>/sec150e_synopsys/aux
%set path = ($SEC_SYNOPSYS/bin/sparcOS5 $path)
or
%set path = ($SEC_SYNOPSYS/bin/hp $path)
```

2. Set .synopsys_dc.setup file. Refer to Appendix A for details.

```
%cp $SEC_SYNOPSYS/etc/misc/synopsys_dc.setup.example .synopsys_dc.setup
```

3. Start the dc_shell interface, as described in this chapter.

```
%dc_shell

   Design Compiler (TM)
   . . .
   Initializing...
   dc_shell>
```

   Read in the design file (translate it into internal format) by entering the commands that work with the format of your design from the set of commands listed below.

   Use the following commands for a VHDL design file:

```
dc_shell> analyze -format vhdl {design_name.vhd}
dc_shell> elaborate design_name
or
dc_shell> read -format vhdl design_name.vhd
```

   Use this command for a Verilog design file:

```
dc_shell> analyze -format verilog {design_name.v}
dc_shell> elaborate design_name
or
dc_shell> read -format verilog design_name.v
```

4. Set the target library.

```
dc_shell> list -libraries
Library              File              Path
gtech                gtech.db          synopsys_path/libraries/syn
standard.sldb        standard.sldb     synopsys_path/libraries/syn
```

If it is not a proper technology library, set the target technology: (for example, STD150E library)

```
dc_shell> link_library = { "*" std150e_wst_105_p125.db }
dc_shell> target_library = { std150e_wst_105_p125.db }
dc_shell> symbol_library = { std150e_veri.sdb }
```

5. After reading in the design, check it for obvious errors, such as unconnected ports or recursively defined hierarchy.

```
dc_shell> check_design
```

6. Set design constraints, such as maximum area and maximum path delay to an output port..

```
dc_shell> set_max_area 0
```

7. If you are using Synopsys' DFT Compiler, set the scan style and test methodology.

   Design Compiler supports an optimization mode called test-smart compile. Test-smart compile modifies the sequential optimization algorithm, used during execution of the compile command, to prevent optimization of sequential cells that do not have scannable equivalents. Test-smart compile is invoked when the scan Implementation style is selected prior to executing the compile command.

```
dc_shell> set_scan_configuration -methodology full_scan
dc_shell> set_scan_configuration -style multiplexed_flip_flop
```

For details on scan styles and on test methodologies, please refer to Synopsys On-Line Documentation (SOLD).

Refer to $SEC_SYNOPSYS/etc/misc/Set_Dontuse_List.txt for information about scan cells of technology library.

8. Write the design out as a .db.

```
dc_shell> write -format db -hierarchy current_design -output pre_compile.db
```

9. Get the circuit reports you want.

```
dc_shell> report_area
dc_shell> report_timing
dc_shell> report_cell all_registers()
dc_shell> report_constraint
. . .
```

10. Use change_names command to interface between EDA tools.
    If you are writing out a Verilog netlist to layout your design with Apollo, use the following command:

```
dc_shell > change_names -rules sec_verilog -hierarchy
```

    If you are writing out a VHDL netlist to layout your design with Apollo, use the following command:

```
dc_shell > change_names -rules sec_vss -hierarchy
```

    Refer to 3.5.3 Netlist Conversion for details.

11. Write the design out as a .db. See "Compiling a Hierachical Design," of the Design Compiler Tutorial.

```
dc_shell> write -format db -hierarchy current_design -output pre_scan.db
```

    If the design do not use scan, you do not need to run DFT Compiler. If so, write the netlist (VHDL, or Verilog) and continue to the next step.

For a Verilog netlist:

```
dc_shell> write -format verilog -hierarchy current_design -output DESIGN_NAME.v
```

For a VHDL netlist:

```
dc_shell> write -format vhdl -hierarchy current_design -output DESIGN_NAME.vhdl
```

12. The scan cells of SEC technology library have the "dont_use" attribute. If you will insert scan, you should remove the attribute of them. Use the following command:

```
dc_shell> remove_attribute {library_name/f*s*} dont_use
```

13. In DFT Compiler, perform the initial test rules check on the gate-level netlist to identify testability violations in the design.

```
dc_shell> test_enable_dft_drc = true
dc_shell> create_test_protocol
dc_shell> dft_drc
```

    After you correct the problems that caused warning messages, you can proceed with the next step. (Failure to correct all the warning messages will typically result in lower fault coverage.) See "Checking Test Design Rules," in the DFT Compiler Reference Manual for more information on design rule checking.

14. Insert scan structures. Consult "Testability at the Module Level," in DFT Compiler Tutorial for this step and the next step. See "Adding Internal Scan Circuitry," in the DFT Compiler Reference Manual for more information on the scan circuits.

