Design Compiler™
Tutorial

Version 2000.05, May 2000

Comments?
E-mail your comments about Synopsys
documentation to doc@synopsys.com

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> “This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of ____ and its employees. This is copy number __________.”

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader’s responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks

Synopsys, the Synopsys logo, AMPS, Arcadia, CMOS-CBA, COSSAP, Cyclone, DelayMill, DesignPower, DesignSource, DesignWare, dont_use, EPIC, ExpressModel, Formality, in-Sync, Logic Automation, Logic Modeling, Memory Architect, ModelAccess, ModelTools, PathBlazer, PathMill, PowerArc, PowerMill, PrimeTime, RailMill, Silicon Architects, SmartLicense, SmartModel, SmartModels, SNUG, SOLV-IT!, SolvNET, Stream Driven Simulator, Synopsys Eagle Design Automation, Synopsys Eagle*i*, Synthetic Designs, TestBench Manager, and TimeMill are registered trademarks of Synopsys, Inc.

Trademarks

ACE, BCView, Behavioral Compiler, BOA, BRT, CBA, CBAII, CBA Design System, CBA-Frame, Cedar, CoCentric, DAVIS, DC Expert, DC Expert *Plus*, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Compiler, DesignTime, Direct RTL, Direct Silicon Access, dont_touch, dont_touch_network, DW8051, DWPCI, ECL Compiler, ECO Compiler, Floorplan Manager, FoundryModel, FPGA Compiler, FPGA Compiler II, FPGA *Express*, Frame Compiler, General Purpose Post-Processor, GPP, HDL Advisor, HDL Compiler, Integrator, Interactive Waveform Viewer, Liberty, Library Compiler, Logic Model, MAX, ModelSource, Module Compiler, MS-3200, MS-3400, Nanometer Design Experts, Nanometer IC Design, Nanometer Ready, Odyssey, PowerCODE, PowerGate, Power Compiler, ProFPGA, ProMA, Protocol Compiler, RMM, RoadRunner, RTL Analyzer, Schematic Compiler, Scirocco, Shadow Debugger, SmartModel Library, Source-Level Design, SWIFT, Synopsys EagleV, Test Compiler, Test Compiler Plus, Test Manager, TestGen, TestSim, TetraMAX, TimeTracker, Timing Annotator, Trace-On-Demand, VCS, VCS Express, VCSi, VERA, VHDL Compiler, VHDL System Simulator, Visualyze, VMC, and VSS are trademarks of Synopsys, Inc.

Service Marks

TAP-in is a service mark of Synopsys, Inc.

All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.

Document Order Number: 00386-000 IA
Design Compiler Tutorial, v2000.05

Contents

About This Tutorial

1. Introduction to Design Compiler

2. Design Analyzer Graphical Interface Fundamentals

Appendix A. Tutorial Script Files

Appendix B. UNIX and the Windows NT OS for Synthesis Products

Figures

Tables

About This Tutorial

The *Design Compiler Tutorial* explains how to begin using Design Compiler tools. It explores the two interfaces of the Design Compiler and includes a set of exercises that you can work through to optimize and compile a sample design. The book is organized into two main parts, followed by Appendixes. The two main parts are

- I: Getting to Know Design Compiler and Its Interfaces

  Part I explains the basics of synthesis using Design Compiler. It also describes the two interfaces of Design Compiler: the Design Analyzer graphical user interface (GUI) and the Design Compiler dc_shell command-line interface (CLI).

- II: Using Design Compiler: A Tutorial Session

  Part II explains how to set up your system to run Design Compiler and describes the sample Alarm Clock design used for the tutorial. Then, using the sample design, it steps you through procedures that describe how to optimize a simple circuit. Description of each procedure begins with pertinent conceptual information.

Following the main parts of the book is a set of appendixes that identify scripts files that replicate the tutorial procedures, provide a description of the differences imposed on the operation of the

Synopsys synthesis tools by the underlying operating system (whether UNIX or Windows for NT), give instructions on creating a home directory in Windows NT, and provide general product information.

This preface includes the following sections:

- Audience
- Related Publications
- SOLV-IT! Online Help
- Customer Support
- Conventions

Audience

This manual is for engineers who are familiar with ASIC design but are not familiar with Design Compiler. A working knowledge of high-level design techniques, a hardware description language such as VHDL or Verilog, the operating system for your computer, and various commands derived from the UNIX operating system is assumed.

Related Publications

For additional information about Design Compiler, see

- Synopsys Online Documentation (SOLD), which is included with the software
- Documentation on the Web, which is available through SolvNET on the Synopsys Web page at http://www.synopsys.com
- The Synopsys Print Shop, from which you can order printed copies of Synopsys documents, at http://docs.synopsys.com

You might also want to refer to the following documentation for the following related Synopsys products:

- *Design Compiler User Guide*
- *Design Compiler Command Line Interfaces Guide*
- *Design Compiler Reference Manual: Constraints and Timing*

- *Design Compiler Reference Manual: Optimization and Timing Analysis*
- *PrimeTime Reference Manual*
- *FPGA Compiler User Guide*
- *VHDL Compiler Reference Manual*
- *HDL Compiler for Verilog Reference Manual*
- *Test Compiler Reference Manual*
- *Library Compiler Reference Manual,* volumes 1 and 2
- *DesignWare Developer Guide*

SOLV-IT! Online Help

SOLV-IT! is the Synopsys electronic knowledge base, which contains information about Synopsys and its tools and is updated daily.

To obtain more information about SOLV-IT!,

1. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNET.
2. If prompted, enter your user name and password. If you do not have a SOLV-IT! user name and password, you can obtain them at http://www.synopsys.com/registration.

Customer Support

If you have problems, questions, or suggestions, contact the Synopsys Technical Support Center in one of the following ways:

- Open a call to your local support center from the Web.

  a. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNET (SOLV-IT! user name and password required).

  b. Click “Enter a Call.”

- Send an e-mail message to support_center@synopsys.com.

- Telephone your local support center.

  - Call (800) 245-8005 from within the continental United States.

  - Call (650) 584-4200 from Canada.

  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax. In command syntax and examples, shows system prompts, text from files, error messages, and reports printed by the system.
italic	Indicates a user specification, such as *`object_name`*

Convention	Description
bold	In interactive dialogs, indicates user input (text you type).
[ ]	Denotes optional parameters, such as *`pin1`* `[`*`pin2 ... pinN`*`]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Part I: Getting to Know Design Compiler and Its Interfaces

1

Introduction to Design Compiler

Design Compiler synthesizes your HDL description into a technology-dependent, gate-level design. Using Design Compiler, you define the environmental conditions, constraints, compile methodology, design rules, and target libraries to achieve your design goals.

Design Compiler is the core of the Synopsys synthesis software products. It provides constraint-driven sequential optimization and supports a wide range of design styles.

This chapter includes topics that introduce aspects of Design Compiler in the following sections:

- What’s New in This Release
- Synthesis Process
- Design Styles

- Input Formats
- Output Formats
- User Interfaces
- Files and Directories
- Script Files
- Supported Operating Systems
- Preliminary Requirements

What's New in This Release

This section describes the new features, enhancements, and changes included in Design Compiler version 2000.05. Unless otherwise noted, you can find additional information about these changes in the Design Compiler documentation set.

New Features

Design Compiler version 2000.05 includes the following new features:

- High Performance Arithmetic Component Generator

  Provides a mechanism in Design Compiler that uses Module Compiler as the arithmetic component generator as well as the datapath block generator.

  - Arithmetic Generator Component

    Module Compiler generated architectures are selected for arithmetic components like adders, multipliers, vector sum, and sum-of-product operations.

  - Specialized Architecture for Sum-of-Products

    Module Compiler datapath synthesis capabilities are used to build carry-save adder structures for designs containing vector sums and sum-of-product operations. This feature introduces the new `partition_dp` command.

  For additional information, refer to *Design Compiler Reference Manual: Constraints and Timing*.

- New Generated Clocks Feature

  The `create_generated_clock` command permits a user to generate a new clock, based on an already existing clock, at a specified list of pins or ports of the design. The period and waveform of the newly generated clock can be varied from that of the original clock with arguments specified on the command line. The key advantage of this command is that when parameters controlling the original clock are changed, these changes will be automatically reflected in the clocks generated from it. This feature is consistent with the one in PrimeTime. For additional information, refer to *Design Compiler Reference Manual: Constraints and Timing*.

- Input Parasitics

  The new `set_input_parasitics` command

  - Models the effect of external RC on delay to input ports.
  - Improves accuracy of signal arrival times at input ports by accounting for the delay of long nets (from top-level routing) between blocks during synthesis.

For additional information, refer to *Design Compiler Reference Manual: Constraints and Timing*.

- Pipeline Retiming

  The new pipeline retiming feature is a multiclass retiming capability that retimes asynchronous registers and also improves area and delay for designs with mixed asynchronous and synchronous registers. Design Compiler supports retiming on designs containing subblocks that have the `dont_touch` attribute set.

- RTL Load Annotation

  The RTL-load-annotation capability provides more accurate wire loads for top-level nets during early synthesis. It allows users to annotate a load value greater than the load suggested by a statistical wire load model on specific nets. The annotations are retained throughout the synthesis process and are used during structuring, mapping, and placement-independent optimizations. The net with the annotated load can undergo optimization and still retain the annotated load value. Nets without RTL-load annotations use statistical wire load models for optimization. For additional information, refer to *Design Compiler Reference Manual: Optimization and Timing Analysis*.

- Automated Chip Synthesis (ACS)

  Automated Chip Synthesis (ACS) is a new feature in Design Compiler version 2000.05 that automates a divide and conquer strategy for chip-level synthesis in a single command. For more information please refer to the ACS User Guide.

Enhancements

Design Compiler version 2000.05 includes the following enhancements:

- Area Optimization Improvements

  Provide further improvements in area optimization by building on changes incorporated for the 1999.10 release. The principal target designs are those designs that are loosely constrained for delay. These designs tend to meet delay optimization goals easily.

- Hold Time Fixing Enhancements

  - Improves the hold time fixing algorithms to provide faster runtimes and more complete fixing with the smallest possible impact to delay and design area.
  - Adds an option to prefer cell count over area in hold time fixing phase.
  - Enhances the `set_prefer -min` feature so you can specify a list of buffer/inverter cells to be used for hold time fixing.

- Enhancements to `report_timing`

  New options have been added to `report_timing` to improve ease of use and provide more details for analysis. The new options are `-capacitance` and `-sort_by`. The existing `-path` option supports a new argument, `full_clock`.

- `-of_objects` reporting function in Tcl

  The `get` command `-of_objects` option creates a collection of cells, libraries, nets, pins, ports, library cells, and library cell pins connected to the specified object. For additional information, refer to *Design Compiler Command-Line Interface Guide*.

Changes

Design Compiler version 2000.05 includes the following changes:

- Improved `set_clock_latency` Command

  The `set_clock_latency` command has been enhanced to support the `-early/-late` option for specifying clock source latency. Clock source latency is the time a clock signal takes to propagate from its ideal waveform origin to the clock definition point in the design. Clock source latency can be applied to ideal or propagated clocks. With the `-early/-late` option, users can estimate the fastest (earliest) and the slowest (latest) times for the clock edge to reach the clock definition point. For additional information, refer to *Design Compiler Reference Manual: Constraints and Timing*.

- Setting implicit `size_only`

  Improves Design Compiler usability by setting an implicit `size_only` attribute where the implicit `dont_touch` attribute has been set. With this change the cell can be mapped and resized and still preserve the reference pin.

Known Limitations and Resolved STARs

Information about known problems and limitations, as well as about resolved Synopsys Technical Action Requests (STARs), is available in the *Design Compiler Release Notes* in SolvNET.

To see the *Design Compiler Release Notes,*

1. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNET.

2. If prompted, enter your user name and password. If you do not have a SOLV-IT! user name and password, you can obtain them at http://www.synopsys.com/registration.
3. Click Release Notes, then open the *Design Compiler Release Notes.*

Synthesis Process

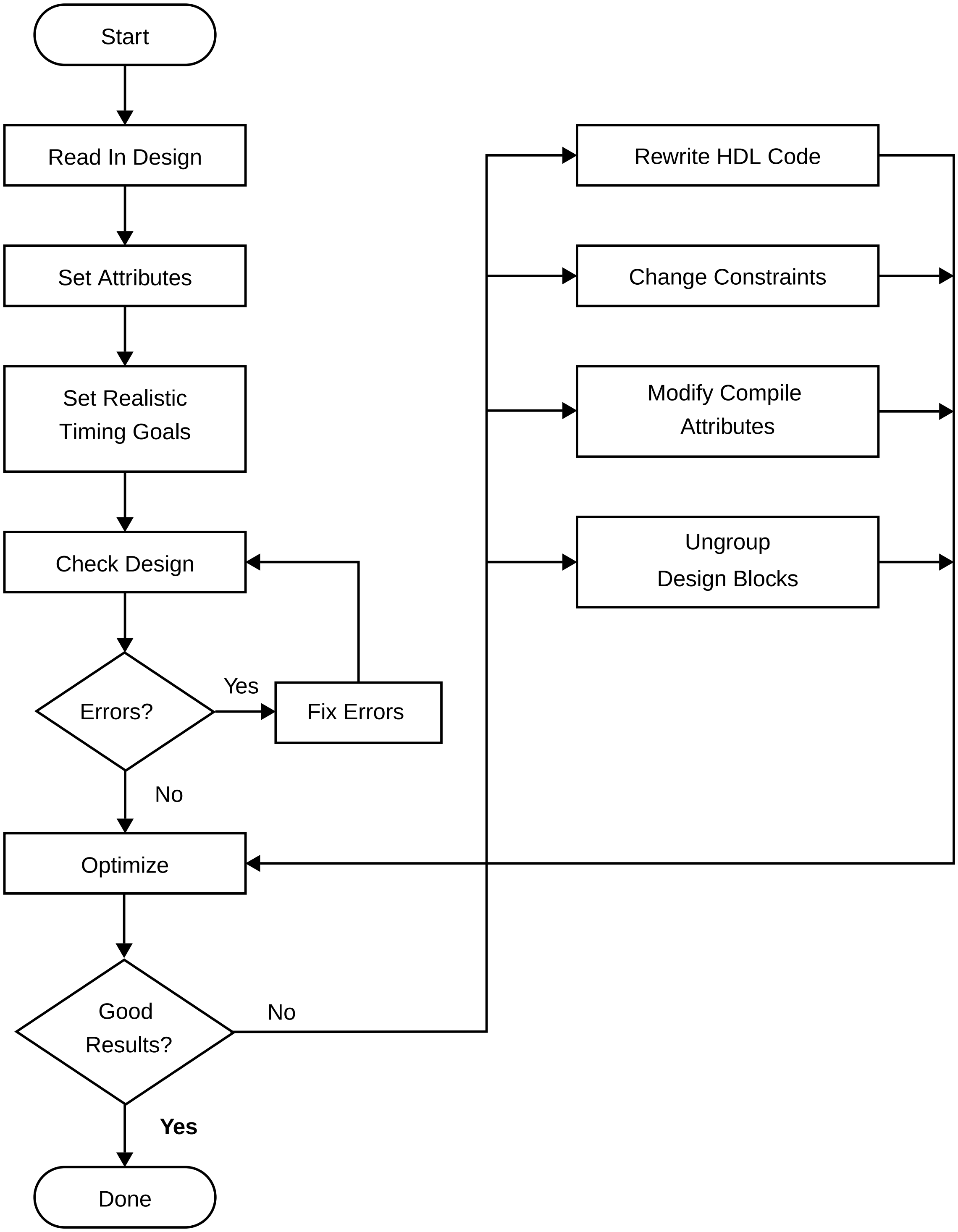
Design Compiler optimizes logic designs for speed, area, and routability. This optimization is performed for hierarchical combinational or sequential circuit design descriptions. From the goals you define for measurable circuit characteristics, Design Compiler synthesizes a circuit and puts it in a target technology. This allows you to generate schematics and netlists compatible with your computer-aided engineering (CAE) tools.

The synthesis process in Figure 1-1 follows this general scheme:

- Read in the design and its subdesigns. See Chapter 7, “Setting the Design Environment.”
- Set design attributes on the top-level design. See Chapter 7.
- Set realistic timing or area goals for the design. See Chapter 8, “Defining Optimization Goals and Setting Constraints.”
- Run check_design to verify the design. Identify and correct any errors. See Chapter 7 and Chapter 8.
- Perform Design Compiler optimization. See Chapter 7.
- Run area and constraint reports to determine whether design goals are met. See Chapter 9, “Compiling a Hierarchical Design.”

- Reoptimize after modifying attributes or constraints if goals are not met. See Chapter 9.
- Run additional reports and schematics to analyze results further. See Chapter 10, “Analyzing Design Results.”

Figure 1-1 Synthesis Process



Design Styles

Designs can be hierarchical or flat, sequential or combinational.

Input Formats

The Design Compiler products support VHDL and Verilog design entry formats for the design description. Design Compiler also supports the programmable logic array (PLA) and EDIF 2 0 0 formats. FPGA Compiler supports the Xilinx XNF format.

Output Formats

In addition to the Synopsys binary format (.db format), the Design Compiler products support VHDL, Verilog, and EDIF 2 0 0 output formats. Design Compiler also supports equation, large-scale integration (LSI), Mentor Graphics, PLA, state table, and Tegas formats. FPGA Compiler supports the Xilinx XNF format.

User Interfaces

This section gives a brief introduction to the two Design Compiler user interfaces. It includes these topics:

- Platform Requirements
- Choosing the Interface to Use
- The Design Analyzer Graphical Interface

- The dc_shell Command Line Interface

Platform Requirements

For UNIX systems, if you are using a workstation without the X Window System, you cannot use the Design Analyzer interface to run the tutorial. Viewing schematics of designs requires the X Window System. However, you can use the command-line interface to perform the tutorial procedures.

With the Windows NT operating system, you can run the tutorial by using Design Analyzer or you can run it in command-line entry mode running the dc_shell in a command prompt window.

Choosing the Interface to Use

You can use either the Design Analyzer or the dc_shell interface to perform the circuit optimization exercises presented in the tutorial. If you prefer, you can use both interfaces, moving between them, depending on which interface is most expedient or informative for a certain task.

Design Analyzer is better than dc_shell for the debugging process. You should also use it to see schematics for a design before and after the synthesis process. Otherwise, dc_shell is sufficient and easier to use.

Note:

> The dc_shell interface supports two scripting languages: dcsh mode, which uses the original Synopsys language, and dctcl mode, which uses the Tool Command Language (Tcl).

To learn to use Design Compiler, many design engineers first use the Design Analyzer graphical user interface (GUI). As they become proficient with the system, they begin to use the dc_shell commands and scripts. To fully exploit the speed and capabilities of Design Compiler, design engineers often devise strategies that use both dc_shell and Design Analyzer.

For example, a design engineer can create script files to be executed from either the dc_shell command line or the Design Analyzer Command Window (dcsh mode only). The engineer might create a script, then run the script from dc_shell repeatedly, modifying values with each cycle to optimize the design. To display schematics or write reports to the Report Output window of the Design Analyzer, the design engineer would run the script periodically from the GUI window rather than from the dc_shell command line. For additional information on the Design Compiler interfaces and how to use Design Compiler, see the *Design Compiler User Guide*.

Note:

> For the tutorial, you can save the sample design as a .db file in Design Analyzer, which preserves the constraints on the file, and then read the design into dc_shell to continue the tutorial within dc_shell.

The Design Analyzer Graphical Interface

Design Analyzer provides a menu-driven interface to most Design Compiler commands. However, some dc_shell commands are not available in Design Analyzer menus; you can enter these commands in the Design Analyzer Command Window. Chapter 2, “Design Analyzer Graphical Interface Fundamentals,” describes how to use the Command Window.

For more information on using the Design Analyzer interface of Design Compiler,

- See Chapter 2, “Design Analyzer Graphical Interface Fundamentals,” for information about Design Analyzer features
- See Chapter 3, “How to Use Design Analyzer to Optimize a Design,” for information on using Design Analyzer to optimize a design

For more detailed information on Design Analyzer features, see the *Design Analyzer Reference Manual*.

The dc_shell Command Line Interface

In the Design compiler dc_shell-based command-line interface, you can enter commands composed of command names, arguments, and variable values to perform circuit optimization tasks. There are two command modes with two corresponding scripting languages. The *dcsh mode* uses a command language developed by Synopsys, and the *Tcl mode* is based on the Tool Command Language (Tcl). The dcsh mode is the default mode.

For more information on using the dc_shell command-line interface of Design Compiler,

- See Chapter 4, “How to Use the dc_shell Command-Line Interface,” for information on using the command-line interface to perform the tutorial tasks
- See the *Design Analyzer Reference Manual* and the Synopsys man pages for detailed descriptions of the commands and menus used in the tutorial exercises

- See the *Design Compiler User Guide* for additional information about the command-line interface
- See the *Design Compiler Command-Line Interfaces Guide* for additional information on the dc_shell interface. (This guide also addresses the new tcl-based shell interface.)

Files and Directories

To perform the exercises in this tutorial, your system files and directories must be modified to enable Design Compiler and Design Analyzer to run on your system.

On UNIX systems, you can run Design Compiler in the X Window System or in dc_shell from your workstation. Your system administrator can help you modify your .cshrc file and your home directory to enable you to do this. Chapter 5, “Setting Up the Tutorial,” provides instructions.

On Windows NT systems, you run Design Compiler from a command prompt window. To do this, you must first set environment variables. For instruction on setting environment variables, see “About the Environment Variables” in Chapter 5.

For complete instructions about installing the Synopsys synthesis tools and modifying your system files and directories, see the *Installation Guide* for your system.

Script Files

A script file is a collection of dc_shell commands that you can run using either the Design Analyzer interface or the dc_shell Command-Line interface of Design Compiler.

The script files for the commands used in the tutorial exercises are provided online in the appendix_A directory and are listed in Appendix A of this manual.

Script file names in this tutorial have the .script file extension if they contain dcsh-mode commands or the .tcl file extension if they contain dctcl-mode commands.

Note:

> You cannot use Tcl scripts in Design Analyzer.

For additional information, see

- Chapter 3, “How to Use Design Analyzer to Optimize a Design,” which describes how to execute script files from within Design Analyzer
- Chapter 4, “How to Use the dc_shell Command-Line Interface,” which describes how to execute script files from the dc_shell command-line interface

Supported Operating Systems

You can install and run Design Compiler on either the UNIX or the Windows NT operating system. The two interfaces, Design Analyzer and dc_shell command-line, run on both operating systems.