```
dc_shell> insert_dft
```

15. While using DFT Compiler, perform the test rules check on the gate-level netlist to identify testability violations in the design and to extract the scan chain information.

    ```
    dc_shell> dft_drc
    ```

16. Get the circuit reports you want. See "Analyzing Design Results," of the Design Compiler Tutorial.

    ```
    dc_shell> report_constraint -verbose
    dc_shell> report_timing
    ```

17. When you add scan-test circuitry to a design, its area can be changed. You can examine the area and speed costs with the appropriate report commands. (See the Design Compiler Family Reference Manual or the man pages for details.) Design Compiler can minimize the effect of adding scan-test circuitry on area and performance. To minimize the effect on performance and area on the scan circuitry, use the following command:

    ```
    dc_shell> set_dft_optimization_configuration -none
    ```

18. SEC does not support DFT Compiler to generate a set of sample test patterns. If you want to know Fault coverage roughly, use “dft_drc –verbose –coverage –sample 30” command.

19. While using “estimate_test_coverage” command, report the fault coverage from a random set of test vectors using TetraMAX’s ATPG engine. The report contains information on the fault coverage percentage achieved for all cells in the hierarchy of the design.

    If your testability goals are met, write the netlist (VHDL or Verilog) and continue to the next step.

    Otherwise, return to the netlist you wrote out before performing insert_dft (step 14). Correct warning messages from dft_drc that you failed to correct earlier.

    Low fault coverage may also be due to the design itself (not just DRC violations). So if dft_drc has no warning or error messages but the design still has low fault coverage, then examine the coverage and fault reports to locate modules with low fault coverage. Design changes, such as adding test modes, may be necessary.

20. To generate a set of pre-layout test patterns for the complete design, use Synopsys’s TetraMAX.

21. While using TetraMAX, report the fault coverage from a particular set of test vectors. The report contains information on the fault coverage percentage achieved for all cells in the hierarchy of the design.

22. Write out the patterns in the Verilog format. These vectors are used as inputs to the Verilog-XL Simulator before extracting tester ready patterns.

    The remainder of this chapter discusses Design Compiler and DFT Compiler concepts and commands and variables used in the example in the previous section.

LIBRARY DEPENDENT CODING STYLE

If reset (SN) and preset (RN) signals of ld4*cells are simultaneously active low, output signal Q is logic one and output signal QN is logic zero because the preset (SN) signal takes precedence of the reset (RN) signal. Therefore if you want to use such cells, you should describe VHDL and Verilog HDL as like the following:

1. VHDL description of LD4 cell

```
process (CLK, SETb, RESETb, DI)
        begin
              if SETb = '0' then
                    DO <= '1';
              elsif RESETb = '0' then
                    DO <= '0';
              elsif CLK = '1' then
                    DO <= DI;
              end if;
        end process;
```

2. Verilog description of LD4 cell

```
always @(CLK or SETb or RESETb or DI)
        begin
              if (!SETb)
                    DO <= 1'b1;
              else if (!RESETb)
                    DO <= 1'b0;
              else if (CLK)
                    DO <= DI;
        end
```

READING A DESIGN

You can load a design file using read for various representation files. Optionally, you can use analyze and elaborate for HDL.

The read Command

The read command performs two major tasks. In the first phase it does syntax error checking and Synthesis Policy Checking (SPC). In the second phase it infers 3-states, DesignWare components, and storage elements.

The analyze and elaborate Commands

You can use the analyze and elaborate commands as an alternative to the read command for HDL files. These commands should be used in sequence and apply only to HDL format files for VHDL or Verilog; they do not apply to db files.

The analyze command reads the HDL file into an intermediate format, checks syntax, and checks for synthesizable logic. The elaborate command creates a design from the intermediate format generated by the analyze command. Using defined parameters, elaborate searches the work library for the design to build.

Use the analyze and elaborate commands when you combine synthesis tasks with Synopsys VHDL System Simulator (VSS) Family tasks. If the -spc or -spc_elab option is passed to vhdlan, files analyzed during simulation need not be reanalyzed for synthesis because the synthesis and simulation tools use a common analyzer. Another advantage is that the analyze and elaborate commands support VHDL library management and simple VHDL configuration statements.

When you use VHDL and Verilog, the following example command sequences execute the default compile process:

VHDL Command Sequence

```
dc_shell> target_library = library_file_name.db
dc_shell> analyze -format vhdl example.vhd
dc_shell> elaborate example
dc_shell> compile
```

You can also use the read command with VHDL designs.

Verilog Command Sequence

```
dc_shell> target_library = library_file_name.db
dc_shell> read -format verilog example.v
dc_shell> compile
```

Netlist Naming Rules

Some typical netlist translation processes are described for reference. **You should copy or append $SEC_SYNOPSYS/etc/misc/synopsys_dc.setup.example to your .synopsys_dc.setup file.** Refer to Synopsys Design Kit User Guide: Appendix A for further information about the naming rules for netlist conversion.

Reading and Writing a Verilog Netlist

If you are reading in a Verilog netlist and writing out a Verilog netlist, set the following variables or **copy or append $SEC_SYNOPSYS/etc/misc/synopsys_dc.setup.example to your .synopsys_dc.setup file.**

```
bus_naming_style = "%s[%d]";
bus_dimension_separator_style = "][";
define_name_rules sec_verilog -type port \
                    - equal_ports_nets \
                    - allowed "A-Z a-z 0-9 _ [] !" \
                    - first_restricted "0-9 _ !" \
                    - last_restricted "_ !";

define_name_rules sec_verilog -type cell
                    - allowed "A-Z a-z 0-9 _ !" \
                    - first_restricted "0-9 _ !" \
                    - last_restricted "_ !" \
                    - map {{"\\*cell\\*","U"},{"*-return","RET"},{"]$","A"}};

define_name_rules sec_verilog -type net \
                    -equal_ports_nets \
                    -allowed "A-Z a-z 0-9 _ !" \
                    -first_restricted "0-9 _ !" \
                    -last_restricted "_ !" \
                    -map {{"\\*cell\\*","N"},{"*-return","RET"},{"]$","A"}};
```

You should use the following command before writing.