Note:

> When you use the Windows NT OS version of Design Compiler, commands, arguments, and options that interact directly with the dc_shell or Design Analyzer remain in the UNIX style. However, if these commands, arguments, and options interact directly with the Windows operating system, you must use the Windows NT OS commands.

See Appendix B, "UNIX and the Windows NT OS for Synthesis Products," for a summary of the differences between using the tools under UNIX and Windows NT OS.

Preliminary Requirements

To use Design Compiler, you must be familiar with your workstation operating system and be able to

- Change directories
- Create directories
- Create alias commands
- View file contents
- Use a text editor
- Set environment variables (for UNIX users, set them in your .cshrc file, and for Windows NT users, set them by using system icons).

2

Design Analyzer Graphical Interface Fundamentals

This chapter describes the basic tasks you need to perform to run Design Analyzer under the UNIX or Windows NT operating system. It also explores the Design Analyzer views and windows, including the command-line entry window.

Fundamental tasks include how to start and quit the program and how to run it remotely. Design Analyzer views and windows are the structures within which you exercise Design Analyzer features and see results. The Command Window allows you to enter Synopsys commands (dcsh mode only) in a text field and see the results, just as you would in the dc_shell interface.

Finally, this chapter describes the Manage Licenses window, which you can use to obtain and relinquish a license to run Design Compiler and to view the list of current license holders.

If you plan to run the tutorial by using the Design Analyzer interface—which is the recommended approach for first-time users—read this chapter before you begin the tutorial exercises covered in Chapter 5, “Setting Up the Tutorial,” and the chapters that follow it.

This chapter includes the following sections:

- Starting Design Analyzer
- Quitting Design Analyzer
- Running Design Analyzer Remotely
- Using the Design Analyzer Window
- Using Design Analyzer Views
- Using the Design Analyzer Command Window
- Using the Manage Licenses Window

Starting Design Analyzer

Before launching Design Analyzer, set up the tutorial as described in Chapter 5, “Setting Up the Tutorial.”

This section includes the following:

- Starting Design Analyzer in UNIX
- Starting Design Analyzer in the Windows NT OS
- Starting Design Analyzer From Within dc_shell in a Command Prompt Window

During the tutorial, always start Design Analyzer from the tutorial directory. Starting from this directory causes the variables defined in the tutorial .synopsys_dc.setup to be initialized.

Starting Design Analyzer in UNIX

If you use the UNIX operating system, invoke Design Analyzer from a UNIX shell, either directly on your machine or in a shell created on the remote host.

To start Design Analyzer in UNIX,

1. Change to your tutorial directory.
2. Type the following command in your command window:

   ```
   % design_analyzer &
   ```

   The Design Analyzer window opens.

Starting Design Analyzer in the Windows NT OS

On a Windows NT OS machine, there are two ways to invoke Design Analyzer:

- Using the command prompt window
- Using icons

Of these methods, the first approach is recommended. This method lets you review any error messages reporting conditions that interrupt or abort invocation of Design Analyzer.

Starting Design Analyzer From Within dc_shell in a Command Prompt Window

This section first describes how to prepare to run Design Analyzer from within a dc_shell launched in a Windows NT OS command prompt window (the recommended approach). Then it explains how to run Design Analyzer.

Prerequisite Task

Before you can launch Design Analyzer from within dc_shell, modify your PATH variable to include the following directory path:

C:\synopsys\msvc50\syn\bin

You can use the Windows NT OS set command or the System Properties page to set the PATH variable.

Note:

> If your Synopsys software is installed in a directory other than C:\synopsys, specify the actual installation directory.

Starting Design Analyzer

This section describes how to start Design Analyzer, using the recommended approach.

To start Design Analyzer in Windows NT OS from within dc_shell using a command prompt window,

1. Open a command prompt window.
2. At the command prompt, enter

   ```
   c:\> dc_shell
   ```

3. At the dc_shell prompt, enter

```
dc_shell> da
```

Quitting Design Analyzer

You can quit Design Analyzer anytime. Do this in one of two ways:

Enter quit in the Design Analyzer Command Window text field.

or

Choose File > Quit.

When you exit Design Analyzer, licenses checked out for your Design Analyzer session are automatically checked back in.

Running Design Analyzer Remotely

Design Analyzer can run directly on your system if your system contains the appropriate software license. If you are using a machine that does not contain the Design Compiler software, you can run Design Analyzer remotely from your licensed machine.

To run Design Analyzer remotely,

1. Enter the name of the server that contains the Design Compiler license and software.

   ```
   % xhost +servername
   ```

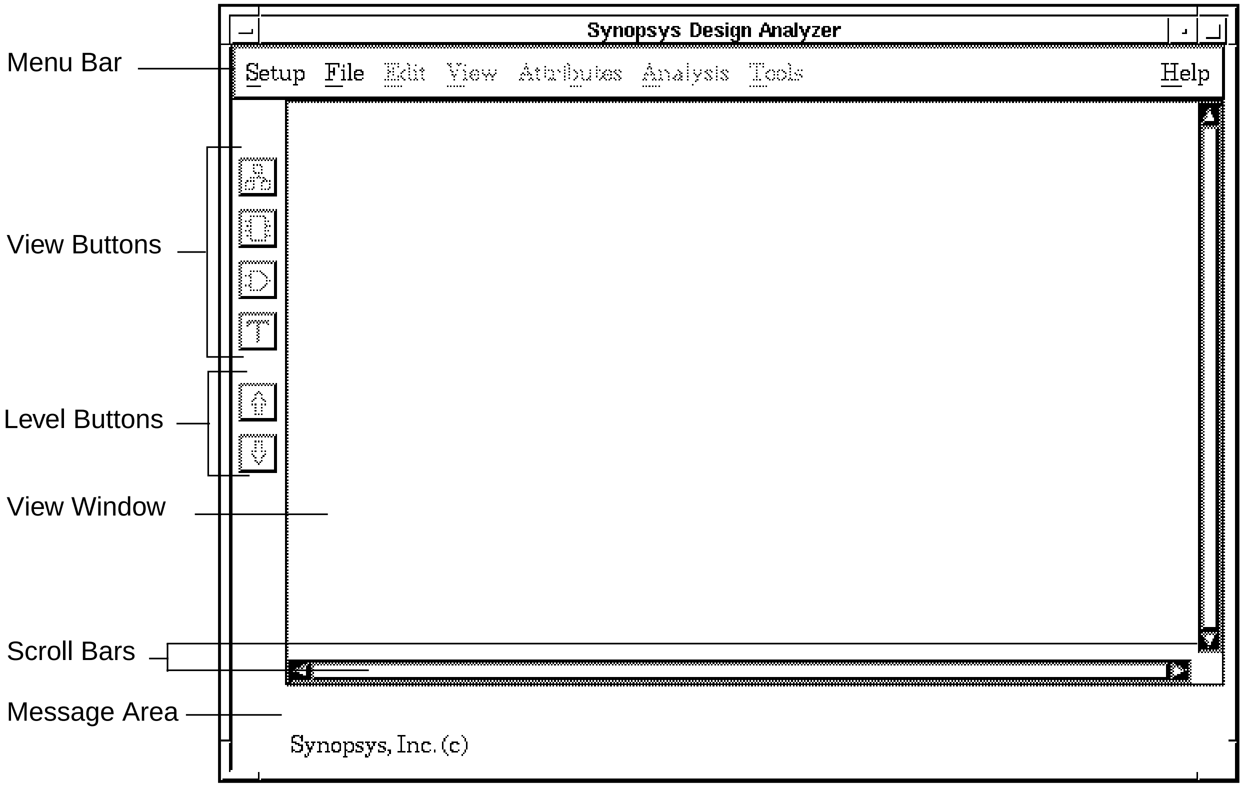
2. Start a remote shell on the machine servername. This opens and displays the results in a window on your local node hostname.

```
% rsh servername -n /usr/bin/X11/xterm -display ‘hostname‘:0 &
```

Using the Design Analyzer Window

When you start Design Analyzer, as described earlier, the Design Analyzer window appears, as shown in Figure 2-1.

Figure 2-1 Design Analyzer Window



Synopsys, Inc. (c)

The Design Analyzer window consists of the following:

Menu bar

Presents the primary menus of Design Analyzer.

View buttons

Change the way Design Analyzer depicts the selected design or instance. Design Analyzer views are shown later in this chapter.

Note:

The T button (Text view) is not used in version 3.4 and later.

Level buttons

Move you up or down through the design hierarchy, allowing you to work at different levels in the design.

View window

Presents designs, schematics of designs, and design hierarchies. The window includes scroll bars on the right and bottom sides.

Message area

Displays information about the current level of hierarchy and selected objects.

The exact appearance of the window depends on whether you are using the UNIX OS (and, if so, which X-windows window manager) or the Windows NT OS.

All primary menus are displayed in the Design Analyzer window. Some menus and menu items are not available at all times. Menus and menu items that are not available appear as dimmed text.

For example, when you first start Design Analyzer, the Edit, View, Attributes, and Tools menus appear dimmed; only the Setup and File menus are available. The dimmed menus are not available until a design is loaded into Design Analyzer.

The entries on the menu bar under UNIX and the menu bar under Windows NT OS differ somewhat. See Figure 2-2 for a sample UNIX menu bar. See Figure 2-3 for a sample Windows NT OS menu bar.

Figure 2-2 UNIX Menu Bar

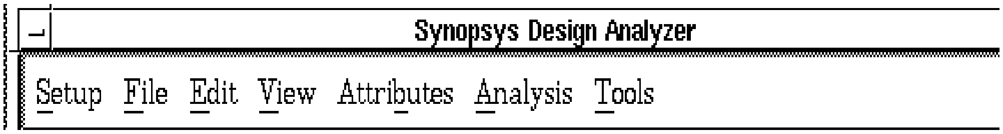
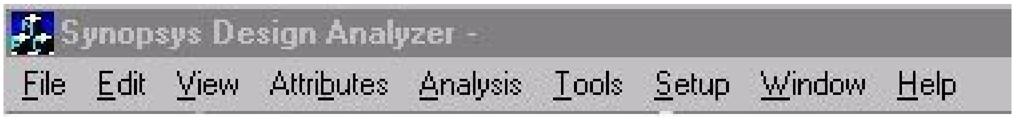


Figure 2-3 Windows NT OS Menu Bar


Synopsys Design Analyzer -
File Edit View Attributes Analysis Tools Setup Window Help

Using Design Analyzer Views

Design Analyzer can display designs in four ways, as described in these sections:

- Using Designs View
- Using Symbol View
- Using Schematic View
- Using Hierarchy View

Design Analyzer provides buttons and menu items for navigating among views and the design hierarchy. View buttons change between the three Design Analyzer views: the Schematic view, the Symbol view, and the Hierarchy view. Level buttons (up and down arrows) change levels of design hierarchy and allow access to other views from the Designs view.

Buttons perform functions that are also available as menu items. View button functions are available as menu items under View > Change View. Level button functions are available as menu items under View > Change Level.

Note:

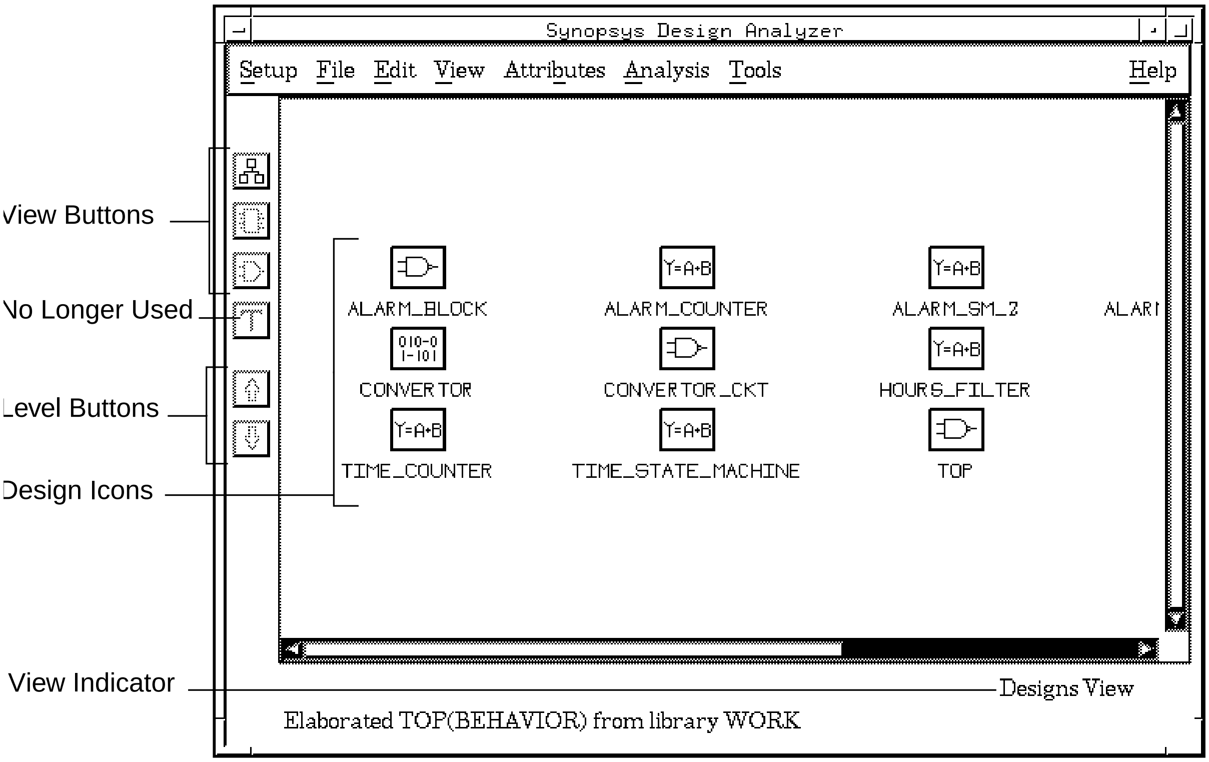
The T button (Text button) is not used in version 3.4 and later.

Each Design Analyzer view displays the type of view in the lower right corner of the window.

Using Designs View

The Designs view displays the designs and subdesigns that are in Design Analyzer memory. The Designs view, as shown in Figure 2-4, is the initial view of a design after you read the design into Design Analyzer.

Figure 2-4   Designs View



Elaborated TOP(BEHAVIOR) from library WORK

View buttons and level buttons are displayed along the left side of all Design Analyzer views. The view buttons are disabled in the Designs view. To move to other views from the Designs view, you must first push into a design.

To push into a design,

1. Click the design icon to select the design.

   The Symbol view or Schematic view appears. (The Schematic view appears if a schematic was previously generated.)

2. Choose View > Change Level.

Shortcut:

You can also push into a design by clicking the down-arrow level button after selecting the design.

To return to the Designs view from any other view,

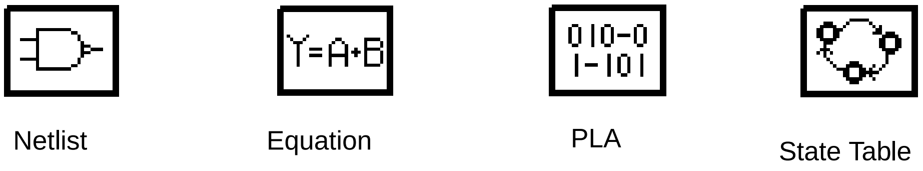
- Choose View > Change Level > Top.

  or

  Click the up-arrow level button one or more times until the Designs view displays.

The Designs view uses four types of design icons to identify design formats, as shown in Figure 2-5.

Figure 2-5 Design Icons



Netlist

Represents a design when it is read in as a netlist or when the design is optimized into gates by Design Compiler.

Equation

Represents a design in VHDL, Verilog, or equation format that is partially or completely behavioral.

PLA

Represents a design in programmable logic array (PLA) format.

State Table

Represents a design in state table format. The design files in the Alarm Clock hierarchy do not use state table format.

Using Symbol View

The Symbol view displays a design as a box with input and output ports. Use the Symbol view when setting attributes and constraints for a design and its ports. You can generate the Symbol view only after reading in the design. To display the Symbol view, you need to push into the design.

To push into a design from the Designs view,

1. Select a design.
2. Select View > Change Level > Down.

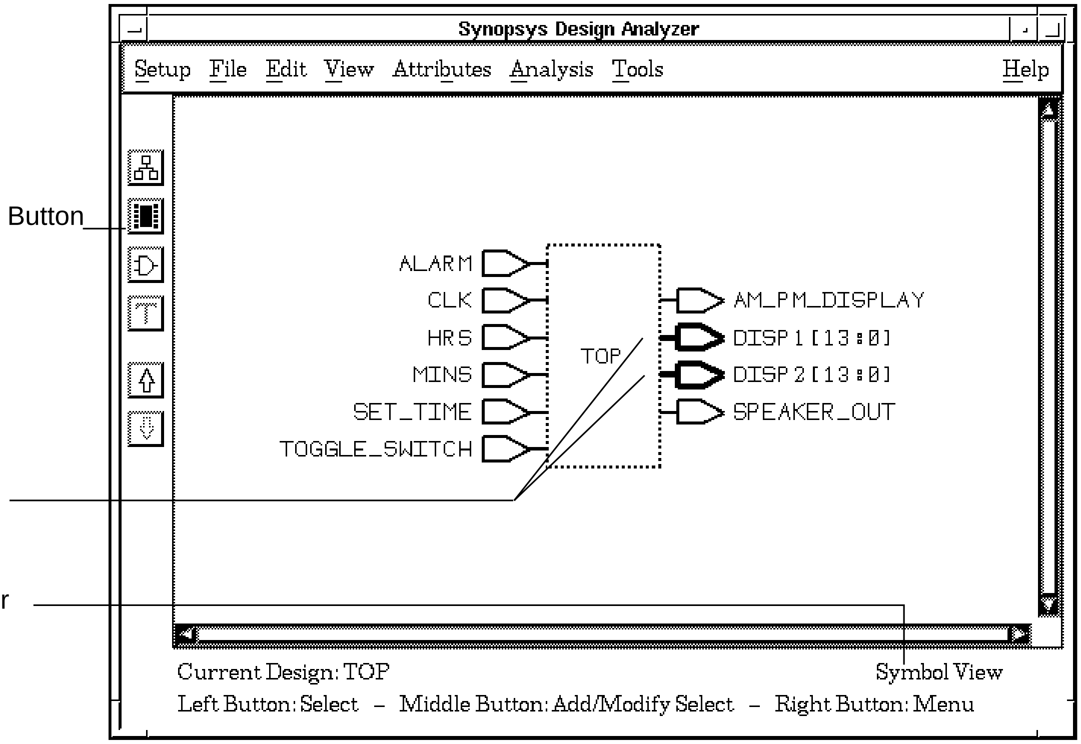
   The Symbol view for the selected design appears.

3. Click the Symbol view button if necessary.

   If a schematic exists, the Schematic view displays when you push into a design from the Designs view. In this case, click on the Symbol view button to display the Symbol view.

Ports displayed with a heavy outline are bused ports. The Symbol view button is highlighted, as shown in Figure 2-6.

Figure 2-6 Symbol View



Current Design: TOP

Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Symbol View Button

Bused Ports

View Indicator

Shortcut:

To push into a design, double-click the design.

Using Schematic View

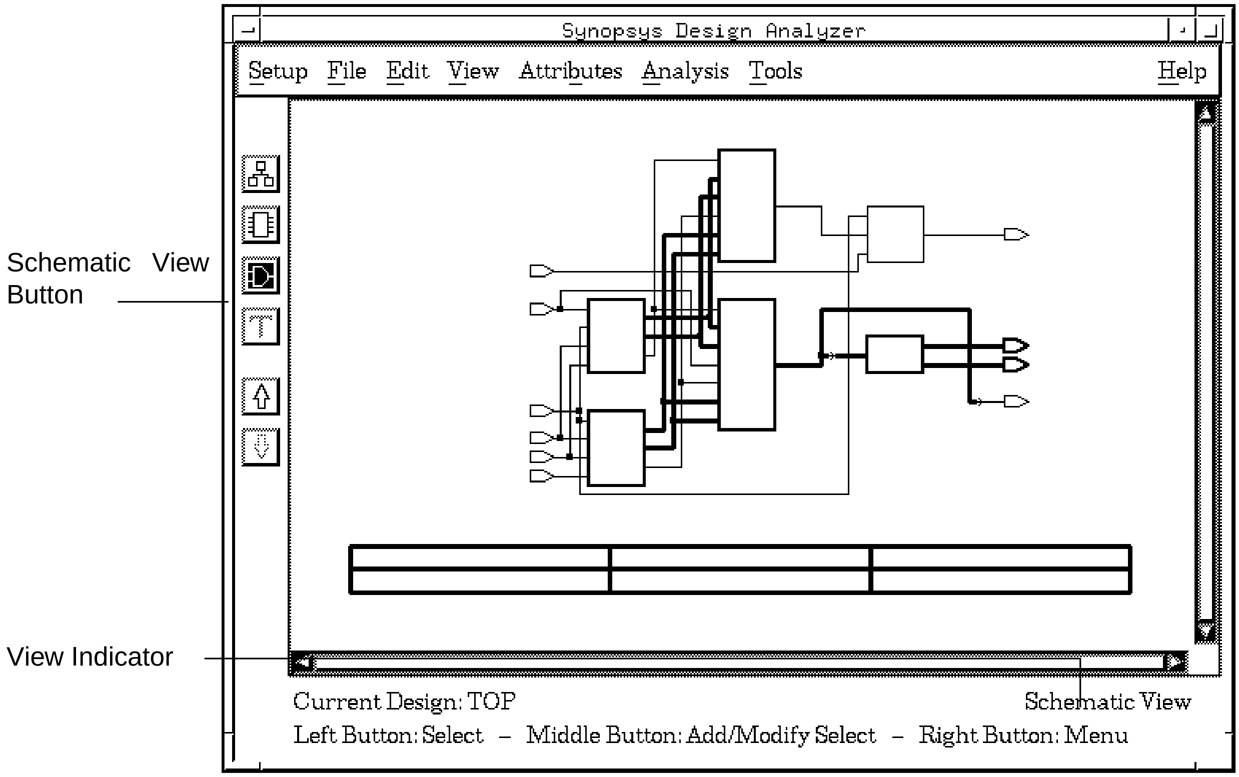
The Schematic view represents the design as instances, nets, and ports. Cells in a schematic can be subdesigns.