```
dc_shell> change_names -rules sec_verilog -hierarchy
dc_shell> write -h -o design_name.db
dc_shell> write -f verilog -h -o design_name.v
```

Reading and Writing a VHDL Netlist

If you are reading in a VHDL netlist and writing out a VHDL netlist, set the following variables or copy **or append $SEC_SYNOPSYS/etc/misc/ synopsys_dc.setup.example to your .synopsys_dc.setup file.**

```
bus_naming_style = "%s[%d]";
bus_dimension_separator_style = "][";
define_name_rules sec_vss -type port \
                      - equal_ports_nets \
                      - allowed "A-Z a-z 0-9 _  " \
                      - first_restricted "0-9 _  " \
                      - last_restricted "_ " \
                      - case_insensitive;

define_name_rules sec_vss -type cell \
                      - allowed "A-Z a-z 0-9 _  " \
                      - first_restricted "0-9 _  " \
                      - last_restricted "_ " \
                      - map {{"_$",""},{"\*cell\*","U"},{"*-return","RET"}} \
                      - case_insensitive;

define_name_rules sec_vss -type net \
                      -equal_ports_nets \
                      -allowed "A-Z a-z 0-9 _  " \
                      -first_restricted "0-9 _  " \
                      -last_restricted "_ " \
                      -map {{"_$",""},{"\*cell\*","U"},{"*-return","RET"}} \
                      -case_insensitive;
```

You should use the following command before writing.

```
dc_shell> change_names -rules sec_vss -hierarchy
dc_shell> write -h -o design_name.db
dc_shell> write -f vhdl -h -o design_name.vhd
```

Fixing Feedthroughs and Multiple Port Nets

The SEC ASIC flow does not allow feedthroughs or nets to connect multiple output ports. You should fix this problem using the set_fix_multiple_port_nets command before synthesis.

```
dc_shell> set_fix_multiple_port_nets -all -buffer_constants
```

In case of using command reoptimize_design, notice the following description. The SEC ASIC flow dose not allow feedthroughs or nets to connect multiple output ports. You should fix this problem setting compile_fix_multiple_port_nets variable before synthesis.

```
dc_shell> compile_fix_multiple_port_nets = "true";
```

DEFINING THE TARGET LIBRARY

Before you optimize a design or specify operating conditions, you need to define the target library. The target library variable determines the ASIC technology that the design is to be mapped to and contains definitions for the design's operating conditions.

Library Structure

An SEC Synopsys synthesis library provides comprehensive timing and functional descriptions of SEC macrocells in the Synopsys library format. The library also contains specifications of environmental delay conditions and wire load tables that correspond to those used for delay calculations.

Defining Library Variables

Technology libraries are usually identified through variables at the time you optimize a design. However, instead of typing in these variables for each session, you can define library variables in a .synopsys_dc.setup file in your design directory, as shown below: (for example, STD150E library)

```
search_path = {} + search_path
link_library = { "*" std150e_wst_105_p125.db };
target_library = { std150e_wst_105_p125.db };
symbol_library = { std150e_veri.sdb };
define_design_lib work -path work;
```

The search_path variable provides the tool with the directories in which to search for unresolved references.

The link_library variable names technology libraries and design files to use during the link process for a design. If a design references other design files, the design files should be identified with link_library so they can be located. (If a design's full path name is not in link_library, you should list the directories within which the design file resides in the search_path).

The target_library variable identifies a technology or a list of technology libraries of the components to use when you optimize a design.

The symbol_library variable identifies symbol libraries to be used during schematic generation. The value of this variable can be one or more symbol libraries.

The define_design_lib variable identifies the directory to be used by the analyze command for creation of intermediate files.

Setting Library Variables

Set the following Synopsys Design Compiler variables to allow the use of the SEC Synopsys synthesis library. You may enter them as defaults in the .synopsys_dc.setup file in your home directory or enter them on the command line during the Design Compiler session.

The library variables can also be entered into the setup file in the design directory, not just the home directory or on the command line.

If you are using the SEC Synopsys synthesis library, (for example, STD150E library)

set these variables in case of WORST condition library with SS / 1.05V / 125°C operating conditions:

```
link_library = { "*" std150e_wst_105_p125.db }
target_library = { std150e_wst_105_p125.db }
symbol_library = { std150e_veri.sdb }
```

or set these variables in case of BEST condition library with FF / 1.35V / -40°C operating conditions:

```
link_library = { "*" std150e_best_135_n040.db }
target_library = { std150e_best_135_n040.db }
symbol_library = { std150e_veri.sdb }
```

Library Support for DFT Compiler

One of Synopsys DFT Compiler's capabilities is inserting scan flip-flops. DFT Compiler swaps out non-scan flip-flops and inserts scan flip-flops while threading a scan chain. A requirement for using DFT Compiler is that every non-scan flip-flop should have a scan flip-flop counterpart, and every scan flip-flop in a design should have a non-scan counterpart.

Using multiplexed flip-flops (for example, fd1s, fd2s, fd3s, and fd4s), which is the most common scan implementation, is recommended. Other supported scan implementations include clocked scan (for example, fd2cs). Each scan implementation style has certain area and performance impacts on the original design.

The SEC Synopsys synthesis libraries provide limited support for the Synopsys DFT Compiler.(Scan insertion only)

Obtaining Cell Lists

You can view an on-line list of cells together with available operating conditions, wire load tables and cell attributes contained in the library by using the report_lib command in a Synopsys Design Compiler session.

The following example shows how to use this command to get information for the STD150E library:

```
dc_shell> report_lib std150e_wst_105_p125
```

SEC Synopsys Design Kit already includes the information of the Synopsys library as the output of "report_lib" command. Please refer to $SEC_SYNOPSYS/etc/report_lib/STD150E/std150e_wst_105p125.rpt.

RUNNING check_design

The check_design command verifies that the internal representation of the design is correct. Warning and error messages are issued, as appropriate. After constraints are set on a design and before the design is compiled, run the check_design command to identify and correct any problems in the design.

DESCRIBING DESIGNS

Variations in operating temperature, supply voltage, and the manufacturing process can strongly affect circuit performance (speed). These factors are called operating conditions. The SEC technology libraries have predefined sets of operating conditions and wire load models. To ensure that acceptable performance levels are maintained over a range of operating conditions, you can direct Design Compiler to reflect variations in process, temperature, and voltage. Wire load model estimates the effect of wire length and fanout on resistance, capacitance, and area.

Specifying the Wire Load Model

The wire_load_selection group that is defined in the SEC library allows Design Compiler to select a wire_load group automatically to use for wire load estimation. Selection is based on the total cell area of the design.

Wire load models contain all the information required by compile to estimate interconnect wire delays. By default, the wire load area is automatically selected based on the size of the module being optimized because of the default value, **auto_wire_load_selection = "true.**

If you set **auto_wire_load_selection = "false"** and you do not specify a wire load, then automatic wire load selection is disabled for that design. It is not recommended.

If you set a wire load model on a design (using the command **set_wire_load**), automatic wire load selection is disabled for that design.

The **report_lib** command shows the wire models that are defined in the SEC Synopsys technology library, as well as any wire_load_selection table that may be defined there.

Selecting Wire Load Model Group

The wire load selection groups are provided in the technology file. Four layer metal(4LM), five layer metal(5LM), six layer metal(6LM), and seven layer metal(7LM) groups are supported. “default_wire_load_selection” group is 4LM. In this case, if you want to use 7LM wire load model group, use the following dc_shell command:

```
dc_shell> set_wire_load_selection_group 7LM -library std150e_wst_105_p125
```

Hierarchical Wire Load Model

The default wire load mode for SEC Synopsys synthesis libraries is enclosed. The mode **enclosed** specifies using the wire load model of the smallest design that fully encloses the net. This mode is more accurate than mode top when cells in the same design are placed in a contiguous region during layout.

The **report_port** command can be used to display the wire_load_model of the output ports of a design. The **report_design** command displays the wire load model and mode for the current design, and the libraries to which the current design is linked. The report_clusters command shows the wire load associated with any particular cluster. The **report_wire_load** command can be used to provide a more detailed report on the wire_load_models associated with either a design or cluster.

For more information on wire load models, see "Linking to Physical Design Tools," of the Design Compiler Reference and "Wire Load Model," of the Floorplan Manager.

Overruling Pre-layout Interconnection Delay

For 90 series and later libraries, SEC supports pre-layout interconnection delay (wire load model) as well as cell delay. Interconnection delay is a function of resistance and capacitance including pin capacitance. Usually, most interconnect delay calculated using wire load model will be smaller than 2 nano seconds. However, there are some situations that pin capacitance of a cell is large. This results in unexpected large interconnection delay, i.e., abnormally large interconnection delay may be calculated due to the large pin capacitance. If you are sure that the large delay nets will actually have small interconnect delay after post-layout stage, you can change the large interconnection delay and annotate it for pre-layout delay calculation using user-defined interconnection delay.

After synthesis, run detect_longnet.scr in dc_shell to detect the nets whose interconnect delay is larger than 2 ns. This will generate a file called "long_delays". (You should set the UNIX variables before using Design Kit.

```
dc_shell> current_design top_module_name
dc_shell> include detect_longnet.scr
```

If you get message such as "There are nets having large...", your design includes nets having large delay(>= 2ns). Edit "long_delays" file which is generated by detect_longnet.scr in your working directory using text editor such as vi.

If you have changed values in the "long_delays" file, run new_delay.scr to annotate the user-defined interconnect delay to the modified nets.

```
dc_shell> include new_delay.scr
```

We recommend to contact your local SEC Design Center if you encounter this situation.

Defining Timing Environment

Because the current design is part of a larger system, you need to model the timing context of the design within that system. After a design is read into Design Compiler, information such as drive strengths on the ports, the time that signals arrive on the ports, or the load driven by the output ports needs to be conveyed to Design Compiler.

The set_input_delay command captures information about delays of paths leading to input ports relative to clock signals. The set_output_delay command captures information about delays of paths leading from output ports relative to clock signals. Along with clock, drive, and load information, input and output delay information is usually sufficient to capture the timing environment of the design.

This information is passed on by setting attributes on the design. The design attributes that you set give Design Compiler additional information about the design's environment. Some attributes affect the components selected from your library when the design is optimized.

The default attribute values are unrealistic, so the results of optimization on circuits that do not have attributes set are usually not optimal. For example, input ports have a default drive value of zero (or infinite drive strength), and output ports have a default load of zero. If the input port drives a heavily loaded net, Design Compiler does not add buffers to support the net drive, because the port is defined as having infinite drive strength.

Timing on the design is also affected by attribute values. Drive strengths of the input ports and output port loading are some of the values used for timing calculations. If attribute values are not set, timing results are too optimistic.

Note that for cells that have an internal node that also acts as an output, delay values are accurate only under certain circumstances.

Refer to "Specifying Clocks and Clock Networks," and "Describing Logic Functions and Signal Interfaces," of the Design Compiler Reference: Constraints and Timing Manual for additional information on the commands discussed below.

Setting Drive Strength

The **set_driving_cell** command sets attributes on input or inout ports specifying that a library cell or pin will drive

the ports. The transition delay of the ports is determined by the drive capability of the cell.

For internal nodes, the output of an inverter is a reasonable default cell. For default loading, use perhaps five times the inverter load as a reasonable default cell. For module pins that connect directly to I/O buffers, use appropriate I/O buffers for loading and drive.

The following example command associates the drive capability of the library cell nd2 with the port I1.