To generate the schematic of a design,

- choose View > Change View > Schematic.

  This generates the schematic, and the Schematic view appears. The Schematic view button is highlighted.

Figure 2-7 Schematic View



Current Design: TOP

Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

The blocks displayed in the schematic are subdesigns of TOP, the design used in the tutorial. For more information on the tutorial design, see Chapter 6, “About the Alarm Clock Design.”

Shortcut:

To move from the Symbol view to the Schematic view, double-click the design in the Symbol view.

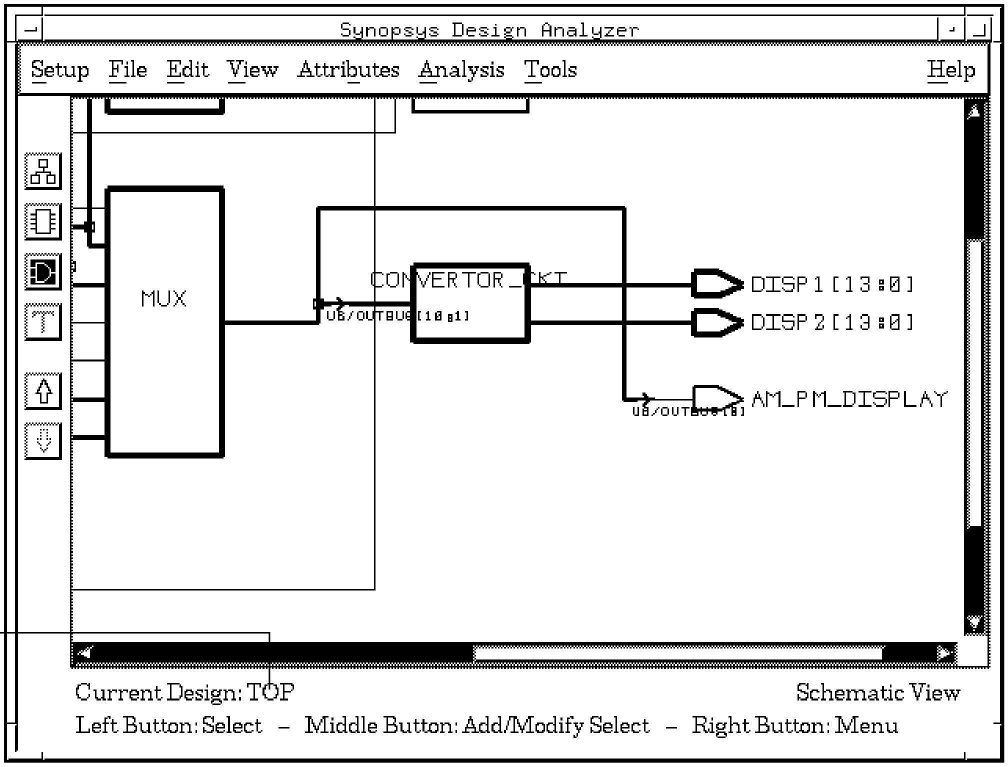
To zoom in on a portion of the schematic,

1. Place the pointer anywhere in the Design Analyzer window.
2. Press the right mouse button, and choose Zoom.

   The pointer changes to a cross hair.

3. Drag the pointer across the area you want to magnify. When you release the mouse button, the selected area fills the window. See Figure 2-8.

Figure 2-8 Zooming In on a Schematic



Current Design: TOP

Schematic View

Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Current Design Indicator

To restore the full view, choose View > Full View.

Keyboard Equivalent

Press Ctrl-f to display the full view.

Using Hierarchy View

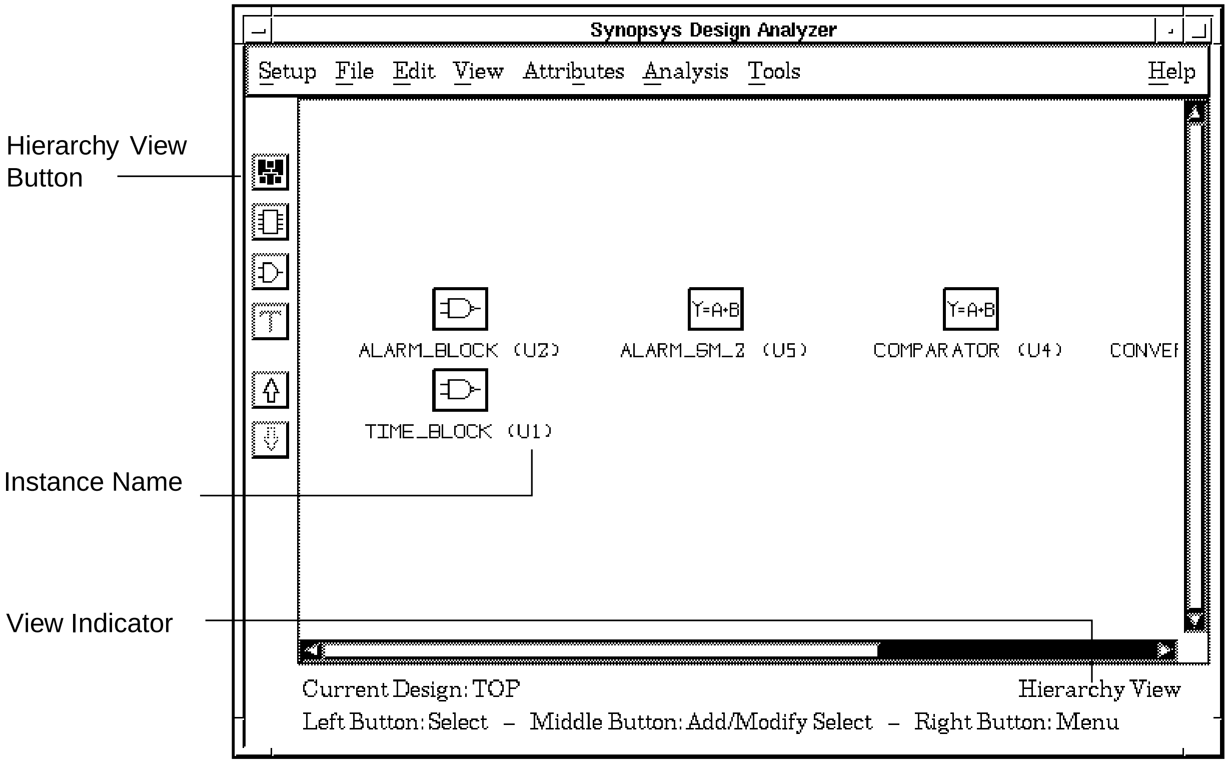
The Hierarchy view displays the subdesigns in the next lower level of hierarchy.

To display the Hierarchy view for a design,

- choose View > Change View > Hierarchy.

  The Hierarchy view appears, and the Hierarchy view button is highlighted. See Figure 2-9.

Figure 2-9 Hierarchy View



In the sample design TOP, the hierarchy level below TOP contains six subdesigns, which are displayed in the window. Scroll back and forth to see all six designs, or enlarge the window by dragging a window border or double-clicking the window's title bar.

The names in parentheses following the design names are the instance names.

Using the Design Analyzer Command Window

Use the Command Window to

- Enter dc_shell commands

  You can enter dcsh mode commands in the command-line text field at the bottom of the window.

  Note:

  Tcl mode commands are not supported in Design Analyzer.

- View information about commands that are executing during a Design Analyzer session

  The Command Window echoes commands and reports generated through Design Analyzer menus. Your Design Analyzer session also writes Command Window information to the view_command.log file.

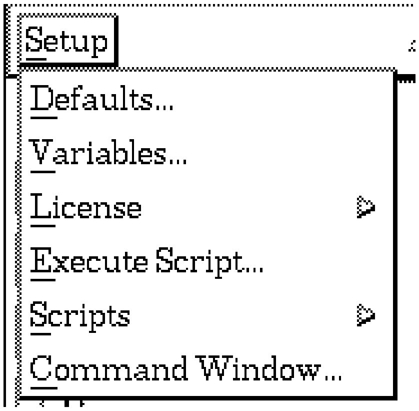
Displaying the Command Window

To display the Command window,

1. Click Setup.

   The Setup menu appears. See Figure 2-10.

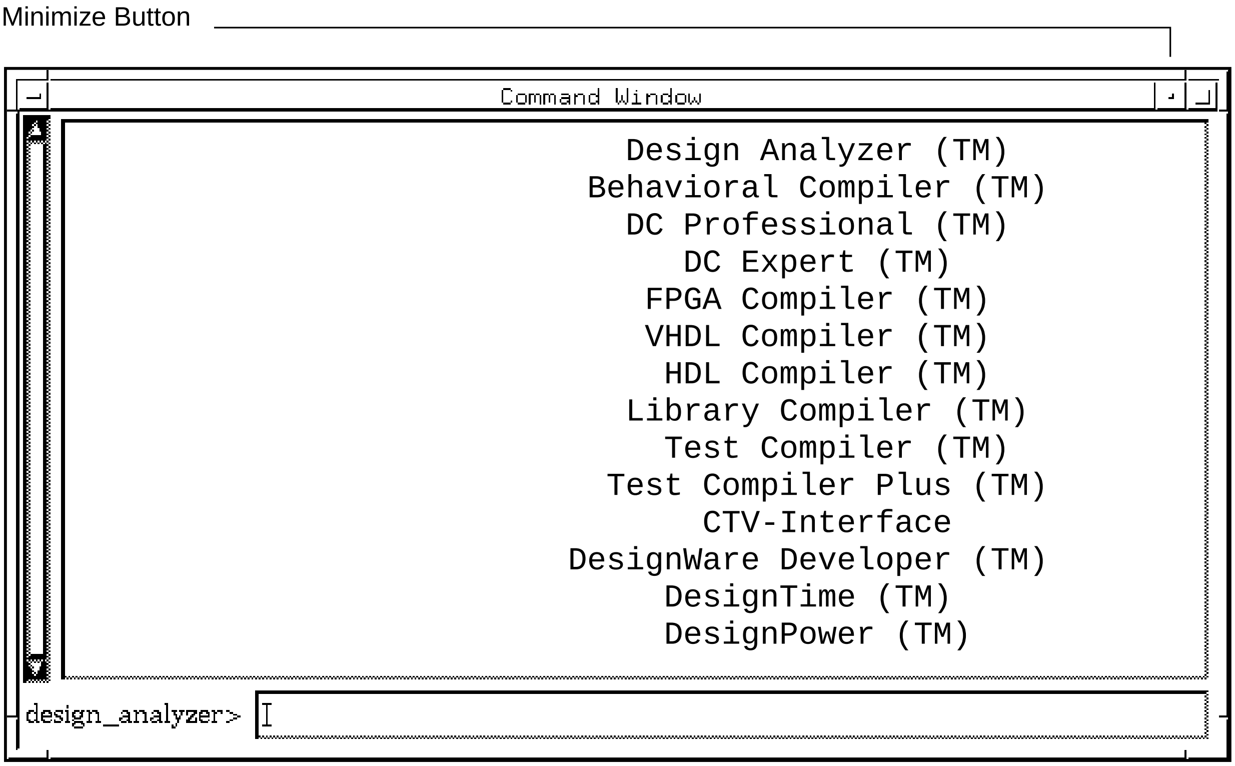
Figure 2-10 Setup Menu



2. Click Command Window.

   The Command Window opens. See Figure 2-11.

Figure 2-11   Command Window



```
Design Analyzer (TM)
Behavioral Compiler (TM)
DC Professional (TM)
DC Expert (TM)
FPGA Compiler (TM)
VHDL Compiler (TM)
HDL Compiler (TM)
Library Compiler (TM)
Test Compiler (TM)
Test Compiler Plus (TM)
CTV-Interface
DesignWare Developer (TM)
DesignTime (TM)
DesignPower (TM)
```

Design Analyzer writes the information displayed in the Command Window to the view_command.log file.

To minimize the Command Window, click the Minimize button.

Entering Command-Line Commands

The command-line text field in the Command Window functions exactly like the Design Compiler command line. You can enter any dcsh mode command supported by Design Compiler. Tcl mode commands are not supported.

For more information on running dc_shell commands, see the *Design Compiler User Guide*.

Using the Manage Licenses Window

If your Synopsys software is licensed on a network, you can use the Manage Licenses window to

- Check out and return licenses
- Display lists of licenses in use
- Display licenses available for checkout

If you know the tools and interfaces you need, you can check out the required licenses as soon as you invoke Design Analyzer. Checking out licenses early ensures that a license is available when you are ready to use it. You can use the Manage Licenses window to give up a license when you are finished with it.

If you don’t use the Manage Licenses window to check out licenses, your Design Analyzer session checks out the required licenses when they are needed during processing. Unless you release a license, your Design Analyzer session keeps any licenses until you exit the Design Analyzer session.

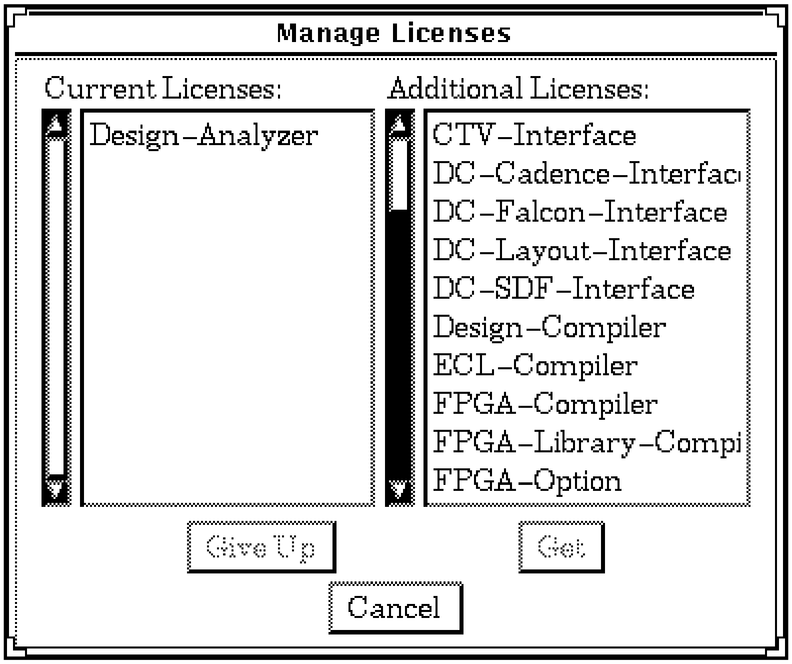
Returning Licenses

To display the Manage Licenses window,

- Choose Setup > License > Manage.

  The Manage Licenses window appears, as shown in Figure 2-12.

Figure 2-12 Manage Licenses Window



Manage Licenses

Current Licenses:

Design-Analyzer

Additional Licenses:

CTV-Interface
DC-Cadence-Interfac
DC-Falcon-Interface
DC-Layout-Interface
DC-SDF-Interface
Design-Compiler
ECL-Compiler
FPGA-Compiler
FPGA-Library-Compi
FPGA-Option

Shortcut:

> To move licenses from one column to another, use the left mouse button to double-click the license.

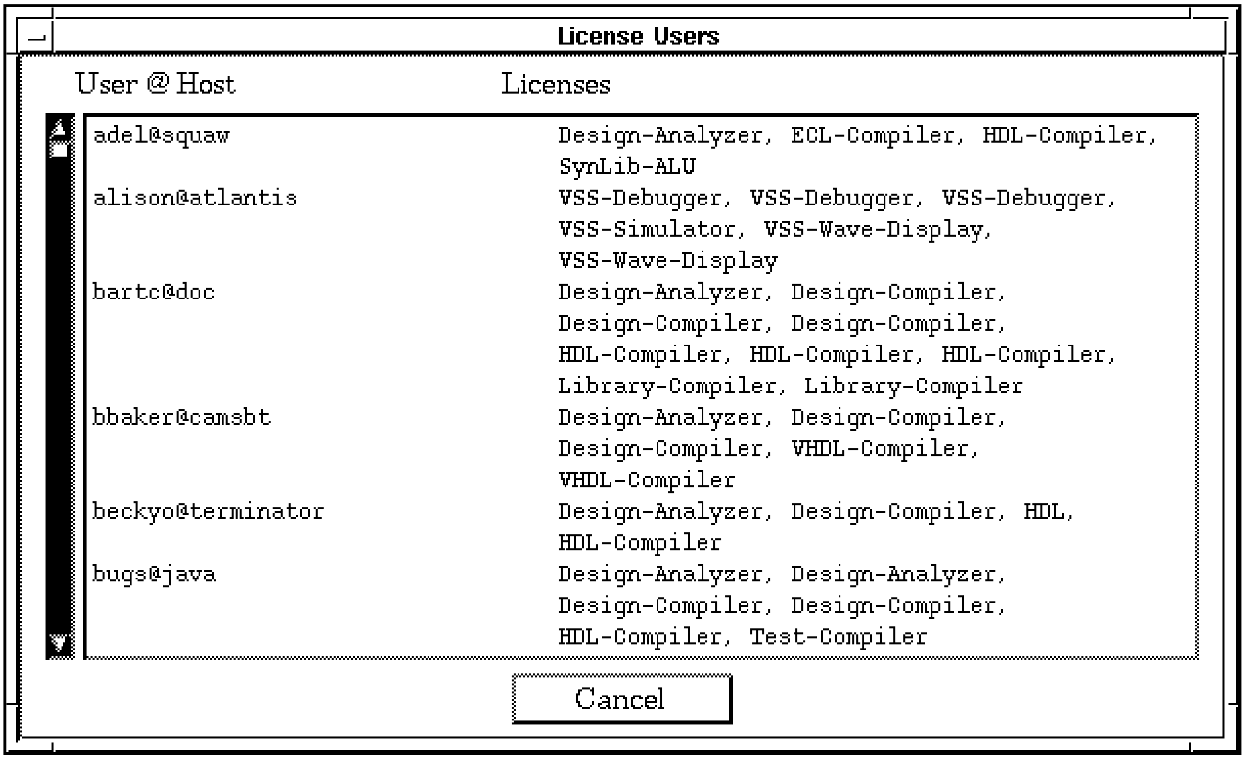
Viewing the List of License Users

Use the License Users window to display a list of users and the licenses each user has checked out.

Open the License Users window by choosing
Setup > License > Users.

The License Users window appears. See Figure 2-13.

Figure 2-13 License Users Window



User @ Host	Licenses
adel@squaw	Design-Analyzer, ECL-Compiler, HDL-Compiler, SynLib-ALU
alison@atlantis	VSS-Debugger, VSS-Debugger, VSS-Debugger, VSS-Simulator, VSS-Wave-Display, VSS-Wave-Display
bartc@doc	Design-Analyzer, Design-Compiler, Design-Compiler, Design-Compiler, HDL-Compiler, HDL-Compiler, HDL-Compiler, Library-Compiler, Library-Compiler
bbaker@camsbt	Design-Analyzer, Design-Compiler, Design-Compiler, VHDL-Compiler, VHDL-Compiler
beckyo@terminator	Design-Analyzer, Design-Compiler, HDL, HDL-Compiler
bugs@java	Design-Analyzer, Design-Analyzer, Design-Compiler, Design-Compiler, HDL-Compiler, Test-Compiler

To close the License Users window, click Cancel.

For more information about network licensing, see the *Design Compiler User Guide* and the *Licensing Installation and Administration Guide.*

3

How to Use Design Analyzer to Optimize a Design

Design Analyzer is the graphical interface to Design Compiler. This chapter introduces many of the tasks you perform with Design Analyzer to optimize a design. Beginning with Chapter 7, “Setting the Design Environment,” the tutorial guides you through these procedures using the sample Alarm Clock design.

Before you begin the tutorial for the first time, using Design Analyzer, read this chapter to gain understanding of Design Analyzer features and the Design Compiler tasks they address.

For more details on using Design Analyzer, see the *Design Analyzer Reference Manual*.

This chapter includes the following sections:

- Defining the Target Library

- Reading In a Design
- Saving a Design
- Setting Design Attributes
- Optimizing the Design
- Locating Problem Design Objects
- Generating Reports
- Running Script Files

Defining the Target Library

You define a target library by assigning a value to the target_library variable.

The target library is the technology library to which Design Compiler maps your design during optimization. The technology library embodies the ASIC technology of your final gate-level design, as well as definitions for various types of operating conditions.

Design Compiler checks the value of the target_library variable to determine which technology library to use in optimizing your design.

For the exercises in this tutorial, the target library to be used is specified by the target_library variable value set in the synopsys_dc.setup file in the tutorial directory.

To display the current value of the target_library variable in the Design Analyzer window, choose Setup > Defaults.

This displays the Defaults window. The Target Library field shows the current value of the target_library variable.

To change the target library value,

1. Enter the name of the technology library in the Target Library field.
2. Click OK to set the new value.
3. Click Cancel to close the Defaults window.

Reading In a Design

Before using Design Analyzer or Design Compiler to work on a design, read the design from disk into active memory. This procedure is called reading in a design. As the design is read in, it is translated to a binary format, called .db format. Files in .db format have the .db file extension.

Design Compiler provides these commands for reading in files:

analyze and elaborate

These two commands are always used together to read in Verilog or VHDL design files.

The analyze command reads a VHDL or Verilog file, checks for proper syntax and synthesizable logic, and stores the design in an intermediate format. Use analyze to read in each subdesign as well as the top level of the design hierarchy.

The elaborate command creates a design from the intermediate format produced by analyze. The elaborate command replaces the HDL operators in your design with synthetic operators and determines the correct bus sizes. Use elaborate on the top level and on subdesigns to which you are passing parameters.

read

Use the read command to read in files that are in formats other than HDL, such as db, pla, and other formats supported by Design Compiler. Although read supports HDL formats, it does not do the checking accomplished by analyze and elaborate.

Commands for reading in designs are available at the dc_shell prompt or on the File menu in Design Analyzer.

Note:

For additional information, see the analyze, elaborate, and read Synopsys man pages. You can view the man pages from within dc_shell.

After reading in a design, Design Analyzer represents the design as an icon in the Designs View window. The Designs View window is shown in “Using Designs View” on page 2-9.