```
dc_shell> set_driving_cell -cell nd2d1_hd {I1}
```

Use **report_port -drive** to view drive information on ports.

Creating a Clock

Sequential circuits always have clocks. A combinational module is not fed by a clock. You should create virtual clocks in the design before setting input or output delays. Using the create_clock command, if a clock_name is given but no pins or ports are specified as clock sources in the current design, a virtual clock is created. A virtual clock can be created to represent an off-chip clock for input or output delay specification.

The following example command creates a clock with the name CLK having a period of 20 time units:

```
create_clock -name CLK -period "20" -waveform { "0" "10" } { CLK }
```

To constrain designs, accurately capture the timing environment for most sequential designs by using the **set_input_delay** and **set_output_delay** commands to set input and output delays relative to a clock.

Use **report_clock** to display all of the design's clock-related information.

Refer to "Specifying Clocks and Clock Networks," of the Design Compiler Reference: Constraints and Timing Manual for additional information on **create_clock**.

Setting Input Delays

Input delays are used to model the external delays arriving at the input ports of the constrained module. These delays are defined relative to a real or virtual clock and are specified to the right of the active clock edge.

Setting Output Delays

Output delays are used to model the external delays leaving the output ports of the constrained module. These delays should be defined relative to a real or virtual clock to be considered as a path constraint. Output delays are defined to the left of the active clock edge; this delay corresponds to the time before the next rising edge.

Setting Load Values

Load values are used to model the capacitive load on the output ports of the constrained module. For example, the following command:

```
dc_shell> set_load 5 * load_of(std150e_wst_105_p125/ivd1_hd/A) all_outputs()
```

describes a capacitive load of five times the load of an ivd1_hd element in the std150e library.

To view load values on ports, use the **report_port** command.

Refer to "Describing Logic Functions and Signal Interfaces," of the Design Compiler Reference: Constraints and Timing Manual for additional information on **set_load.**

CONSTRAINING DESIGNS

Design Compiler uses constraints to guide the optimization and implementation of a design in a given technology. Constraints define the goals of the synthesis process. Before a design is optimized, you need to define area and timing goals for the design. Design Compiler then tries to meet these goals when synthesizing your design.

The constraints should be realistic to get the most effective results from Design Compiler. With realistic timing goals, Design Compiler produces the smallest circuit that most closely satisfies the goal.

Constraints refer to measurable circuit characteristics such as timing and area. Design-rule constraints (such as max_capacitance and max_transition) reflect technology-specific restrictions that should be met for a functional design, and optimization constraints represent design goals and restrictions that are not as crucial to the operation of the design. See $SEC_SYNOPSYS/etc/misc/ Max_Transition.txt file for the information about default max_transition time of SEC technology library.

The max_capacitance constraint is specified implicitly in the SEC technology library. Maximum delay, minimum delay, and maximum area are optimization constraints.

To get the most effective results from Design Compiler, set your optimization constraints to be close to your design goals. Sometimes you can improve the results by setting the constraints slightly higher (1% to 10%) than needed, but improvement is not guaranteed. If a timing goal is not set (the default), only design rule constraints are applied to the design. If timing goals are too restrictive, Design Compiler adds buffers to critical paths or duplicates logic on heavily loaded nets to try to meet these goals, resulting in a significant increase in area.

Design Compiler optimizes a design for delay and area using a cost function. Area is less important than max_delay and setup. Therefore Design Compiler accepts a change that improves max_delay and setup cost but degrades area cost. It does not accept a change that improves area cost if the resulting delay cost is worse.

When a subdesign needs to be optimized, use the characterize command to generate required constraints for the subdesign.

In most cases it is best to set realistic optimization goals. For example, do not specify a max_area of 0 and a clock with period 0 if you really need a design with a 15 ns clock period that fits in an array containing 30,000 units of usable area. Set a realistic max_area constraint instead.

Usually the clock period, timing numbers, and area of the circuit are given in the specification of a design. Sometimes, however, goals may not be available in a design specification, or only a portion (subdesign) of a larger design may need to be optimized. In the latter case, goals may be known for the entire design but not for the subdesign.

For cases when realistic goals are unknown, it is best to map the design or subdesign to gates, without setting constraints, to determine the current design speed. From these results, you determine the design constraints to be set. Then you recompile the design.

If the design is already in netlist format (mapped to gates), you can use the **derive_timing_constraints** command to extract the constraints.

SDF BACK-ANNOTATION

You can do the Static Timing Analysis with timing data(SDF) generated by CubicDelay. In order to get the correct SDF file from the CubicDelay (SEC In-House delay calculator), you should select the correct option when you run the CubicDelay.

SEC Synopsys Design Kit does not contain CubicDelay, CubicDelay is included in SEC Utility.

For STA or resynthesis, delay should be back-annotated using SDF file. You should use CubicDelay to generate a SDF file containing best, typical, and worst case delay information.

If you use commands as follows, you can generate proper type of SDF which can be back-annotated to Synopsys.

(After invoking CubicDelay, you can command it at the command line.)

SDF File Generation

In case of verilog user, use the following commands.