Saving a Design

You can save (write to disk) a design at any time and to any name or supported format. Supported output formats are shown in Chapter 1, “Introduction to Design Compiler.”

Design Compiler does not automatically save designs when you exit. After you modify a design, you must save it to keep any changes made to the design. Some formats, such as VHDL and Verilog, require a license to write to disk.

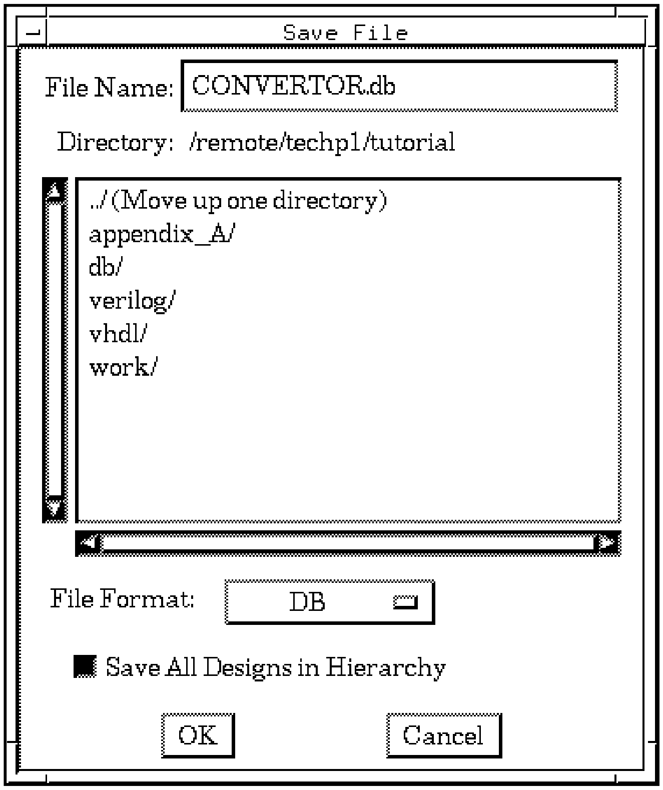
Use the write command in dc_shell to write a file to disk.

To save a file by using Design Analyzer menu options,

1. Choose File > Save As in the Design Analyzer menu bar to open the Save File window.

   Figure 3-1 shows how the Save File window looks when you run Design Analyzer in UNIX.

Figure 3-1   Design Analyzer Save File Window in UNIX



Save File

File Name: CONVERTOR.db

Directory: /remote/techp1/tutorial

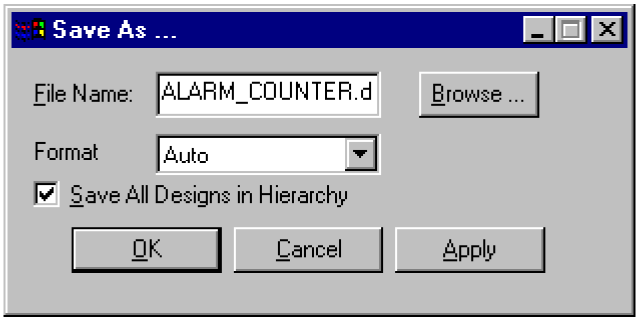
```
../ (Move up one directory)
appendix_A/
db/
verilog/
vhdl/
work/
```

File Format: DB

■ Save All Designs in Hierarchy

Figure 3-2 shows how the Save File window looks when you run Design Analyzer in Windows NT OS.

Figure 3-2   Design Analyzer Save As Dialog Box in Windows NT OS



Save As ...

File Name: ALARM_COUNTER.d

Format: Auto

☑ Save All Designs in Hierarchy

2. After choosing a file format, click OK. The window closes, and the design file is saved on the disk.

Setting Design Attributes

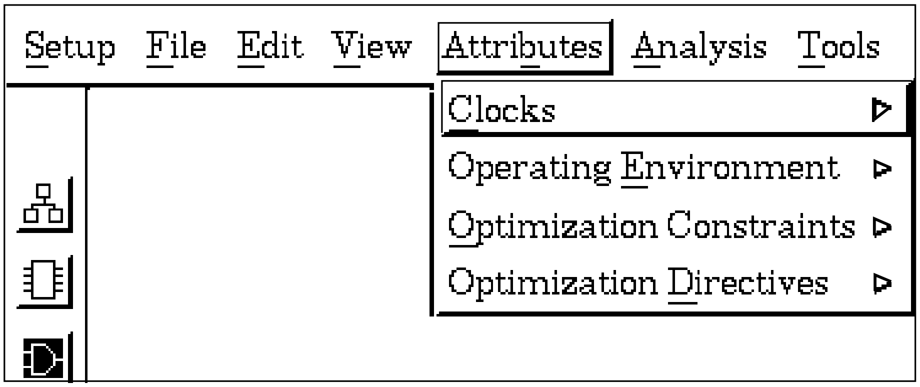
Attributes are values you set on a design that control or affect the optimization of the design. For example, attributes can specify the operating environment of the design and the constraint goals of optimization, as well as many other design parameters. See the *Design Compiler User Guide* for complete information on attributes.

Attributes are available in the Attributes menu of Design Analyzer.

The Attributes menu, shown in Figure 3-3, contains submenus and windows you can use to

- Set input and output delays
- Set drive strengths
- Set loads
- Characterize subdesigns
- Select operating conditions
- Choose a wire load model
- Create or modify a clock

Figure 3-3 Attributes Menu



For descriptions of all the menu items in the Attributes menu, refer to the *Design Analyzer User Guide* and the Design Compiler manuals.

Defining the Operating Environment

The attributes available in the Operating Environment submenu are design properties that describe the internal conditions of a design and the design's interaction with its surroundings.

After reading in a design, set attributes to determine design properties such as the drive strengths on ports, when signals arrive on ports, or the load driven by output ports.

Attributes can affect the timing of your design. For example, drive strengths of the input ports and loading of output ports are two of the values used for timing calculations. If you don't set these attributes, timing results after optimization are unrealistic.

In addition to loads and drive strengths, attributes can also specify information about the operating conditions or about which components optimization selects during gate-level implementation of your design.

Use the Symbol View and the Attributes menu to set attributes on a design's ports. For more information on the Symbol view, see Chapter 2, "Design Analyzer Graphical Interface Fundamentals."

In Chapter 7, "Setting the Design Environment," and Chapter 8, "Defining Optimization Goals and Setting Constraints," you set these attributes on a simple hierarchical design.

Setting Optimization Constraints

Fulfilling constraints represents the goal of Design Compiler optimization. For example, what is the largest delay your design can tolerate? What is the greatest area you can accept in the gate-level implementation of your design? The answers to these questions are the constraints you set for your optimization session.

Design Analyzer has two windows for setting constraints:

1. The Design Constraints window
2. The Timing Constraints window

Choose Attributes > Optimization Constraints > Design Constraints to set

- Optimization constraints

  These are your goals for area and power.

- Design rules

  These are determined by your technology library.

- Test constraints

  You can specify minimum fault coverage and other options.

Choose Attributes > Optimization Constraints > Timing Constraints to set design goals for the timing of your circuit.

Optimizing the Design

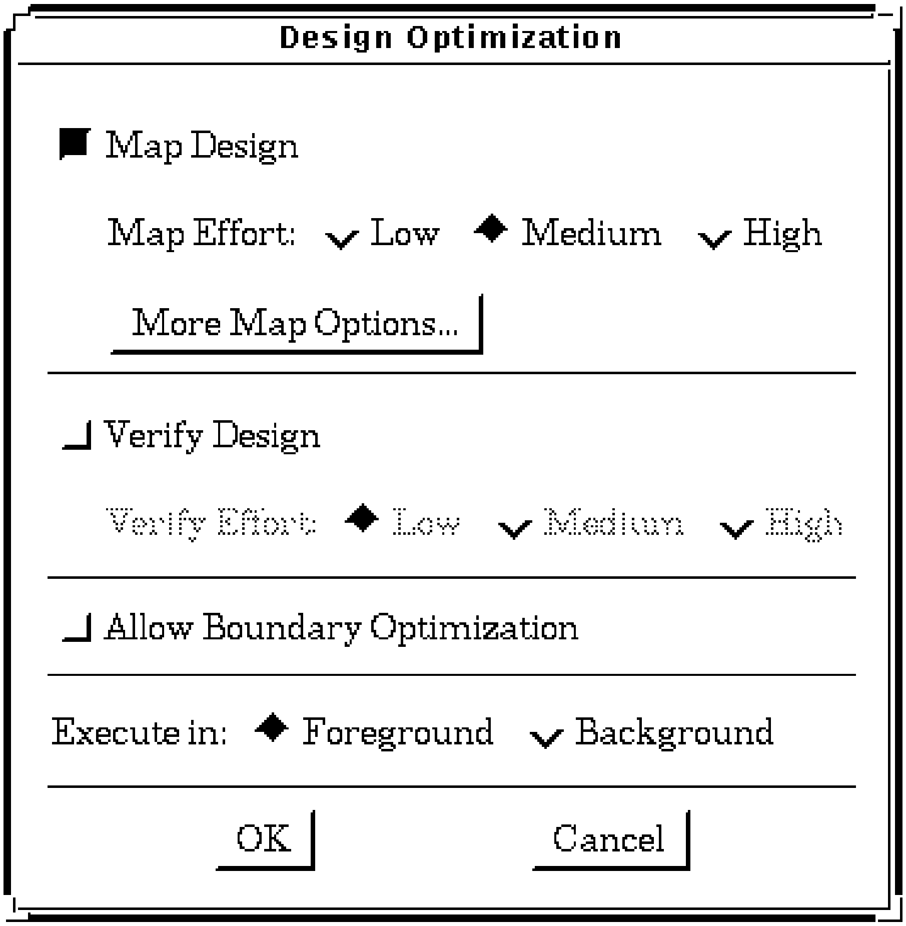
After setting the attributes you want on the design, you can optimize your design.

To start Design Compiler optimization on a selected design,

1. Choose Tools > Design Optimization in the Design Analyzer menu bar.

   The Design Optimization window, shown in Figure 3-4, opens.

Figure 3-4 Design Optimization Window



☑ Map Design

Map Effort: ○ Low ● Medium ○ High

□ Verify Design

Verify Effort: ● Low ○ Medium ○ High

□ Allow Boundary Optimization

Execute in: ● Foreground ○ Background

2. Choose the optimization options you require.

   See the *Design Compiler Reference Manual: Optimization and Timing Analysis* and the *Design Analyzer Reference Manual* for details on these options.

3. Click OK.

   The window closes, and Design Compiler optimization begins.

Locating Problem Design Objects

Before and after optimization, use the Schematic view and the check_design command to locate design objects that generate errors or warnings.

To locate design objects that generate errors or warnings, follow the steps in the next three sections:

- Generating a Schematic
- Checking the Design
- Jumping to a Design Object

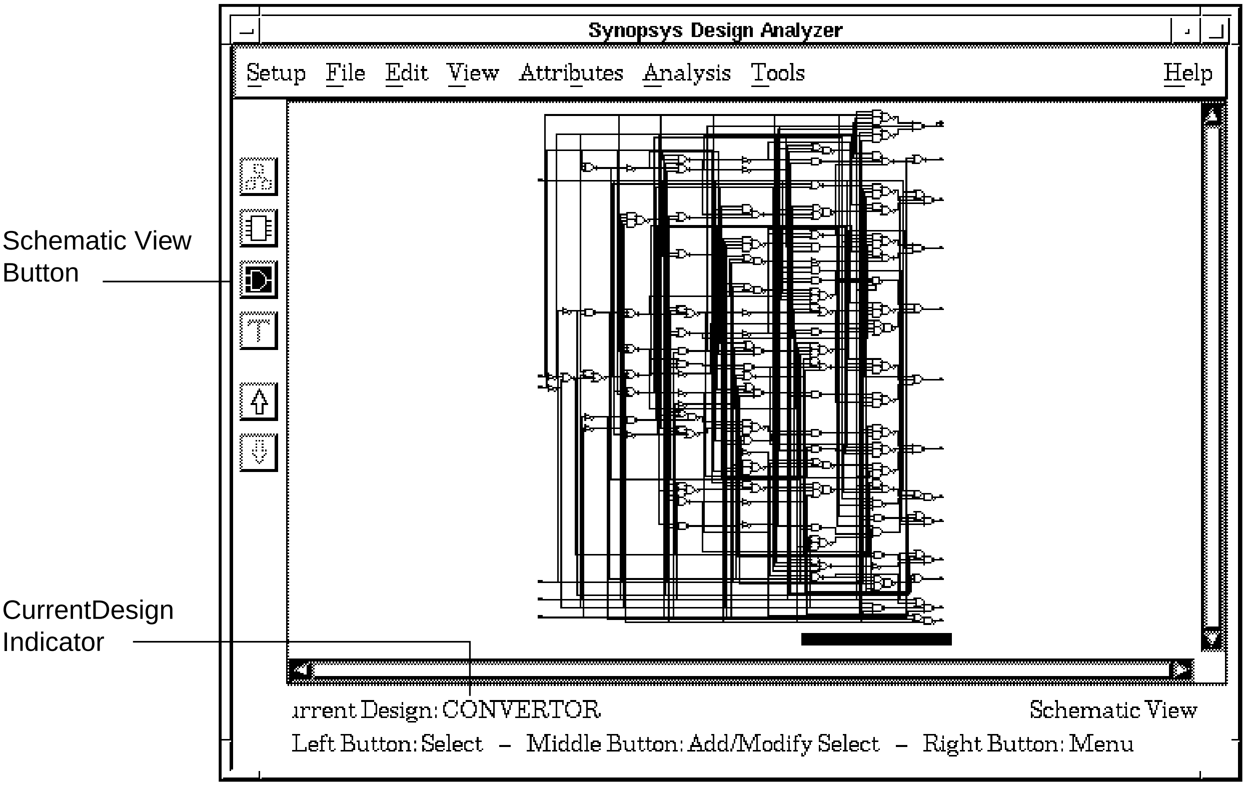
Generating a Schematic

To generate the schematic for a design,

1. Select the design in the Designs view.
2. Click the down arrow.

   Design Analyzer generates and displays the schematic, as shown in Figure 3-5.

Figure 3-5   Synopsys Design Analyzer Window Showing a Schematic


Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools Help
Schematic View Button
CurrentDesign Indicator
ırrent Design: CONVERTOR
Schematic View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Checking the Design

Run the check_design command to determine whether the synthesized design contains errors or warnings.

To run check_design on a design,

1. Choose Analysis > Check Design.

   The Check Design window appears.

2. Click OK.

   The Design Errors window appears and displays warning messages similar to the following:

```
Warning: In design ‘CONVERTOR’, output port ‘A0’ is
connected directly to output port ‘D0’. (LINT-31)
```

Jumping to a Design Object

To view the area of the schematic to which a warning refers,

1. Click an error or warning message in the Design Errors window.

   The message is selected.

2. Click the Show button.

   The Schematic view selects and displays the pertinent design object.

Generating Reports

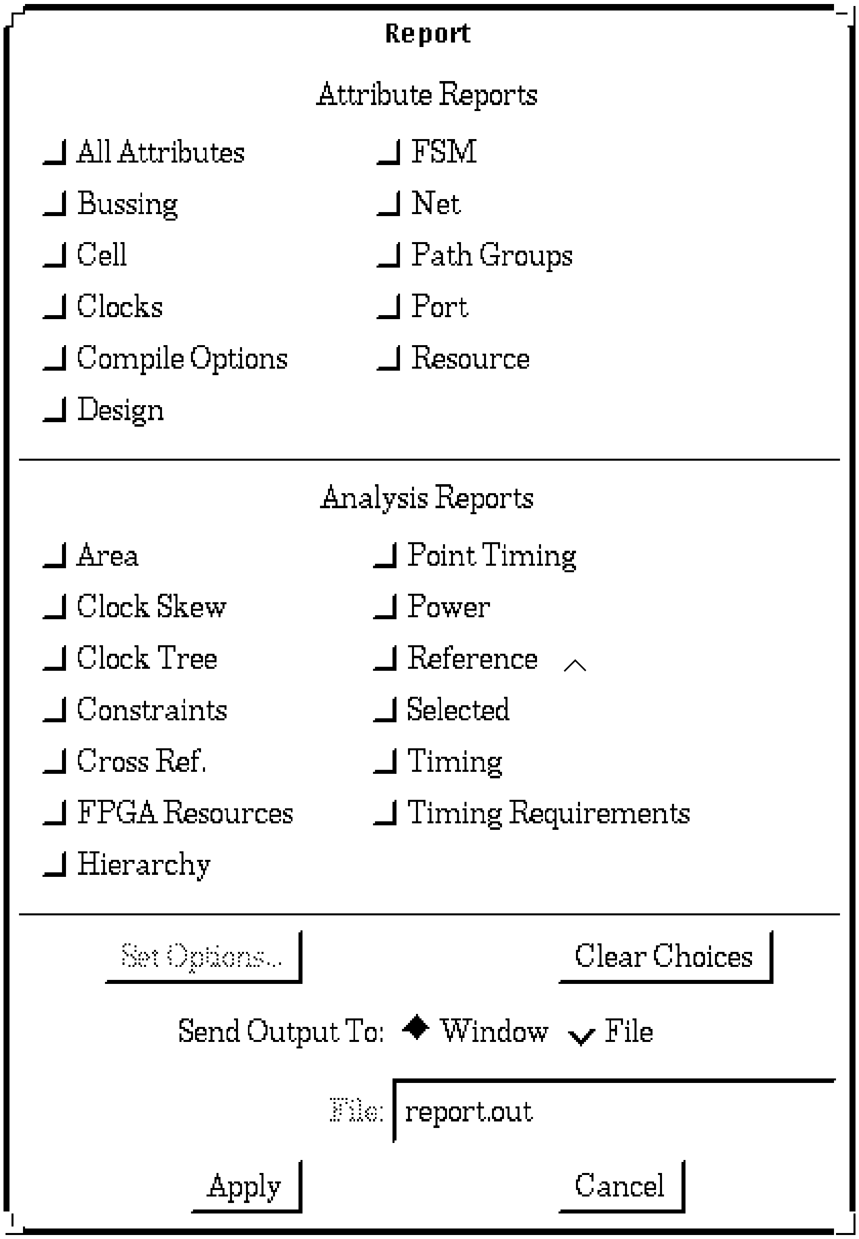
During the tutorial, you can explore some of the reporting features of Design Compiler as well as experiment with other reporting options on your own.

Design Compiler reports provide a variety of information about a design. Reports can be general or specific, providing information on a large choice of topics. The Design Analyzer Report window indicates your choices when generating reports.

You can generate reports after the design is read in as a netlist or after optimization.

To open the Report window, choose Analysis > Report in the Design Analyzer menu bar. See Figure 3-6.

Figure 3-6 Attribute Reports in the Report Window



Report

Attribute Reports

□ All Attributes
□ Bussing
□ Cell
□ Clocks
□ Compile Options
□ Design
□ FSM
□ Net
□ Path Groups
□ Port
□ Resource

Analysis Reports

□ Area
□ Clock Skew
□ Clock Tree
□ Constraints
□ Cross Ref.
□ FPGA Resources
□ Hierarchy
□ Point Timing
□ Power
□ Reference
□ Selected
□ Timing
□ Timing Requirements

Set Options... | Clear Choices

Send Output To: ◆ Window ◇ File

File: report.out

Apply | Cancel

When you click Apply, Design Compiler generates the reports you choose. The window remains open until you click Cancel.

You can direct report output to a report window or to a file. The Design Analyzer tool also writes reports to the view_command.log.

For detailed information on Design Compiler reports, see the Design Compiler manuals.

Running Script Files

A script file is a collection of Synopsys commands. The script files for the commands used in the tutorial exercises are provided in the appendix_A directory and are listed in Appendix A, “Tutorial Script Files,” of this manual.

Each tutorial exercise has two script files associated with it. One script file contains the dcsh version of the commands, and the other script file contains the Tcl version of the commands.The dcsh-mode script file names have a .script file extension, and the dctcl-mode script file names have a .tcl file extension.

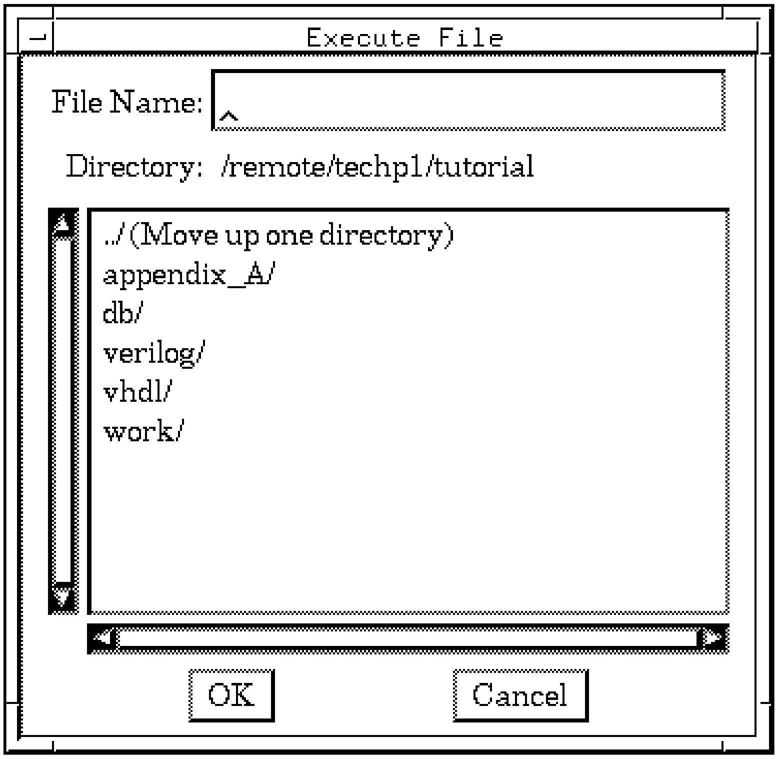
You can run only dcsh-mode script files from the Design Analyzer.

To execute a script file in Design Analyzer,

1. Choose Setup > Execute Script.

   The Execute File window, shown in Figure 3-7, appears.

Figure 3-7   Execute File Window



2. Enter the script file name in the File Name field. You can move up and down the directory structure by clicking a subdirectory or by clicking “Move up one directory.”

   The tutorial script files are located in the appendix_A subdirectory.