```
Cubic> read -format verilog design.v
...
Cubic> write_timing -format verilog_sta_sdf -mode all SDF_file_name
```

In case of VHDL user, use the following commnads.

```
Cubic> read -format vhdl design.vhd
...
Cubic> write_timing -format vhdl_sta_sdf -mode all SDF_file_name
```

Refer to CubicDelay User's Manual for details.

PrimeTime Back-annotation

To static timing analysis using PrimeTime, you should back-annotate the SDF file generated by CubicDelay to your design. Use the following command.

```
pt_shell> read_sdf -type sdf_min | sdf_max SDF_file_name
```

Refer to Static Timing Analysis Sign-off (with PrimeTime) for details.

IN CASE OF CHIP LEVEL DESIGN

Chip level design means that a design includes IO PAD cells. In case of chip level design, Design-Compiler/PrimeTime's delay calculation can be different from what a designer is expected. Because Design-Compiler/PrimeTime calculate interconnect delay from IO PAD to port. Designer should remove nets from IO PAD to port.

Design-Compiler Feature

Design-Compiler/PrimeTime calculate interconnect delay from IO PAD to port as shown in Figure 3.3

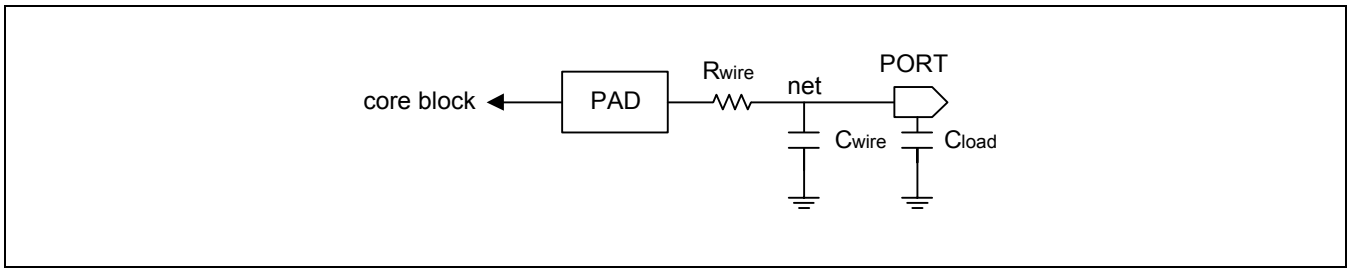

Figure 3-3. Design-Compiler Feature

Note that interconnect net between PAD and virtual PORT does not exist in reality. But Design-Compiler calculates total capacitance of interconnect net and PAD's output using the following equation.

total_capacitance = **C**wire + **C**load

interconnect_delay of net = **R**wire (**C**wire + **C**load)

How to solve above problem

To avoid this problem, use the following commands.

```
dc_shell> set_load 0 net
dc_shell> set_resistance 0 net
```

If your design's port name is different from net name connected to port, you should command manually above.

If your design's port name is the same as the net name connected to port, SEC Synopsys Design Kit supports the script to avoid this problem.

To use the script, follow the steps below.

Step 1: If you did not set UNIX variable, SEC_SYNOPSYS_AUX, set it as below:

```
% setenv SEC_SYNOPSYS_AUX $SEC_SYNOPSYS/aux
```

Step 2: Run "del_cap_res.scr" to remove load and resistance of port's net.

```
dc_shell> include del_cap_res.scr
```

This script first finds all ports. Then it sets load value and resistance value of net connected to port to zero.

CLOCK DEFINITION WITH PLL-EMBEDDED DESIGN

In PLL-embedded design, there are two main points to keep in mind when defining clocks:Note that PLL has jitter and duty ratio. You should reflect these factors when defining a clock source.

For a design including PLL, the output of the PLL should be defined as a clock source. In this case, you may propagate the clock source through clock tree in post-layout stage. Figure 3.4 illustrates this situation.

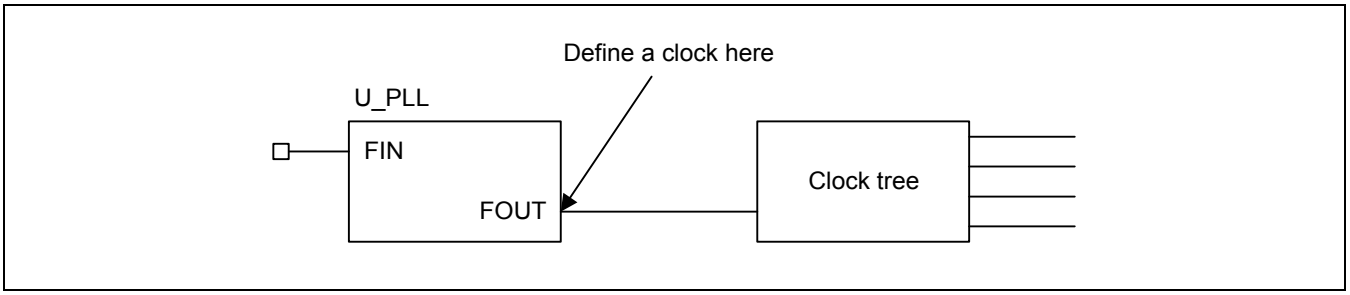

Figure 3-4. Clock Definition of PLL-embedded Design

Note that PLL has jitter and duty ratio. You should reflect these factors when defining a clock source. For example, for a PLL having (+/-)150ps jitter and 45% to 55% duty ratio at 100MHz, you can define the clock source as follows:

Define the clock at the output pin of PLL. Note that U_PLL is the PLL instance name in your netlist.

```
dc_shell> create_clock -period 10 -waveform {0 5} -name GCK find(pin,
"U_PLL/FOUT")
```

Define clock uncertainty to reflect jitter. The uncertainty value applied due to PLL jitter in our example is 0.30ns.

```
dc_shell> set_clock_uncertainty -setup 0.30 GCK
dc_shell> set_clock_uncertainty -hold 0.30 GCK
```

Duty Ratio

SEC ASIC recommends not to use mixed-clock edge design when using PLLs (synthesizer and deskew PLL). In the examples above, duty ratio is not reflected. Actually, duty cycle should be reflected to correctly analyze mixed-clock edge timing paths. In this case, the uncertainty value would be 0.3+0.5 = 0.8ns. To correctly analyze rising edge to falling edge timing path and falling edge to rising edge timing paths, this uncertainty value should be applied to both clock edges, as shown below.

```
dc_shell> set_clock_uncertainty -setup 0.80 GCK
dc_shell> set_clock_uncertainty -hold 0.80 GCK
```

Such uncertainty due to both jitter and duty may severely restrict your logic implementation in mixed-clock edge design. Also, this uncertainty definition causes false uncertainty of 0.5ns between single-clock edge timing paths, which leads many false violations.

ADDITIONAL CASE OF DESKEW PLL IN USING Design Compiler

For a design including a deskew PLL, PLL compensates phase delay (seen at flip-flop clockpins) caused by the clock tree. If you want to define a clock on the input pin of deskew- PLL (SEC ASIC does not recommend design case), you can define the clock source as follows.

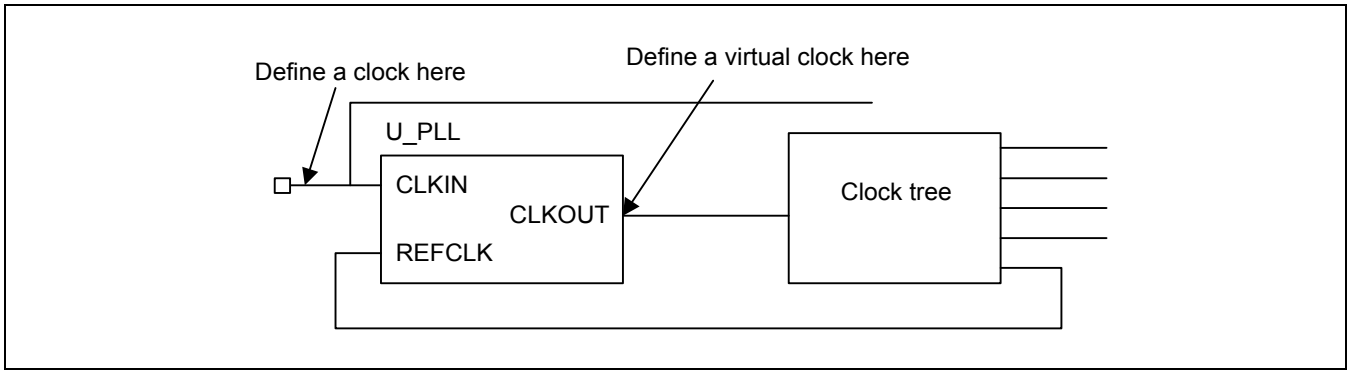

Figure 3.5 In Case of Extra Connection with the Input of Deskew PLL

As a result, clock definition of deskew PLL should reflect this compensated phase delay, in addition to jitter and duty ratio. For example, for a PLL having (+/-)150ps jitter and 45% to 55% duty ratio at 100MHz, you can define the clock source as follows:

Define the source clock of PLL. Assume that the source clock frequency is 50MHz.

```
dc_shell> create_clock -period 20 -waveform {0 10} find(port,"RCLK")
```

Define the output clock of PLL using the source clock of PLL. Note that U_PLL is the instance name of PLL in your netlist.

```
dc_shell> create_clock -period 10 -waveform {0 5} -name GCK find(pin,
"U_PLL/CLKOUT")
```

Define clock uncertainty to reflect jitter. The uncertainty value applied due to PLL jitter in our example is 0.30ns.

```
dc_shell> set_clock_uncertainty -setup 0.30 GCK
dc_shell> set_clock_uncertainty -hold 0.30 GCK
```

Define negative clock source latency of the generated clock to reflect compensated phase delay. The latency value is the clock tree insertion delay. For post-layout stage, the approximated insertion delay value can be obtained from clock tree synthesis reports,or DesignTime delay calculation (after CubicDelay-generated SDF back-annotation). For pre-layout stage, we have two cases. When CTC flow is used, CTC insertion delay can be used as the insertion delay. Without CTC flow, you can use estimated insertion delay to be satisfied during post-layout stage. Note that INSERTION_DELAY should be negative value.

```
dc_shell> set_clock_latency -min -max -source INSERTION_DELAY GCK
```

You should also define inter-clock uncertainties between RCLK to GCK.

4 APPENDIX

A. synopsys_dc.setup.example

In case of STDE150 Design Kit, synopsys_dc.setup.example is the following text.