3. Click OK.

   The script runs, and the Execute File window closes.

4

How to Use the dc_shell Command-Line Interface

This chapter describes the fundamental tasks you perform when running the Design Compiler dc_shell under the UNIX OS or the Windows NT OS.

The dc_shell commands are presented in both the original Synopsys scripting language (dcsh mode), and in the Tool command language (Tcl mode). The dcsh mode is the default mode. To run in the Tcl mode, you must use the -tcl_mode switch when you start dc_shell.

This chapter includes the following sections:

- Starting dc_shell
- Quitting dc_shell
- Running dc_shell Commands

- About dc_shell Command Output in the Windows NT OS
- Executing Operating System Supplied Commands From dc_shell
- Executing Script Files in dc_shell

Starting dc_shell

Before launching dc_shell, set up the tutorial as described in Chapter 5, “Setting Up the Tutorial.”

This section includes the following:

- Starting dc_shell in UNIX
- Starting dc_shell in the Windows NT OS

Starting dc_shell in UNIX

You can start the dc_shell in either the dcsh mode or the Tcl mode. The default mode is dcsh. Use the -tcl_mode command switch to run in the Tcl mode.

To start the dc_shell command-line interface,

- At the operating system command prompt, type

  ```
  mysystem% dc_shell
  ```

  which launches dc_shell in dcsh mode and displays the dc_shell command-line prompt

  ```
  dc_shell>
  ```

  or type

```
mysystem% dc_shell -tcl_mode
```

which launches dc_shell in Tcl mode and displays the dc_shell command-line prompt

```
dc_shell-t>
```

Starting dc_shell in the Windows NT OS

To run dc_shell in the Windows NT OS from a command prompt window, make sure the PATH statement includes the path to the Synopsys installation directory.

Note:

> This requirement applies also if you use the Design Analyzer interface and launch Design Analyzer from within dc_shell.

Prerequisite Task

If the PATH statement does not already include the Synopsys executable directory path, modify your PATH variable before you start dc_shell to include the following directory path:

c:\Synopsys\msvc50\syn\bin

You can use the Windows NT OS set command or the System Properties page to set the PATH variable.

Note:

> If your Synopsys software was installed in a directory other than C:\synopsys, specify the actual installation directory.

Starting dc_shell

To start dc_shell, using a command prompt window,

1. Open a command prompt window.
2. Type dc_shell at the command prompt,

   ```
   c:\> dc_shell
   ```

This action launches dc_shell, displaying the dc_shell command line prompt.

```
dc_shell>
```

Quitting dc_shell

To quit dc_shell, at the command line prompt, do the following:

- Type quit or exit.

  ```
  dc_shell> quit
  ```

  or

  ```
  dc_shell-t> quit
  ```

  Your operating system prompt returns.

Running dc_shell Commands

This section includes these topics:

- "Running Commands in Windows NT OS and UNIX OS," which describes some syntactical differences.

- “Running Commands in Design Analyzer,” which describes how to run dc_shell commands in the Design Analyzer command window.
- “Displaying a List of dc_shell Commands,” which identifies the Design Compiler dc_shell commands that are similar to commands used in the UNIX operating system.

Running Commands in Windows NT OS and UNIX OS

When running dc_shell commands in either the UNIX OS or the Windows NT OS, always use forward slashes, not backward slashes as you would normally do under Windows NT OS itself. The dc_shell interface is designed to recognize forward slashes when slashes are intrinsic to command syntax.

Once you launch dc_shell, command entry is the same, regardless of the underlying operating system. For minor output differences, see “About dc_shell Command Output in the Windows NT OS.”

Running Commands in Design Analyzer

In addition to running dc_shell commands from within dc_shell, you can run dc_shell commands (dcsh mode) from within the Design Analyzer command window.

Note:

> You cannot run dc_shell commands in *Tcl mode* from within Design Analyzer.

Design Analyzer provides a window-driven and menu-driven interface to a large subset of dc_shell commands. To make available all Design Compiler features, Design Analyzer includes a command window. You can enter all commands supported by dc_shell in the Design

Analyzer command window. For more information about the Design Analyzer command window, see Chapter 2, “Design Analyzer Graphical Interface Fundamentals.”

Displaying a List of dc_shell Commands

To see a list of supported commands for dc_shell,

1. Start the dc_shell command line.
2. At the dc_shell prompt, enter the following:

   ```
   dc_shell> list -commands
   ```

   or

   ```
   dc_shell-t> list_commands
   ```

Examples

The following command examples are shown with the dc_shell prompt of the dcsh mode. Except for the alias command, you can enter these commands, as shown, in dctcl mode.

```
dc_shell> cd db
```

Changes to a directory called db.

```
dc_shell> alias wv write -f verilog
```

```
dc_shell> wv CONVERTOR -output CONVERTOR.v
```

Creates a dc_shell command alias called wv. In this example, wv replaces the write -f verilog command and writes the file you specify to disk in Verilog format.

```
dc_shell> pwd
```

Shows you the path of your current directory location.

```
dc_shell> company = “Synopsys Valued Customer”
```

Sets the company variable to the value in quotes.

```
dc_shell> history n
```

Displays a list of previously executed commands in dc_shell. The list is *n* commands long.

```
dc_shell> !n
```

Runs command number n in a history list of commands. Use the history command to determine the number n.

```
dc_shell> !!
```

Runs the last command entered.

```
dc_shell> !string
```

Runs the most recent command beginning with the character string you specify.

```
dc_shell> man any_command
```

Displays the online man page for any command in dc_shell.

About dc_shell Command Output in the Windows NT OS

Once you launch dc_shell, the only difference in behavior based on the underlying operating system is in command output.

Command output for Windows NT OS shows the fully qualified path name, including the drive.

Executing Operating System Supplied Commands From dc_shell

Use the sh command to run commands that are not specifically supported by dc_shell. Use double quotation marks around the unsupported command.

For example, you can use an operating-system-supplied command to print the contents of a file named filename to the line printer.

On UNIX systems, type the following:

```
dc_shell> sh "lpr filename"
```

or

```
dc_shell-t> eval sh {lpr filename}
```

On Windows NT systems type the following:

```
dc_shell> sh "lpr -S server -P printer filename "
```

or

```
dc_shell-t> eval sh {lpr -S server -P printer filename }
```

In the above commands, *server* is your print server and *printer* is the name of the printer.

Executing Script Files in dc_shell

A script file is a collection of Synopsys commands. The script files for the commands used throughout the tutorial are provided in the appendix_A directory and are listed in Appendix A, “Tutorial Script Files.” In this tutorial, the dcsh-mode script file names have a .script file extension, and the dctcl-mode script file names have a .tcl file extension.

To execute a script file from the dc_shell command line,

- In dcsh mode, use the include command followed by the script file name; in dctcl mode, use the source command instead of the include command.

  ```
  dc_shell> include script_file
  ```

  or

  ```
  dc_shell-t> source script_file
  ```

Part II: Using Design Compiler: A Tutorial Session

5

Setting Up the Tutorial

This chapter describes the structure of the tutorial and how to set up the tutorial for both UNIX and Windows NT operating system environments .

Before you begin the tutorial exercises described in “Setting the Design Environment” in Chapter 7, you must set up the tutorial. You do this by creating your tutorial directories and then copying and modifying Design Compiler initialization files.

Before you begin the tutorial,

- To learn about the Design Analyzer views, menus, and other interface features that allow you to execute functions, see results, and navigate the GUI, read Chapter 2, “Design Analyzer Graphical Interface Fundamentals.”
- To learn about the dc_shell command-line interface, read Chapter 4, “How to Use the dc_shell Command-Line Interface.”

This chapter includes the following sections:

- How This Tutorial Is Organized
- Creating Tutorial Directories
- Tutorial Directory Contents
- Setting the Path or Alias
- Setting Environment Variables in the Windows NT OS
- Creating Setup Files

How This Tutorial Is Organized

The tutorial exercises presented in this part of the book guide you through the process of optimizing a simple hierarchical design for a digital display alarm clock. These chapters comprise Part II of the *Design Compiler Tutorial:*

- Chapter 6: About the Alarm Clock Design

  Describes the sample hierarchical design you use for the tutorial exercises.

- Chapter 7: Setting the Design Environment

  Contains exercises you perform to establish the basic design environment, which entails specifying a minimum set of attributes and constraints necessary for synthesis.

- Chapter 8: Defining Optimization Goals and Setting Constraints

Contains exercises you perform to define optimization goals and constraints before you optimize the design. During optimization of the design, Design Compiler algorithms assess how best to implement the design by using the constraints you specify.

- Chapter 9: Compiling a Hierarchical Design

  Contains exercises you perform to optimize the hierarchical design by using the attributes and constraints you set earlier.

- Chapter 10: Analyzing Design Results

  Describes how to generate and analyze reports, including how to generate bus, cell, net, compile options, and hierarchy reports; how to analyze circuits, using schematics; how to report point-to-point timing; and how to generate a sized schematic for printing.

These chapters have a similar structure, including the following parts:

- Preliminary information and tasks

  You must complete these tasks before undertaking the exercises. You perform these preliminary tasks before using either interface for the main exercises.

- How to perform the exercises, using Design Analyzer

  For each task in the set of related tasks, this section contains conceptual information followed by steps explaining how to perform the task when using Design Analyzer.

- How to perform the exercises, using dc_shell

  For each task in the set of related tasks, this section contains conceptual information followed by commands and discussion explaining how to perform the task when using dc_shell.

For each chapter, Appendix A, “Tutorial Script Files,” contains listings of script files whose commands replicate the tasks covered in that chapter. The appendix also identifies the location of the script files themselves.

Creating Tutorial Directories

This section describes the following tasks:

- Creating Tutorial Directories in UNIX
- Creating Tutorial Directories in the Windows NT OS

The tutorial files are located in the Synopsys installation directory at your site. Throughout the tutorial, the variable *usr*/*synopsys* denotes your installation root directory.

Where you read *usr/synopsys* in text, examples, and commands, substitute your own installation directory for UNIX. For Windows NT, you must substitute C:\Synopsys for *usr/synopsys*.

Creating Tutorial Directories in UNIX

The files needed for the tutorial are in the //*usr/synopsys*/doc/syn/tutorial/ directory.

To create a tutorial directory under your home directory and copy all subdirectories and files from the Synopsys tutorial directory,

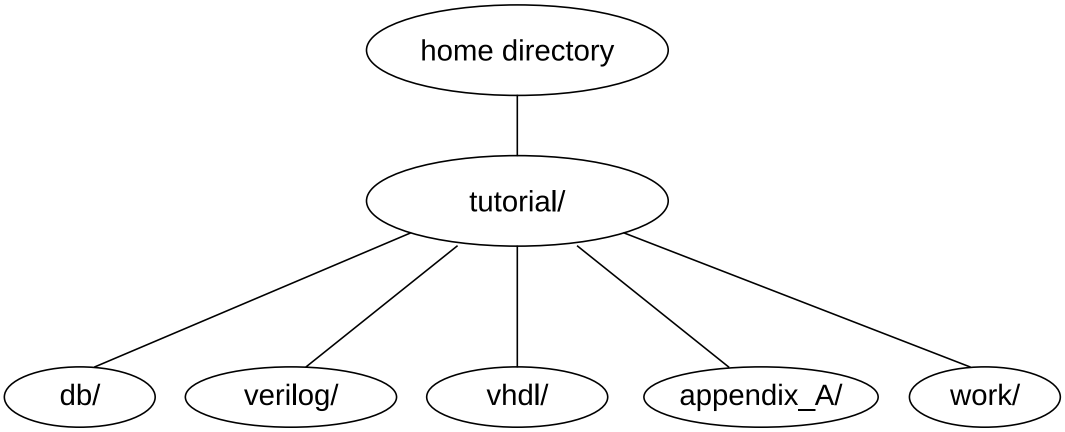
1. Change to your home directory.
2. Enter the following command:

```
% cp -r /usr/synopsys/doc/syn/tutorial
```

3. Check the contents of the tutorial directory and subdirectories against the directories and their files listed in “Tutorial Directory Contents.”

After you copy the tutorial files into your home directory, your directory structure has the structure shown in Figure 5-1.

Figure 5-1 Tutorial Directory Structure for UNIX Systems



The db, verilog, and vhdl directories contain the same designs in different formats. Any one of the three directories is sufficient for performing the tutorial exercises. If you have a Verilog license, you can use the design files in the verilog directory; however, the exercises in the documentation use the vhdl format.

Creating Tutorial Directories in the Windows NT OS

Before you create the tutorial directory structure, you must have a home directory. For instruction on creating your home folder if it does not already exist, see Appendix C, “Creating a Home Directory in the Windows NT Operating System.”

The files needed for the tutorial are in the c:\synopsys\msvc50\doc\syn\tutorial directory.

To create a tutorial directory under your home directory and copy all subdirectories and files from the tutorial directory,

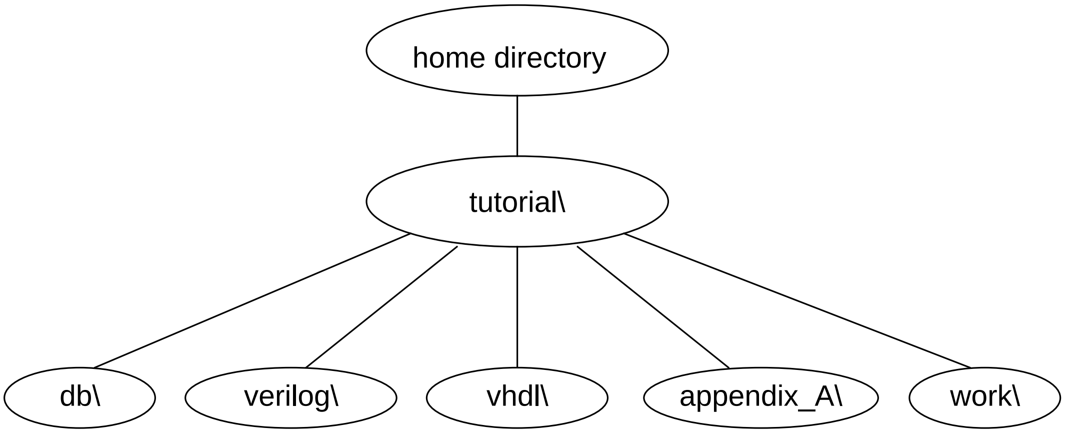
1. Change to your home directory.
2. Enter the following command:

   ```
   copy c:\synopsys\msvc50\doc\syn\tutorial
   c:\users\%username%\
   ```

3. Check the contents of the tutorial directory and subdirectories against the directories and their files listed in “Tutorial Directory Contents.”

After you copy the tutorial folders and files into your home directory, your tutorial directory hierarchy has the structure shown in Figure 5-2.

Figure 5-2 Tutorial Directory Structure for the Windows NT OS



Tutorial Directory Contents

Following are the Design Compiler tutorial directory contents. Compare these lists with your directories to verify that you have the files you need for the tutorial.

The db Directory

The db directory contains tutorial design files in Synopsys database (.db) format.

ALARM_BLOCK.db

ALARM_COUNTER.db

ALARM_SM_2.db

ALARM_STATE_MACHINE.db

COMPARATOR.db

HOURS_FILTER.db

MUX.db

TIME_BLOCK.db

TIME_COUNTER.db

TIME_STATE_MACHINE.db

CONVERTOR.pla

CONVERTOR_CKT.db

TOP.db

The verilog Directory

The verilog directory contains design files in Verilog (.v) format.

ALARM_BLOCK.v

ALARM_COUNTER.v

ALARM_SM_2.v

ALARM_STATE_MACHINE.v

COMPARATOR.v

CONVERTOR.pla

CONVERTOR_CKT.v

HOURS_FILTER.v

MUX.v

TIME_BLOCK.v

TIME_COUNTER.v

TIME_STATE_MACHINE.v

TOP.v

The vhdl Directory

The vhdl directory contains design files in VHDL (.vhd) format.

ALARM_BLOCK.vhd

ALARM_COUNTER.vhd

ALARM_SM_2.vhd

ALARM_STATE_MACHINE.vhd

COMPARATOR.vhd

CONVERTOR.pla

HOURS_FILTER.vhd

MUX.vhd

TIME_BLOCK.vhd

TIME_COUNTER.vhd

TIME_STATE_MACHINE.vhd

TOP.vhd

```
CONVERTOR_CKT.vhd
```

The appendix_A Directory

The appendix_A directory contains the script files for each tutorial exercise. Both a dcsh version and a dctcl version are provided. You can run the dcsh version of these script files from within Design Analyzer, and you can run either version from the dc_shell interface. The first column of Table 5-1 lists the script file names corresponding to the tutorial exercises in this manual. (The .script file name extension identifies the dcsh versions, and the .tcl file name extension identifies the dctcl versions). The second column of the table lists the tutorial chapter associated with each pair of script files.

Table 5-1 List of Script Files

`setenv.script & setenv.tcl`	Chapter 7
`optgoals.script & optgoals.tcl`	Chapter 8
`cmpldes.1.script & cmpldes.1.tcl`	Chapter 9
`cmpldes.2.script & cmpldes.2.tcl`	Chapter 9
`anlyzres.script & anlyzres.tcl`	Chapter 10

The work Directory

The work directory is empty at the beginning of the tutorial. Design Compiler software uses the directory to store files created by the analyze command.

Setting the Path or Alias

This section describes the following setup tasks:

- Setting the Path in UNIX
- Creating an Alias in UNIX
- Setting the Path in the Windows NT OS

To find and run Design Analyzer and Design Compiler, make sure your system points to the installation path name for your Synopsys synthesis tools.

To set a pointer to the installation path,

- Add the path name to your path environment variable

  or

  for UNIX environments, create an alias command that points to the path name.

Setting the Path in UNIX

To refer to the installation path, you add its path to your PATH environment variable. A typical UNIX path name that points to the Synopsys software is

/usr/synopsys/arch/`syn/bin/`

Substitute your installation directory for *usr/synopsys* and your machine architecture for *arch* (for example, decmips, hp300, and sparc).

This path allows you to run the design_analyzer and dc_shell commands.

To add the path name to your UNIX path variable,

1. Add this line to the PATH variable in your .cshrc file:

   ```
   /usr/synopsys/arch/syn/bin
   ```

2. Enter these UNIX commands:

   ```
   % source .cshrc
   % rehash
   ```

Creating an Alias in UNIX

As an alternative to adding the path name to your UNIX PATH variable, you can create aliases for the design_analyzer and dc_shell commands.

To create an alias that points to the path name for Design Compiler software and the Design Analyzer interface,

1. Add these lines to your .cshrc file:

```
alias design_analyzer /usr/synopsys/arch/syn/bin/design_analyzer
alias dc_shell /usr/synopsys/arch/syn/bin/dc_shell
```

   Substitute your installation directory for *usr/synopsys* and your machine architecture for *arch*.

2. Enter these UNIX commands:

   ```
   % source .cshrc
   % rehash
   ```

Setting the Path in the Windows NT OS

To find and run the Design Analyzer and dc_shell commands, your system must point to the installation path name for your Synopsys synthesis tools for the Windows NT OS.

To set a pointer to the installation path, you must add the path name to your PATH environment variable.

Based on the recommended installation directory, the Windows NT OS path name that points to Synopsys software is

```
C:\synopsys\msvc50\syn\bin
```

This path allows you to run the Design Analyzer and dc_shell commands.

To add the path name to your Windows NT OS path variable,

1. Right-click on the “My Computer” icon.
2. Choose Properties.

   The System Properties window appears.
3. Click the Environment tab.
4. Highlight PATH under System variables.
5. Add the following path name to the list of directories in the Value field for the PATH variable: c:\synopsys\msvc50\syn\bin.
6. Click the Set button in the lower right corner.
7. Click Apply.
8. Click OK.

Setting Environment Variables in the Windows NT OS

This section describes how to set the five environment variables required to run Synopsys synthesis tools for the Windows NT OS environment. In addition to the PATH variable, which you have already set, there are two categories of environment variables you need to set: system and user environment variables.

The system-wide variables required for the Synopsys synthesis tools for the Windows NT OS are

- SYNOPSYS_FILE_NAME_DELIM
- SYNOPSYS_COMSPEC
- SYNOPSYS_COMSPEC_SWITCH

The user variables required for the Synopsys synthesis tools for the Windows NT OS, whose settings affect only the currently logged-in user, are

- SYNOPSYS_KEY_FILE
- HOME

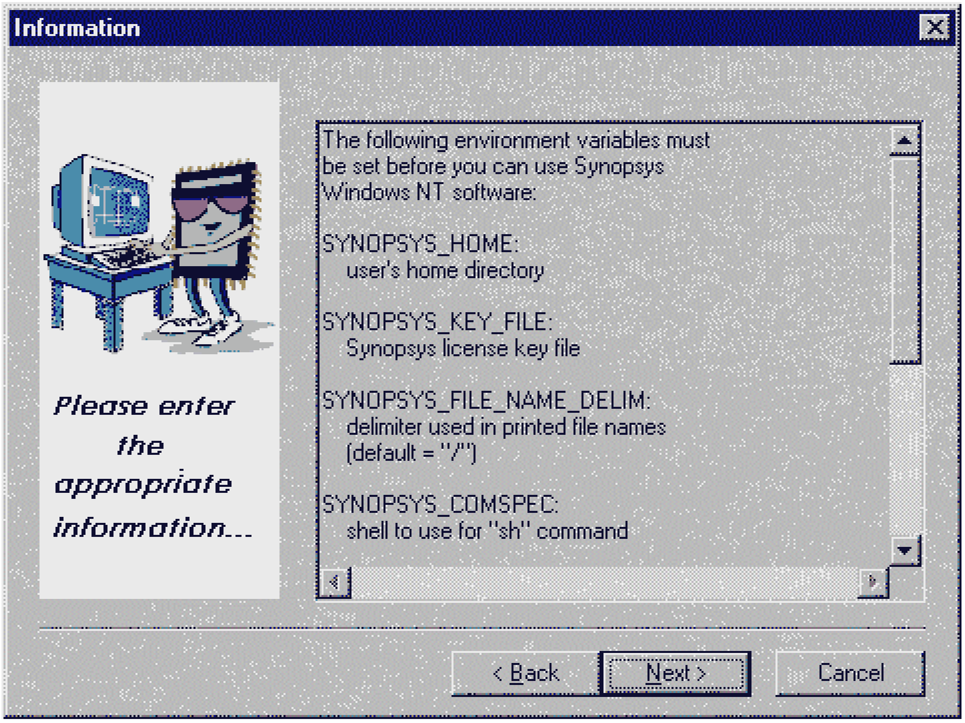
You can set these variables by using the iconic approach or the command-line approach. This section explains both methods.

About the Environment Variables

When you install Synopsys synthesis tools for the Windows NT OS environment, the first screen you see is the Software License Agreement screen.

After you click Yes to make the agreement, the installation program displays the Information screen shown in Figure 5-3.

Figure 5-3   Information Screen



Please enter the appropriate information...

The following environment variables must be set before you can use Synopsys Windows NT software:

SYNOPSYS_HOME:
    user's home directory

SYNOPSYS_KEY_FILE:
    Synopsys license key file

SYNOPSYS_FILE_NAME_DELIM:
    delimiter used in printed file names
    (default = "/")

SYNOPSYS_COMSPEC:
    shell to use for "sh" command

This screen identifies the environment variables you must set. It is important to note that the values for these variables are not automatically assigned by the installation software—you must set them.

After reading the Information screen, click Next to display the System Properties dialog box.

You use the Environment tab of the System Properties dialog box to set the environment variable values. The next section, “Using Windows Dialog Boxes to Set Variable Values,” explains how to do this.

To set the environment variables from a command window, which limits their effect to that window environment, see “Setting Environment Variables Using the Command Prompt Window” on page 5-17.

Using Windows Dialog Boxes to Set Variable Values

When you click Next in the Information screen, the System Properties dialog box appears. This dialog box has two sections: System Variables and User Variables. User variable settings pertain to the currently logged-in user.

Table 5-2 identifies the system variables and their required values you must set for the Synopsys synthesis tools for the Windows NT OS environment:

Table 5-2 System Variables for Windows NT OS

Variable	Value
SYNOPSYS_FILE_NAME_DELIM	/
SYNOPSYS_COMSPEC	c:\winnt\system32\cmd.exe
SYNOPSYS_COMSPEC_SWITCH	/c

To set the system environment variables,

1. Select the variable in the System Variables section.

   The selected variable appears in the Variable field below.

2. In the Value field, enter the value to be assigned to the selected variable.

3. Click Set.

4. Repeat steps 1 through 3 for the remaining two system environment variables.

You also use the Environment page of the System Properties dialog box to set the user environment variables.

Table 5-3 shows the user environment variables and common, sample values you set for them:

Table 5-3 User Environment Variables for Windows NT OS

Variable	**Value**
LM_LICENSE_FILE	c:\flexlm\license.dat
HOME	c:\
SYNOPSYS_HOME	c:\users\%username%

To set the user variables,

1. Select the variable in the "User Variables for username" section.

   The selected variable appears in the Variable field below.

2. In the Value field, enter the value to be assigned to the selected variable.
3. Click Set.
4. Repeat steps 1 through 3 for the other user environment variable.
5. Click OK.

Setting Environment Variables Using the Command Prompt Window

Setting an environment variable from the command window limits the effect of that variable setting to the window environment. If you want to use different sets of environment variables for different window environments, use this method of setting the variable values. To set a variable, you assign its value from the command line.

To set environment variables within the command prompt window,

1. Click the Start button.
2. Choose Programs, then choose Command Prompt from the submenu that appears.
3. At the command prompt in the Command Prompt window, enter the following command to view the current settings of the environment variables:

   ```
   c:\ > set
   ```

4. At the command prompt, enter the following commands to set the system environment variables:

   ```
   c:\ > set SYNOPSYS_FILE_NAME_DELIM=/

   c:\ > SYNOPSYS_COMSPEC=c:\winnt\system32\cmd.exe

   c:\ > SYNOPSYS_COMSPEC_SWITCH=/c
   ```

5. At the command prompt, enter the following commands to set the user environment variables:

   ```
   c:\ > SYNOPSYS_KEY_NAME_FILE=c:\flexlm\license.dat

   c:\ > HOME=c:\

   c:\ > set
   ```

Creating Setup Files

At startup, the Design Compiler searches for a series of three setup files, all of which are named .synopsys_dc.setup. From the .synopsys_dc.setup files, the software tools read initialization information, such as which libraries to use and how to tailor your graphical environment. Although they are named the same, each of these files contains different information and serves a different purpose, depending on their location.

Table 5-4 describes their contents, location, and use:

Table 5-4 Design Compiler Setup Files

File Name	**Location**	**Contents**	**Use**
.synopsys_dc.setup	Synopsys directory	System variables set by system administrator	Sets system-wide Design Compiler values
.synopsys_dc.setup	User home directory	Working environment variables: - company name - your name - background color	Sets user-specific Design Compiler values
.synopsys_dc.setup	Design directory	Minimally - search path - target library - link library - symbol library	Sets design-specific Design Compiler values

Depending on the order in which these files are read, information in one file overrides that in the previously read file. Design Compiler uses the definitions for common information from the last file it reads.

In general, the .synopsys_dc.setup files are searched and read in the following order:

1. The Synopsys root directory. This system-wide file resides in /usr/SYNOPSYS/admin/setup for UNIX. For Windows NT, this system-wide file resides in the C:\Synopsys\msvc50\admin\setup folder.
2. Your home directory.
3. The directory from which you start Design Compiler, referred to as the design directory or the user directory. This file contains project- or design-specific variables.

Therefore, settings in the design directory override settings in your home directory, which in turn override those in the Synopsys system directory.

Note:

> You cannot use operating system environment variables in a .synopsys_dc.setup file. Examples in this manual use the UNIX variable syntax for operating system environment variables. UNIX variables have a $ character as the initial character.

Allowed Combinations of Modes and Setup Files

The startup file in the Synopsys root directory is always defined in a Tcl-s scripting language. Tcl-s signifies a subset of the full Tcl scripting language, for which only the following Tcl commands are supported in setup files.

Tcl-s Subset of Supported Commands

- alias
- annotate

- define_design_lib
- define_name_rules
- getenv
- get_unix_variable
- group_variable
- if
- set
- setenv
- set_layer
- set_unix_variable
- source

The startup files in your home directory and in your current working directory can be defined in either dcsh or dctcl scripting language. However, only certain combinations of the scripting languages are allowed. In particular, the combination of dcsh in the home directory and Tcl in the current working directory is not supported.

See Table 5-5 for allowed combinations.

Table 5-5 Allowed Combinations of Modes and Setup Files

dc_shell Mode	**Synopsys Root**	**Home Directory**	**Current Working Directory**
dctcl mode	Tcl-s	Tcl	Tcl

Table 5-5 Allowed Combinations of Modes and Setup Files

dc_shell Mode	**Synopsys Root**	**Home Directory**	**Current Working Directory**
dcsh mode	Tcl-s	dcsh	dcsh
	Tcl-s	Tcl-s	dcsh
	Tcl-s	Tcl-s	Tcl-s

Note:

> In dcsh mode, the Tcl setup files in your home and local directories must have a # character in the first column of the first line.

Browsing the System-Wide .synopsys_dc.setup File

The system-wide .synopsys_dc.setup file contains the system variables defined by Synopsys and affects all Design Compiler users. Only the system administrator can modify this file.

To browse the contents of the system-wide .synopsys_dc.setup file in UNIX, enter

```
% more /usr/synopsys/admin/setup/.synopsys_dc.setup
```

To browse the contents of the system-wide .synopsys_dc.setup file in Windows NT, open the file in WordPad or another word processor. The .synopsys_dc.setup file is located in the C:\Synopsys\msv50\admin\setup installation directory. If you installed the Synopsys files in a directory other than the recommended one, substitute that directory.

Creating a .synopsys_dc.setup File in Your UNIX Home Directory

The .synopsys_dc.setup file you create in your home directory contains Design Compiler variable definitions. The variables you define in this setup file determine your Design Compiler working environment—for example, you define the colors you want your Design Analyzer GUI to display. Variable definitions in the user-defined .synopsys_dc.setup file override those in the system-wide setup file.

As you become more familiar with Design Compiler, you can continue to define variables to customize your environment.

This tutorial includes a sample file you can use to create a setup file in your home directory.

To create a .synopsys_dc.setup file for this tutorial in your home directory on UNIX,

1. Enter the following command at the operating system prompt:

```
% cp /usr/synopsys/doc/syn/tutorial/.synopsys_dc.setup1 ~/.synopsys_dc.setup1
```

   This sample file has the .setup1 file name extension to avoid confusion with setup files you might already be using.

2. Browse the contents of this setup file, change to your home directory, and enter

```
% more .synopsys_dc.setup1
```

   The contents of the file are displayed as follows:

```
company = “your_company”;
```

Specifies the company name that appears on hardcopy reports and schematics.

```
designer = “your_name”;
```

Specifies the name that appears on hardcopy reports and schematics.

```
view_background = “black”;
```

Determines the background color for the Design Analyzer window. The default is black. The images of screens in this tutorial use view_background = “white”.

3. Customize your work environment by using a text editor to define your name, company, and the background color you prefer.

   If you already have a .synopsys_dc.setup file in your home directory, include the contents of .synopsys_dc.setup1 in your .synopsys_dc.setup file.

If you do not already have a .synopsys_dc.setup file in your home directory, rename .synopsys_dc.setup1 as .synopsys_dc.setup.

To rename the setup file,

- Enter the following command at the operating system prompt:

```
% cp .synopsys_dc.setup1 .synopsys_dc.setup
```

Renaming the file with the correct setup file name allows Design Compiler to find and set the information on startup.

Creating a .synopsys_dc.setup File in Your Home Directory in Windows NT

The .synopsys_dc.setup file you create in your home directory contains Design Compiler variable definitions. The variables you define in this setup file determine your Design Compiler working environment, for example, the colors you want for Design Analyzer GUI to display. Variable definitions in the user-defined .synopsys_dc.setup file override those in the system-wide setup file.

As you become more familiar with Design Compiler, you can continue to define variables to customize your environment.

Under Windows NT OS, the Synopsys synthesis products and the tutorial use information in the .synopsys_dc.setup files to set both system and user environment variables. The same order of precedence that applies to these files when they are read in UNIX applies in Windows NT.

As in UNIX, you cannot create files beginning with the dot (.) character in Windows NT. However, this tutorial includes a sample file you can use to create a setup file in your home directory.

To create a .synopsys_dc.setup file for this tutorial in your home directory on Windows NT,

1. Open the .synopsys_dc.setup1 file in WordPad or another word processor.

   The .synopsys_dc.setup1 file is stored in your home folder. By convention, the home folder path in Windows NT is C:\users\username. Appendix C, “Creating a Home Directory in the Windows NT Operating System” describes how to create a home folder.

This sample file has the .setup1 file extension to avoid confusion with setup files you might already be using.

2. Look over the contents of this setup file within WordPad or another word processor.

   The file contains these variables and values:

   ```
   company = “your_company”;
   ```

   Specifies the company name that appears on hardcopy reports and schematics.

   ```
   designer = “your_name”;
   ```

   Specifies the name that appears on hardcopy reports and schematics.

   ```
   view_background = “black”;
   ```

   Determines the background color for the Design Analyzer window. The default is black. The images of screens in this tutorial use view_background = “white”.

3. Customize your work environment by editing the pertinent variable values.

   If you already have a .synopsys_dc.setup file in your home directory, include the contents of .synopsys_dc.setup1 in your .synopsys_dc.setup file.

4. Save the modified file as .synopsys_dc.setup1.

Renaming the file with the correct setup file name allows Design Compiler to find and set the information on startup.

Browsing the Design-Specific .synopsys_dc.setup File

It is convenient to have a working directory for each design and a design-specific .synopsys_dc.setup file in each working directory. The design-specific .synopsys_dc.setup file is read last when you start Design Analyzer or Design Compiler. The setup file in the working directory contains variables that affect the optimization of designs in the working directory only.

To use a design-specific setup file, invoke Design Compiler from the particular design directory.

For this tutorial, you don’t have to create a design-specific .synopsys_dc.setup file. The tutorial includes a setup file in the tutorial directory. If you start Design Analyzer in the tutorial directory, the setup file located there is the design-specific setup file.

Note:

> The Tcl version of the design-specific setup file differs from the dcsh version. Both versions are shown in the following procedure. If you want to run in dctcl mode and the tutorial directory does not contain the Tcl version, you can create this version of the file by duplicating the following Tcl example, using a text editor, or you can run the dc-transcript program to translate the dcsh version to the Tcl version.

To browse the contents of the setup file in the tutorial directory, do the following:

1. To change to the tutorial directory, enter

   ```
   % cd usr/synopsys/doc/syn/tutorial
   ```

2. To see the contents of the design-specific setup file for the tutorial, enter

```
% more .synopsys_dc.setup
```

The following information appears on your screen:

dcsh mode

```
search_path = . + search_path
link_library = {class.db};
target_library = {class.db};
symbol_library = {class.sdb};
define_design_lib work -path work;
```

Tcl mode

```
set search_path [concat [list] $search_path]
set link_library [list class.db]
set target_library [list class.db]
set symbol_library [list class.sdb]
define_design_lib work -path work
```

The tutorial uses libraries located in the tutorial directory. You can designate technology libraries when you optimize a design, using variables entered at the command line. However, it is convenient to place variables you commonly use into a setup file.

The .synopsys_dc.setup file in the tutorial directory sets the following variables for the tutorial:

`search_path`

Provides Design Compiler with the directories to search for unresolved design references.

If you installed the tutorial directories and files in a directory other than your home directory, you need to alter the search_path variable.

In dcsh mode, change the search path to

```
search_path = {directory} + search_path
```

In Tcl mode, change the search path to

```
set search_path [concat [list directory] $search_path]
```

`link_library`

Identifies the location of subdesigns that are referenced by the design. When a design references other design files, Design Compiler software uses link_library to locate the referenced designs. If a design's full path name is not defined by link_library, set the search_path variable to include the directory locations of any referenced designs. The tutorial uses class.db as both the link library and the target (technology) library.

`target_library`

Identifies a technology or a list of technology libraries of the components to use when you optimize a design. This tutorial uses class.db as the technology library.

`symbol_library`

Identifies symbol libraries to use for generating and viewing schematics. The value can be one or more symbol libraries. This tutorial uses class.sdb as the symbol library.

`define_design_lib`

Identifies the directory in which to store the intermediate files created by the analyze command. This tutorial uses the work directory.

6

About the Alarm Clock Design

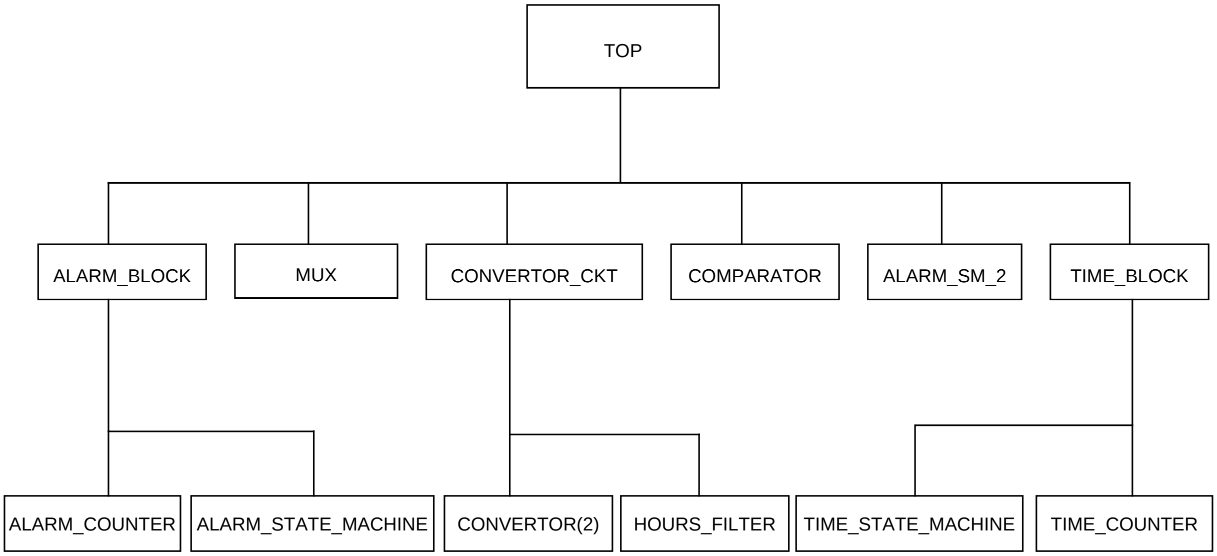
This tutorial optimizes a simple hierarchical design for a digital display alarm clock. The TOP design contains the six blocks, or subdesigns, of the alarm clock design. These subdesigns are described in the following sections:

- TOP
- ALARM_BLOCK
- TIME_BLOCK
- MUX
- COMPARATOR
- ALARM_SM_2
- CONVERTOR_CKT

These blocks are used for most of the exercises in the tutorial.

Figure 6-1 shows the hierarchy for the Alarm Clock design.

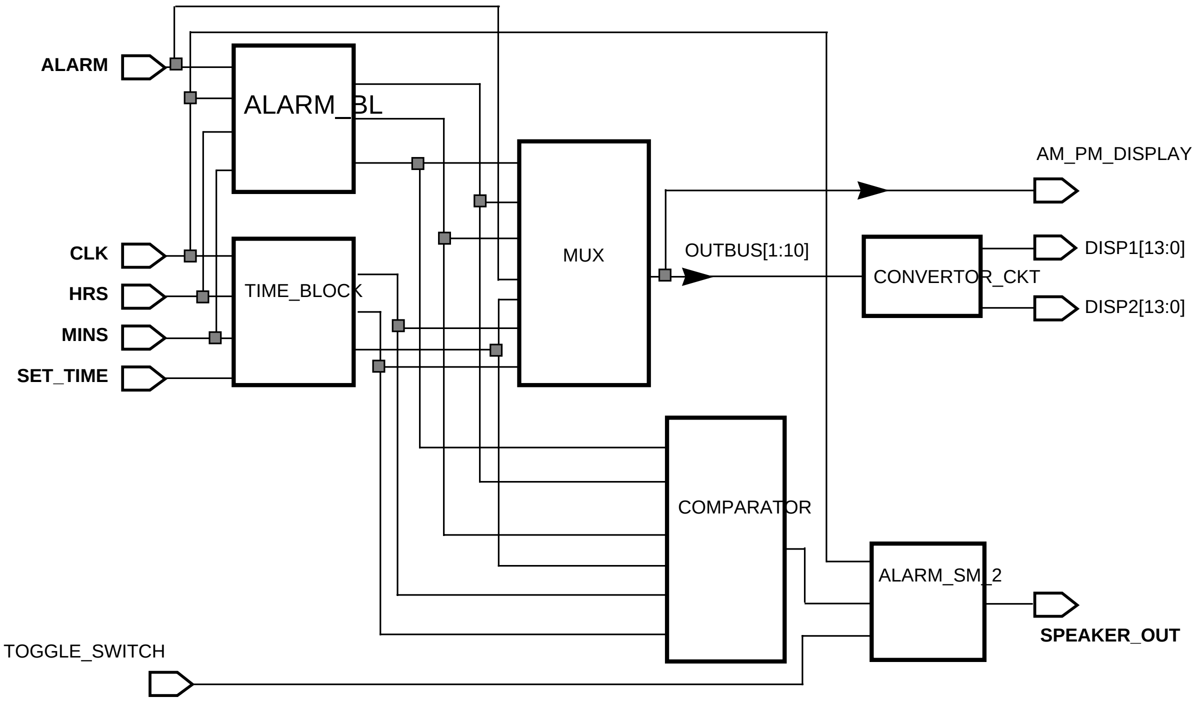
Figure 6-1 Alarm Clock Design Hierarchy



The design blocks are arranged schematically as shown in Figure 6-2. Some design blocks have a further layer of hierarchy: ALARM_BLOCK, CONVERTOR_CKT, and TIME_BLOCK contain two additional designs each.

Figure 6-2 shows the block diagram for the Alarm Clock design.

Figure 6-2 Alarm Clock Block Diagram



Following are descriptions of the blocks in the Alarm Clock design.

TOP

TOP is the top-level block of the alarm clock design. TOP contains references to all the subdesigns. Each subdesign performs a separate function of the Alarm Clock design.

ALARM_BLOCK

ALARM_BLOCK is a two-level hierarchical block. ALARM_BLOCK controls the alarm-setting function of the design.

ALARM_BLOCK has four input signals:

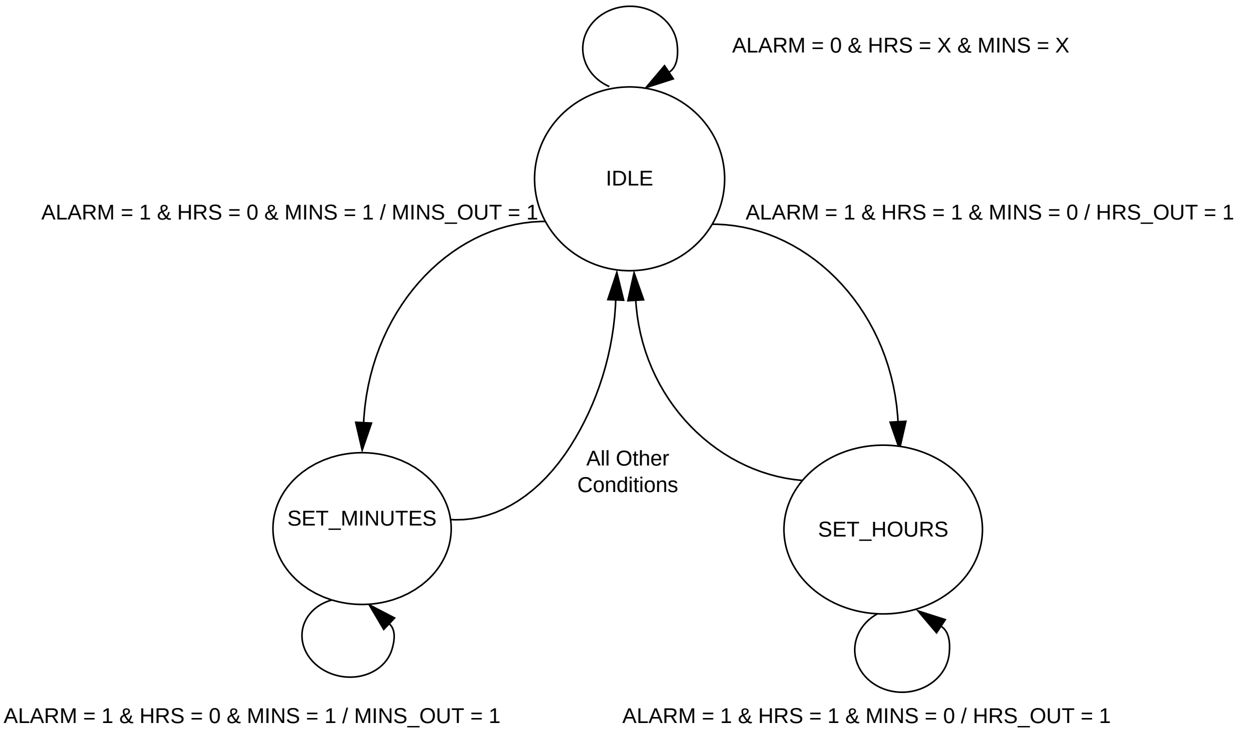
- ALARM is used with HRS or MINS to set alarm time.
- CLK is the system clock.
- HRS is used with ALARM to set alarm hours.
- MINS is used with ALARM to set alarm minutes.