```
/*****************************************************************************
**               Synopsys environment file for STDE150 library                **
*****************************************************************************/


/*****************************************************************************
**               Set New Variable for "search_path"                           **
*****************************************************************************/
dk_home = get_unix_variable("SEC_SYNOPSYS")
dk_home_aux = get_unix_variable("SEC_SYNOPSYS_AUX")

/*****************************************************************************
**               Set 'search_path'                                            **
*****************************************************************************/
search_path = { . synopsys_root + "/libraries/syn" dk_home + "/syn/STD150E"
   dk_home_aux};

/*****************************************************************************
**               Set libraries                                                **
*****************************************************************************/
link_library   = {"*" std150e_wst_105_p125.db} ;
target_library = {std150e_wst_105_p125.db} ;

/* link_library   = {"*" std150e_bst_135_n040.db} ; */
/* target_library = {std150e_bst_135_n040.db} ; */

/* link_library   = {"*" std150e_typ_120_p025.db} ; */
/* target_library = {std150e_typ_120_p025.db} ; */

symbol_library = {std150e_veri.sdb} ;
verilog_write_unmapped_design = "true";
change_name_use_ultanaative = "false";

/*****************************************************************************
```

```
** NAMING RULE FOR Verilog HDL:                                          **
** If you have a plan to layout your chip with Apollo,                   **
** you MUST use this naming rule.                                  **
**  [1] max_length war removed                                     **
**  [2] '!' character was allowed for CTS-ed netlist.                    **
*************************************************************************/
define_name_rules    sec_verilog   - type port \
                        - equal_ports_nets \
                        - allowed "A-Z a-z 0-9 _ [] !" \
                        - first_restricted "0-9 _ !" \
                        - last_restricted "_ !" \

define_name_rules    sec_verilog   - type cell \
                        - allowed "A-Z a-z 0-9 _ !" \
                        - first_restricted "0-9 _ !" \
                        - last_restricted "_ !" \
                        - map {{"\\*cell\\*","U"},{"*-return","RET"},{"]$","A"}};

define_name_rules    sec_verilog   - type net \
                        - equal_ports_nets \
                        - allowed "A-Z a-z 0-9 _ !" \
                        - first_restricted "0-9 _ !" \
                        - last_restricted "_ !" \
                        - map {{"\\*cell\\*","N"},{"*-return",”RET"},{"]$","A"}};


/*************************************************************************
** NAMING RULE FOR VHDL:                                                 **
** If you are VHDL user, you MUST use this naming rule.                  **
*************************************************************************/
define_name_rules    sec_vss   - type port \
                        - equal_ports_nets \
                        - allowed "A-Z a-z 0-9 _ " \
                        - first_restricted "0-9 _ " \
                        - last_restricted "_ " \
                        - case_insensitive

define_name_rules    sec_vss   - type cell \
                        - allowed "A-Z a-z 0-9 _ " \
                        - first_restricted "0-9 _ " \
                        - last_restricted "_ " \
                        - map {{"_$",""},{"\*cell\*","U"},{"*-return","RET"}} \
                        - case_insensitive

define_name_rules    sec_vss   - type net \
                        - equal_ports_nets \
```

```
                        - allowed "A-Z a-z 0-9 _ " \
                        - first_restricted "0-9 _ " \
                        - last_restricted "_ " \
                        - map {{"_$",""},{"\*cell\*","U"},{"*-return","RET"}} \
                        - case_insensitive

/*****************************************************************************
**                MISC                                                      **
*****************************************************************************/
/* These variables affects compile, report_timing and report_constaints commands. */
enable_recovery_removal_arcs = “true”;
/* To get non-functional Verilog-HDL Netlist */
verilogout_no_tri = "true";
/* To allow infer latches which have async. reset/set pin */
/* hdlin_latch_always_async_set_reset = "true";  */
/* To fix the design which have multiple ports */
if (compatibility_version < "1999.10"){
   echo "     ************************************************"
   echo "          NOTE>>> SEC does NOT suport " + compatibility_version
   echo "     ************************************************"
} else {
   echo "     ********************************************************************"
   echo "          NOTE>>> You MUST fix the design which have multiple ports";
   echo "                  Use the following command";
   echo "                  set_fix_multiple_port_nets -all -buffer_constants";
   echo "     ********************************************************************"
   alias compile "set_fix_multiple_port_nets -all -buffer_constants;compile";
   echo "     ********************************************************************"
   echo "          NOTE>>> Use the old variable compile_fix_multiple_port_nets";
   echo "                  because reoptimize_design does not honor";
   echo "                  the set_fix_multiple_port_nets command";
   echo "     ********************************************************************"
   alias reoptimize_design "compile_fix_multiple_port_nets =
   "true";reoptimize_design";
}
/* To fix the auto_wire_load selection */
auto_wire_load_selection = "true" ;

/*****************************************************************************
**                Link-to-Layout Flow,Resynthesis                           **
*****************************************************************************/
compile_create_wire_load_table = "true" ;
lbo_lfo_enable_at_pin_count = 3;
lbo_cells_in_regions = "false";
pdefin_use_nameprefix = "false";
enable_slew_degradation = "true";
```

```
/*****************************************************************************
**               For VHDL User                                              **
*****************************************************************************/
/* "use" clause is written into the VHDL file automatically */
vhdlout_use_packages = {"IEEE.std_logic_1164"};
/* VHDLout writes a configuration statement if necessary */
vhdlout_write_top_configuration = "true";
/* VHDLout writes a architecture_name as SCHEMATIC */
vhdlout_architecture_name = "SCHEMATIC";
/* All ports on lower-level designs are written with their original data types.
This value affects only designs that are read in VHDL format. */
vhdlout_preserve_hierarchical_types = "USER";

/*****************************************************************************
  If you want to get more information about the interface between EDA tools
  Then refer the SYNOPSYS Reference Manuals
*****************************************************************************/
```

NOTES