ALARM_BLOCK has two output signals that are hours and minutes of the alarm time. The output signals are input signals to the MUX and COMPARATOR blocks.

ALARM_BLOCK instantiates two subdesigns:

- ALARM_COUNTER increments alarm hours and minutes and reflects AM and PM settings.
- ALARM_STATE_MACHINE sets the alarm time. It has three states, shown in Figure 6-3.

Figure 6-3 Alarm State Machine State Diagram


ALARM = 0 & HRS = X & MINS = X
IDLE
ALARM = 1 & HRS = 0 & MINS = 1 / MINS_OUT = 1
ALARM = 1 & HRS = 1 & MINS = 0 / HRS_OUT = 1
All Other Conditions
SET_MINUTES
SET_HOURS
ALARM = 1 & HRS = 0 & MINS = 1 / MINS_OUT = 1
ALARM = 1 & HRS = 1 & MINS = 0 / HRS_OUT = 1

If the block state is IDLE, it waits for a set of input signals that can change the state to SET_MINUTES or SET_HOURS.

When ALARM = 1, HRS = 0, and MINS = 1, the state changes to SET_MINUTES. From this state, a MINS_OUT pulse is fed into the ALARM_COUNTER block, which increments the minutes count. As long as the block is in the SET_MINUTES state, the minutes continue to increment in ALARM_COUNTER.

The SET_HOURS state functions the same as SET_MINUTES, except SET_HOURS is activated when ALARM = 1, HRS = 1, and MINS = 0. SET_HOURS sends an HRS_OUT pulse to ALARM_COUNTER to increment the hours.

TIME_BLOCK

TIME_BLOCK is similar to ALARM_BLOCK; however, it controls the time-of-day feature of the design. TIME_BLOCK is a two-level hierarchical block. TIME_BLOCK has four input signals:

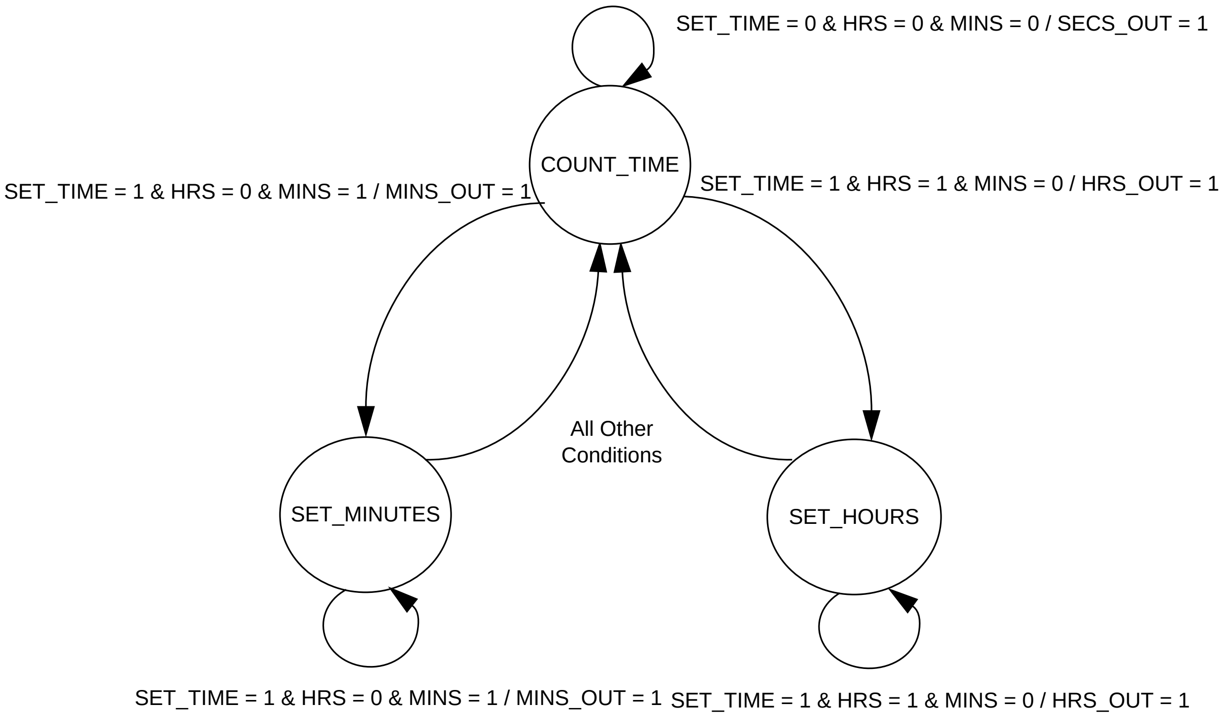
- SET_TIME is used with HRS or MINS to set the time of day.
- CLK is the system clock.
- HRS is used with SET_TIME to set time-of-day hours.
- MINS is used with SET_TIME to set time-of-day minutes.

Time-of-day hours and minutes are the two TIME_BLOCK output signals. These output signals are input signals to the MUX and COMPARATOR blocks.

TIME_BLOCK instantiates two subdesigns:

- TIME_COUNTER increments hours and minutes for the time of day and reflects AM and PM settings.
- TIME_STATE_MACHINE is used to set and keep the time of day. The state machine has three states, shown in Figure 6-4.

Figure 6-4   Time State Machine State Diagram


SET_TIME = 0 & HRS = 0 & MINS = 0 / SECS_OUT = 1
COUNT_TIME
SET_TIME = 1 & HRS = 0 & MINS = 1 / MINS_OUT = 1
SET_TIME = 1 & HRS = 1 & MINS = 0 / HRS_OUT = 1
All Other
Conditions
SET_MINUTES
SET_HOURS
SET_TIME = 1 & HRS = 0 & MINS = 1 / MINS_OUT = 1
SET_TIME = 1 & HRS = 1 & MINS = 0 / HRS_OUT = 1

When TIME_STATE_MACHINE is in the COUNT_TIME state, it outputs a pulse every second that updates the time of day. This pulse is fed into the TIME_COUNTER block. The block stays in the COUNT_TIME state until it receives a set of inputs that change the state to SET_MINUTES or SET_HOURS.

When SET_TIME = 1, HRS=0, and MINS = 1, the state changes to SET_MINUTES. From this state, a MINS_OUT pulse is fed into the TIME_COUNTER block to increment the minutes. As long as the block is in the SET_MINUTES state, the minutes continue to increment in TIME_COUNTER.

The SET_HOURS state functions the same as SET_MINUTES, except the SET_HOURS state is activated when SET_TIME = 1, HRS = 1, and MINS = 0. SET_HOURS sends an HRS_OUT pulse to TIME_COUNTER to increment the hours.

MUX

MUX determines the time setting to display. MUX enables either the time-of-day or the alarm time to be displayed.

MUX has five main input signals:

- ALARM is used with HRS or MINS to set alarm time.
- ALARM_HRS is alarm hours from ALARM_BLOCK.
- ALARM_MIN is alarm minutes from ALARM_BLOCK.
- TIME_HRS is time-of-day hours from TIME_BLOCK.
- TIME_MIN is time-of-day minutes from TIME_BLOCK.

MUX processes these input signals and feeds the information to CONVERTOR_CKT, allowing CONVERTOR_CKT to display the appropriate time. The default display is the time of day. When ALARM = 1, the alarm time is displayed.

MUX also processes and enables the AM and PM settings.

COMPARATOR

COMPARATOR compares the time-of-day to the alarm time.

COMPARATOR has four main input signals:

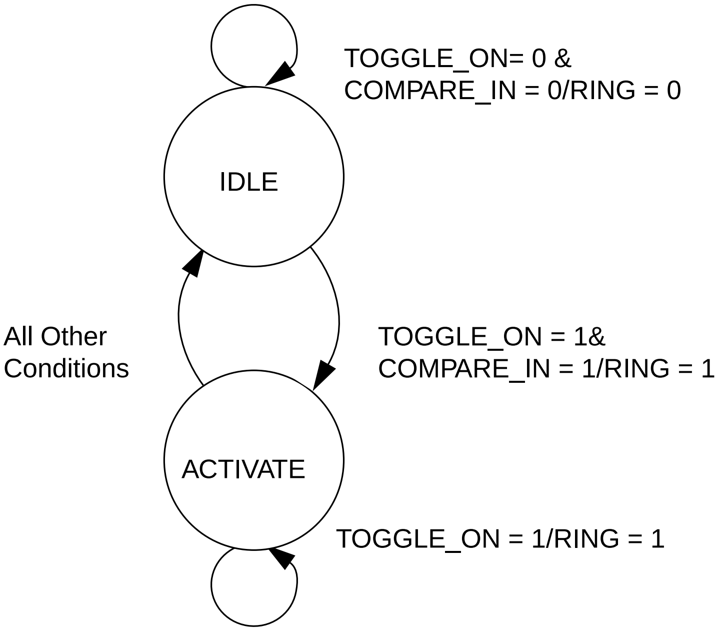
- ALARM_HRS is alarm hours from ALARM_BLOCK.
- ALARM_MIN is alarm minutes from ALARM_BLOCK.
- TIME_HRS is time-of-day hours from TIME_BLOCK.
- TIME_MIN is time-of-day minutes from TIME_BLOCK.

When the alarm, time-of-day, AM, and PM settings are equal, COMPARATOR sends a signal to the ALARM_SM_2 block.

ALARM_SM_2

ALARM_SM_2 is a state machine that has two states, IDLE and ACTIVATE, as shown in Figure 6-5.

Figure 6-5 Activate Alarm State Diagram



ALARM_SM_2 has three input signals:

- COMPARE_IN is from the COMPARATOR block; it equals 1 when time-of-day equals alarm time.
- TOGGLE_ON turns the alarm on or off.
- CLOCK is the system clock.

When the alarm time equals the time of day and TOGGLE_ON = 1 (or ON), ALARM_SM_2 goes to the ACTIVATE state. From this state, the block sends a signal to RING to enable the alarm to sound. ALARM_SM_2 remains in the ACTIVATE state as long as TOGGLE_ON is ON or until the alarm time is reset.

CONVERTOR_CKT

The CONVERTOR_CKT hierarchical block implements a binary-coded-decimal (BCD)-to-seven-segment decoder function. CONVERTOR_CKT converts the alarm time or time-of-day binary representations to signals that determine the alarm clock number display.

CONVERTOR_CKT instantiates two subdesigns:

- CONVERTOR has two instances in the CONVERTOR_CKT design. One CONVERTOR instance converts the binary representation of hours; the other CONVERTOR instance converts minutes. CONVERTOR_CKT prepares the converted information for a seven-segment light-emitting diode (LED) display.
- HOURS_FILTER disables a zero-digit display in the ten-digit column of the hours for time settings under 10:00 and over 12:59. For example, hours are filtered so that the time-of-day display for nine o’clock is 9:00 instead of 09:00.

7

Setting the Design Environment

Before you synthesize a design, you need to establish its basic design environment. The basic design environment is a minimum set of attributes and constraints necessary for synthesis.

Model the environment surrounding the design as accurately as possible. Accurate modeling of the environment results in a design that works properly when used in the system for which it is designed.

Allow less than two hours to complete the exercises in this section.

This chapter comprises the following main sections:

- Before You Begin

  Explains the preliminary decisions you must make. That is, you must choose the input format for the design example and the Design Compiler interface to use.

- Using Design Analyzer to Set the Design Environment

  Contains the complete set of tasks and exercises you perform to set the environment for the Alarm Clock design. This section begins with how to start Design Analyzer and concludes with how to save the design settings. For a list of the exercises, see the introduction to the section.

- Using dc_shell to Set the Design Environment

  Contains the complete set of tasks and exercises you perform, using the dc_shell commands (in both dcsh and dctcl modes), to set the environment for the Alarm Clock design example. This section begins with how to start dc_shell and concludes with how to save the design settings. For a list of the exercises, see the introduction to the section.

Before You Begin

Before undertaking the exercises in this chapter, you must perform the tasks described in Chapter 5, “Setting Up the Tutorial.”

These section explains the two choices you must make before you begin the tutorial:

- Choosing an Input Format
- Choosing an Interface for Design Compiler

Choosing an Input Format

Choose one input format to use throughout the tutorial. The Alarm Clock design used in this tutorial is available in these input formats:

- VHDL (file extension .vhd, located in the vhdl directory)
- Verilog (file extension .v, located in the verilog directory)

  If you use Verilog format,

  - Use the files in the verilog directory.
  - Substitute the .v file extension for .vhd in the exercises.
  - Use the VHDL analyze and elaborate procedures to read your designs.
- Synopsys database (file extension .db, located in the db directory)

  If you use Synopsys database format,

  - Use the files in the db directory.
  - Substitute the .db file extension for .vhd in the exercises.
  - Use the File >Read command in place of the analyze and elaborate commands to read your designs.

Instructions, examples, and messages throughout this chapter reflect the VHDL version of the design. Results are similar for all formats; differences in displays and messages are noted.

Choosing an Interface for Design Compiler

When you are ready to begin the tutorial, decide whether you want to use the Design Analyzer graphical interface or the dc_shell command-line interface. Using Design Analyzer is the recommended approach for first-time users. For a quick assessment of the benefits of each interface and ideas on how you might use of them in combination, see “Choosing the Interface to Use” in Chapter 1.

- If you use Design Analyzer, read Chapter 2, “Design Analyzer Graphical Interface Fundamentals,” to gain familiarity with the Design Analyzer views, menus, and windows to use during the tutorial.
- If you use the dc_shell command-line interface, read Chapter 4, “How to Use the dc_shell Command-Line Interface,” to gain an overall sense of how to use dc_shell to execute Synopsys commands, operating system commands, and script files.

When performing the exercises in the tutorial, whether you are using Design Analyzer or dc_shell, always start Design Compiler from your tutorial directory.

Using Design Analyzer to Set the Design Environment

This section contains a description of tasks and exercises in the following sections:

- Starting Design Analyzer

  Explains how to start the Design Analyzer graphical interface in UNIX and in the Windows NT OS.

- About Reading In a Hierarchical Design

  Explains what a hierarchical design is and how to view the highest level of the Alarm Clock design example, which is called TOP.

  Contains procedures that explain how to use Design Analyzer to read in the entire design, level by level, beginning with the VHDL package and continuing with the three hierarchy levels from lowest to TOP.

- Setting Attributes Using Design Analyzer

Contains tasks describing how to use Design Analyzer to set the drive strength on input ports, set the load on output ports, and set other attributes at the top level of the design.

- Saving the Design Using Design Analyzer

  Explains how to save the design environment settings using Design Analyzer.

Starting Design Analyzer

Before launching Design Analyzer, set up the tutorial as described in Chapter 5, “Setting Up the Tutorial.” Always start Design Analyzer from the tutorial directory.

On a UNIX machine, you invoke Design Analyzer from a UNIX shell, either directly on your machine or in a shell created on the remote host. From within your tutorial directory, type the following command in your command window:

```
% design_analyzer &
```

For complete instruction, see “Starting Design Analyzer in UNIX” on page 2-3.

On a Windows NT OS machine, you can use the command prompt window or icons to start Design Analyzer.

To start Design Analyzer from within dc_shell using a command prompt window,

1. Open a command prompt window.
2. At the command prompt, enter

```
c:\> dc_shell
```

3. At the dc_shell prompt, enter

```
dc_shell> da
```

For complete instructions, see “Starting Design Analyzer in the Windows NT OS” on page 2-3.

About Reading In a Hierarchical Design

This section provides conceptual information underlying the tasks you perform. Then it describes the set of exercises—each consisting of procedural steps—that you undertake to read in the Alarm Clock hierarchical design example, using Design Analyzer.

Here are the contents of this section:

- Reading In a Hierarchical Design
- Reading In the VHDL Package Using Design Analyzer
- Reading In the Lowest Hierarchy Level
- Reading In the Second Hierarchy Level
- Reading In the Top-Level Design

Reading In a Hierarchical Design

One way to make a large or complex design easier to manipulate is to divide it into smaller components. Each component performs a function of the design. The components are called subdesigns. A design that contains one or more subdesigns is a hierarchical design. Hierarchical designs can have several levels—each containing one or more subdesigns. The design that calls or references the subdesigns in the hierarchy is the top-level design. The Alarm Clock design is hierarchical; TOP is the top-level design.

The design is a VHDL netlist and references the subdesigns in the hierarchy.

Viewing the Top-Level Design File in UNIX

Under UNIX, to view the contents of the top-level design file, change to your tutorial/vhdl/ directory and enter

```
% cat TOP.vhd
```

Viewing the Top-Level Design File in the Windows NT OS

Under the Windows NT OS, to view the contents of the top-level design file, use Windows NT Explorer to change to your tutorial\vhdl directory and open the TOP.vhd file in WordPad or another word processor.

Reading In the VHDL Package Using Design Analyzer

If you are using VHDL format, you must read in the VHDL package before you read in the design files. The package is synopsys.vhd and is located in the vhdl directory.

If you are not using VHDL format, skip to the next section, "Reading In the Lowest Hierarchy Level."

To read in the VHDL package,

1. Choose File > Read.

   The Read File window appears.

2. Double-click vhdl to change to the vhdl directory.
3. Select synopsys.vhd.
4. Click OK.

   The VHDL window appears and displays the activities.

5. When the Design Analyzer prompt (design_analyzer>) appears, click Cancel in the VHDL window. (The VHDL window might be obscured by the Design Analyzer window after the reading-in process is complete.)

The following sections describe using the analyze and elaborate commands to read in VHDL designs.

Reading In the Lowest Hierarchy Level

To ensure that all references are resolved correctly (and to eliminate occurrence of unresolved references), read in the designs starting with the lowest level (third level) subdesigns and work your way up to the top-level design.

The lowest level of hierarchy contains these designs:

ALARM_COUNTER

ALARM_STATE_MACHINE

CONVERTOR (2)

HOURS_FILTER

TIME_COUNTER

TIME_STATE_MACHINE

Use the read command to read in CONVERTOR because it is in PLA format.

Analyzing the VHDL Designs

Use the analyze and elaborate commands to read in the five VHDL files. You can analyze these files in one invocation because they have the same format, but you must elaborate them individually. As you analyze designs, Design Compiler stores the resulting files in the WORK directory.

Read in the VHDL files and create their intermediate vhdl design files.

To analyze the VHDL designs,

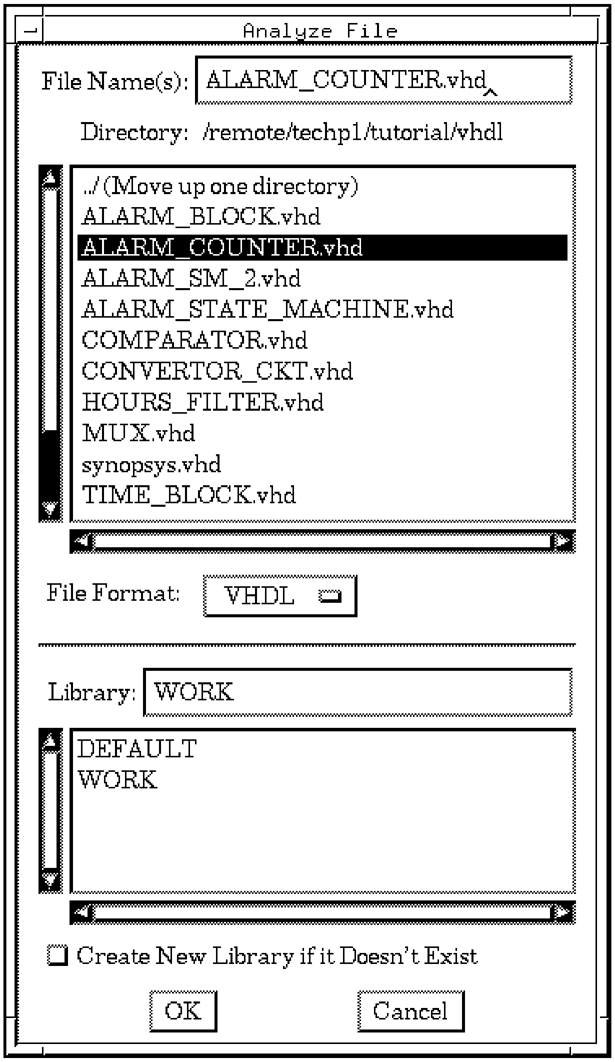
1. Choose File > Analyze.

   The Analyze File window appears. You need to analyze each VHDL design.

2. Select one design using the left mouse button. Select ALARM_COUNTER.vhd.

   When you run Design Analyzer in UNIX to analyze the ALARM_COUNTER design, the Analyze File window displays the ALARM_COUNTER.vhd file name in the File Name(s) field, as shown in Figure 7-1; it also displays the WORK directory in the Library field.

Figure 7-1 Analyze File Window in UNIX



Analyze File

File Name(s): ALARM_COUNTER.vhd

Directory: /remote/techp1/tutorial/vhdl

```
../ (Move up one directory)
ALARM_BLOCK.vhd
ALARM_COUNTER.vhd
ALARM_SM_2.vhd
ALARM_STATE_MACHINE.vhd
COMPARATOR.vhd
CONVERTOR_CKT.vhd
HOURS_FILTER.vhd
MUX.vhd
synopsys.vhd
TIME_BLOCK.vhd
```

File Format: VHDL

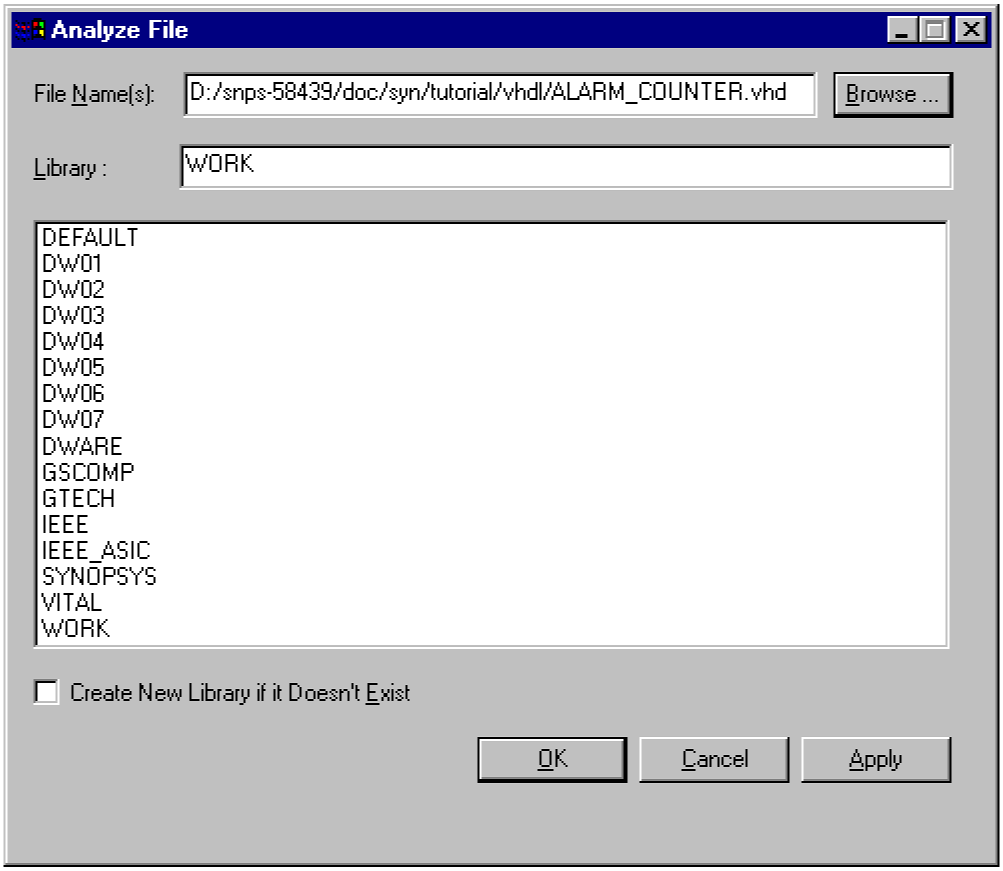
Library: WORK

```
DEFAULT
WORK
```

❑ Create New Library if it Doesn't Exist

As shown in Figure 7-2, under Windows NT OS, Design Analyzer displays the ALARM_COUNTER.vhd file name and its full path in the File Name(s) field, and displays the WORK directory in the Library field.

Figure 7-2 Analyze File Dialog Box in Windows NT OS



File Name(s): D:/snps-58439/doc/syn/tutorial/vhdl/ALARM_COUNTER.vhd

Library : WORK

DEFAULT
DW01
DW02
DW03
DW04
DW05
DW06
DW07
DWARE
GSCOMP
GTECH
IEEE
IEEE_ASIC
SYNOPSYS
VITAL
WORK

□ Create New Library if it Doesn't Exist

3. Select the remaining designs, using the middle mouse button:

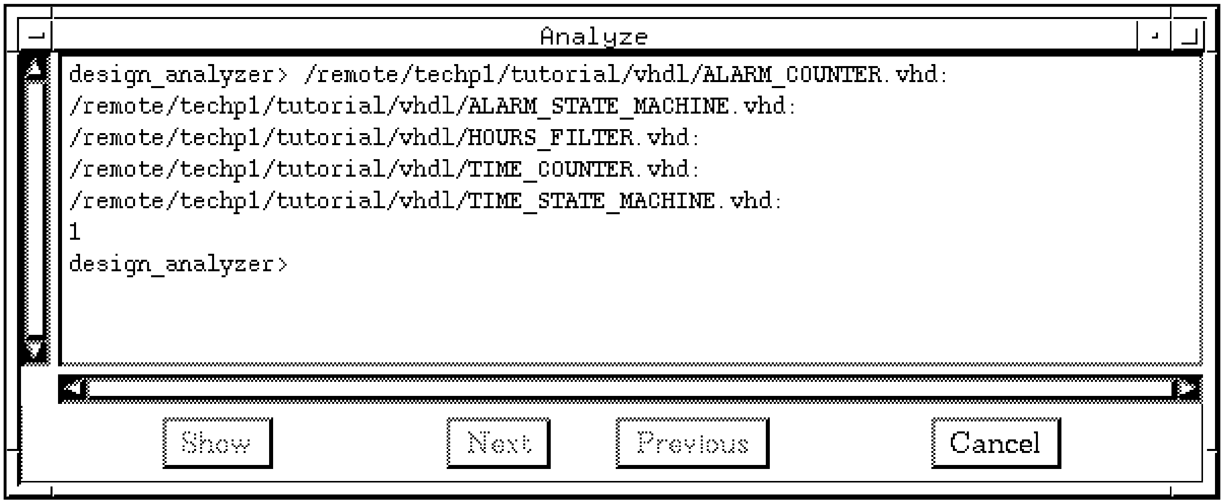
   ALARM_STATE_MACHINE.vhd
   HOURS_FILTER.vhd
   TIME_COUNTER.vhd
   TIME_STATE_MACHINE.vhd

   As you select each name, it is added to any file names in the File Name(s) field. Only one file at a time is highlighted in the list; however, the selected file names concatenate in the File Name(s) field.

4. Click OK.

   The Analyze window, shown in Figure 7-3, appears and displays the activities of the analyze command, stores the intermediate files in the work directory.

Figure 7-3 Analyze Window



```
design_analyzer> /remote/techp1/tutorial/vhdl/ALARM_COUNTER.vhd:
/remote/techp1/tutorial/vhdl/ALARM_STATE_MACHINE.vhd:
/remote/techp1/tutorial/vhdl/HOURS_FILTER.vhd:
/remote/techp1/tutorial/vhdl/TIME_COUNTER.vhd:
/remote/techp1/tutorial/vhdl/TIME_STATE_MACHINE.vhd:
1
design_analyzer>
```

5. Click Cancel to close the Analyze window.

Elaborating the VHDL Designs

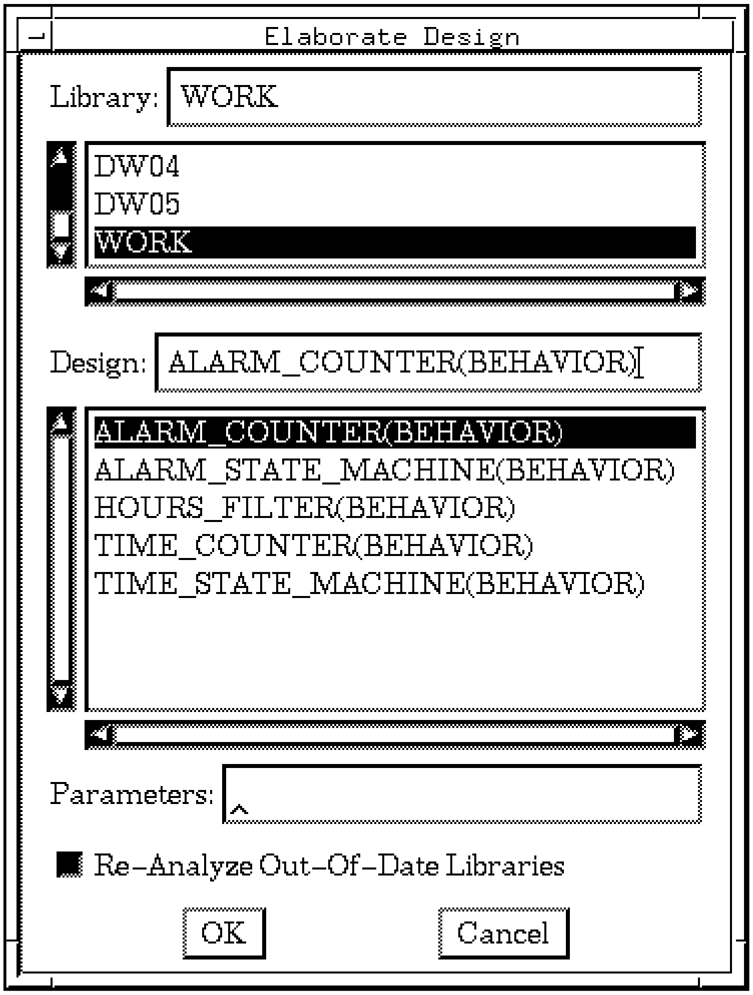
Elaborate the five VHDL designs individually. The elaborate command translates the intermediate design files created by analyze into .db format.

To elaborate each VHDL design,

1. Choose File > Elaborate.

   The Elaborate Design window, shown in Figure 7-4, appears after a short time.

Figure 7-4 Elaborate Design Window



Library: WORK

DW04
DW05
WORK

Design: ALARM_COUNTER(BEHAVIOR)

ALARM_COUNTER(BEHAVIOR)
ALARM_STATE_MACHINE(BEHAVIOR)
HOURS_FILTER(BEHAVIOR)
TIME_COUNTER(BEHAVIOR)
TIME_STATE_MACHINE(BEHAVIOR)

Parameters:

■ Re–Analyze Out–Of–Date Libraries

2. Scroll the Library list, and select WORK.

   The Elaborate Design window displays the contents of the WORK directory.

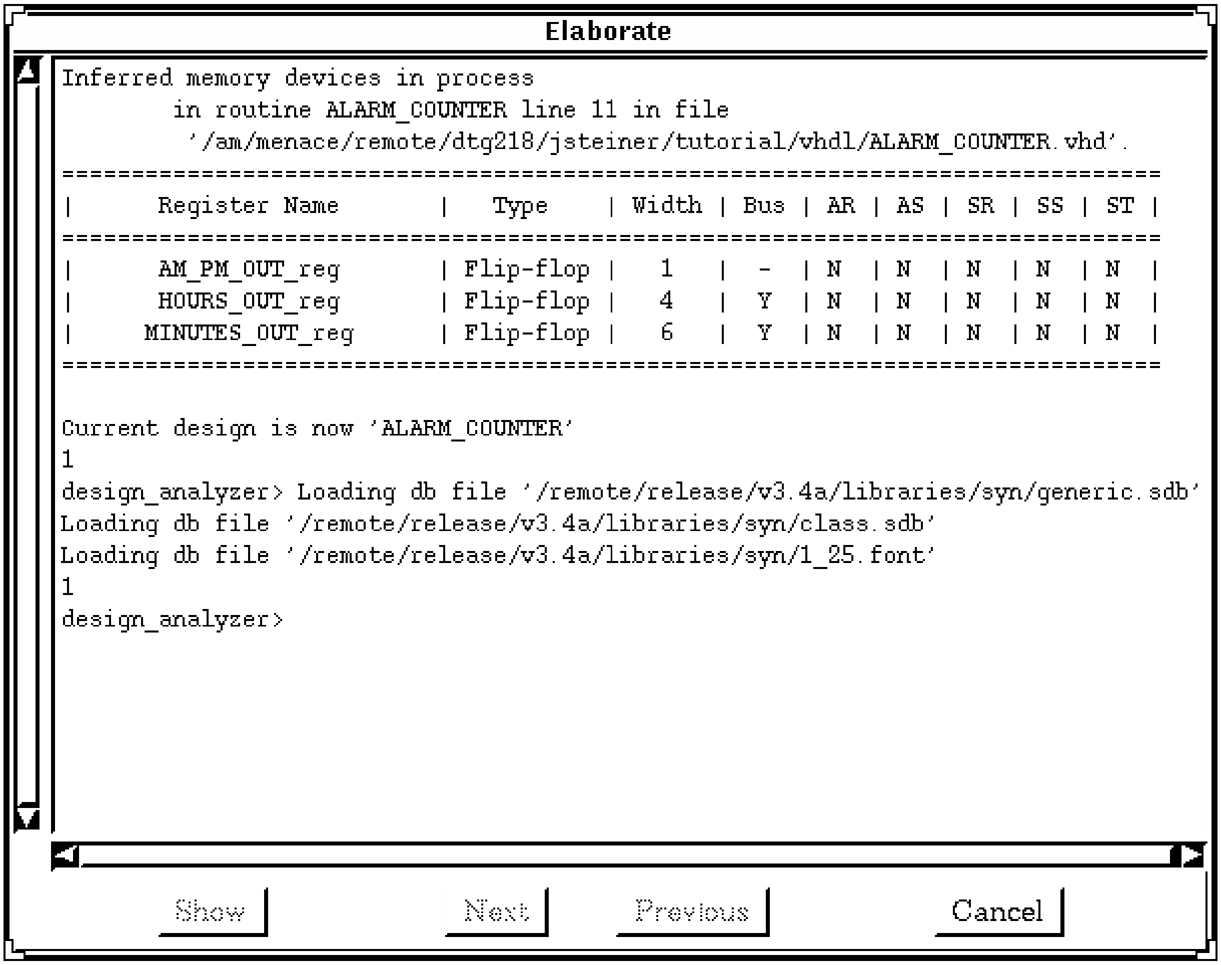
3. Click Re-Analyze Out-Of-Date Libraries to select it.

4. Scroll the Design list and select ALARM_COUNTER(BEHAVIOR).

5. Click OK.

   The Elaborate window, shown in Figure 7-5, appears and displays the activities of the elaborate command.

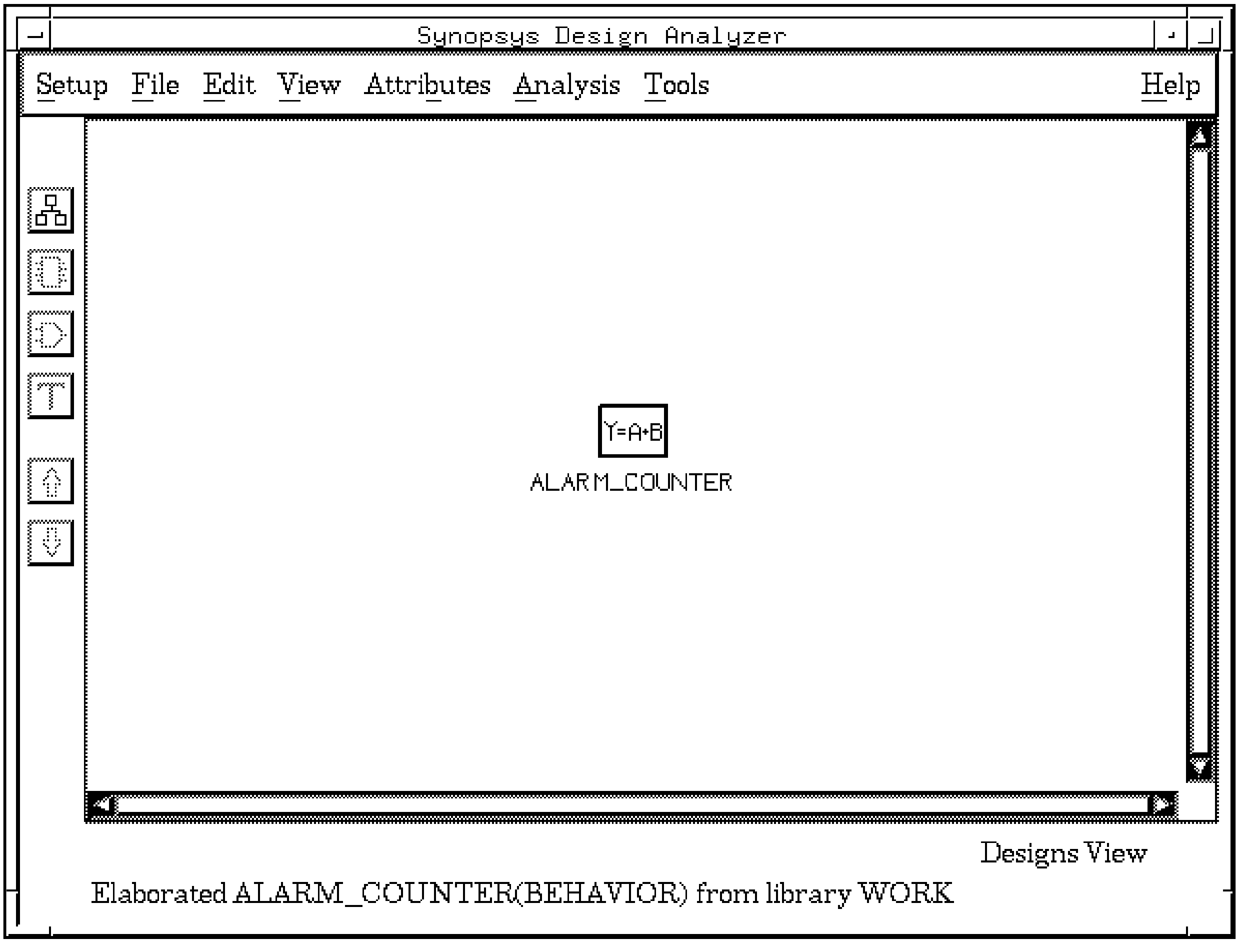
Figure 7-5 Elaborate Window



```
Inferred memory devices in process
        in routine ALARM_COUNTER line 11 in file
         '/am/menace/remote/dtg218/jsteiner/tutorial/vhdl/ALARM_COUNTER.vhd'.
===============================================================================
|      Register Name       |   Type    | Width | Bus | AR | AS | SR | SS | ST |
===============================================================================
|      AM_PM_OUT_reg       | Flip-flop |   1   |  -  | N  | N  | N  | N  | N  |
|      HOURS_OUT_reg       | Flip-flop |   4   |  Y  | N  | N  | N  | N  | N  |
|     MINUTES_OUT_reg      | Flip-flop |   6   |  Y  | N  | N  | N  | N  | N  |
===============================================================================

Current design is now 'ALARM_COUNTER'
1
design_analyzer> Loading db file '/remote/release/v3.4a/libraries/syn/generic.sdb'
Loading db file '/remote/release/v3.4a/libraries/syn/class.sdb'
Loading db file '/remote/release/v3.4a/libraries/syn/1_25.font'
1
design_analyzer>
```

The Design Analyzer window, shown in Figure 7-6, displays the icon for the elaborated ALARM_COUNTER.

Figure 7-6   Synopsys Design Analyzer Window



Elaborated ALARM_COUNTER(BEHAVIOR) from library WORK

6. Repeat the elaboration process (from step 1) for
   ALARM_STATE_MACHINE (BEHAVIOR)
   HOURS_FILTER (BEHAVIOR)
   TIME_COUNTER (BEHAVIOR)
   TIME_STATE_MACHINE (BEHAVIOR)

7. Click Cancel to close the Elaborate window.

The Designs view displays the icons for these designs.

Reading In the PLA Design

Use the read command for formats other than VHDL and Verilog. The CONVERTOR.pla design is in PLA format.

To read in the CONVERTOR block of the Alarm Clock design,

1. Choose File > Read.

   The Read File window appears.

2. Select the db directory.

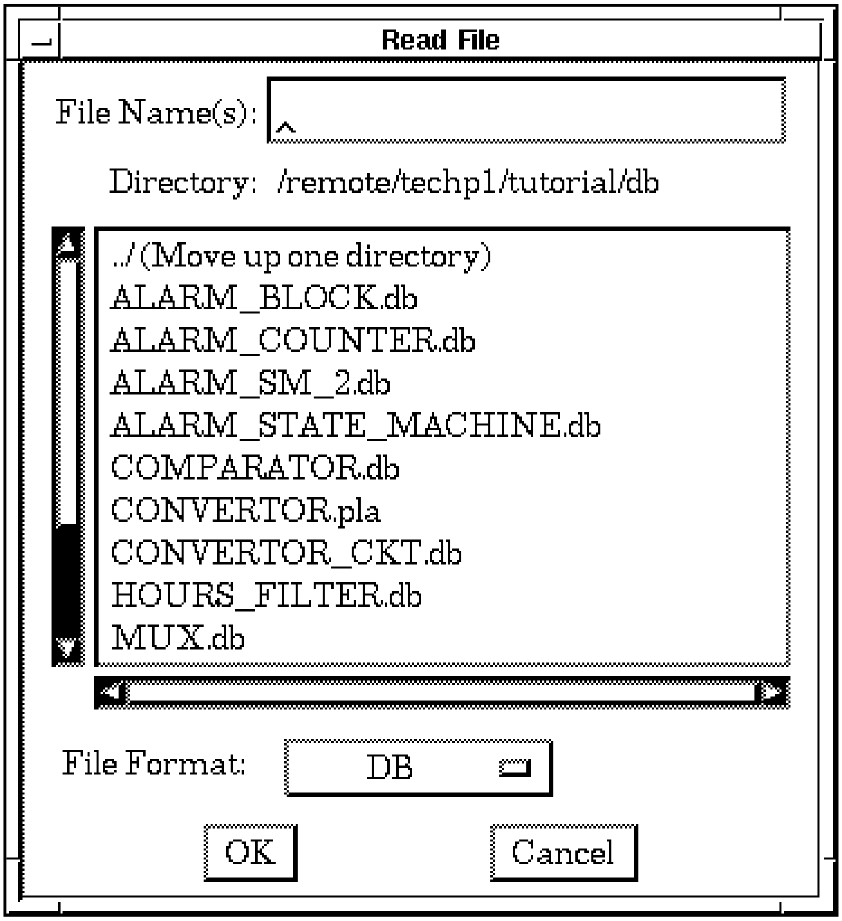
   The db directory contains the design files in Synopsys .db format.

3. Click OK.

   The Read File window appears and lists the design files in the db directory.

   Figure 7-7 shows the Read File window you use to read in designs when you run Design Analyzer under UNIX.

Figure 7-7 Design Analyzer Read File Window in UNIX



File Name(s):

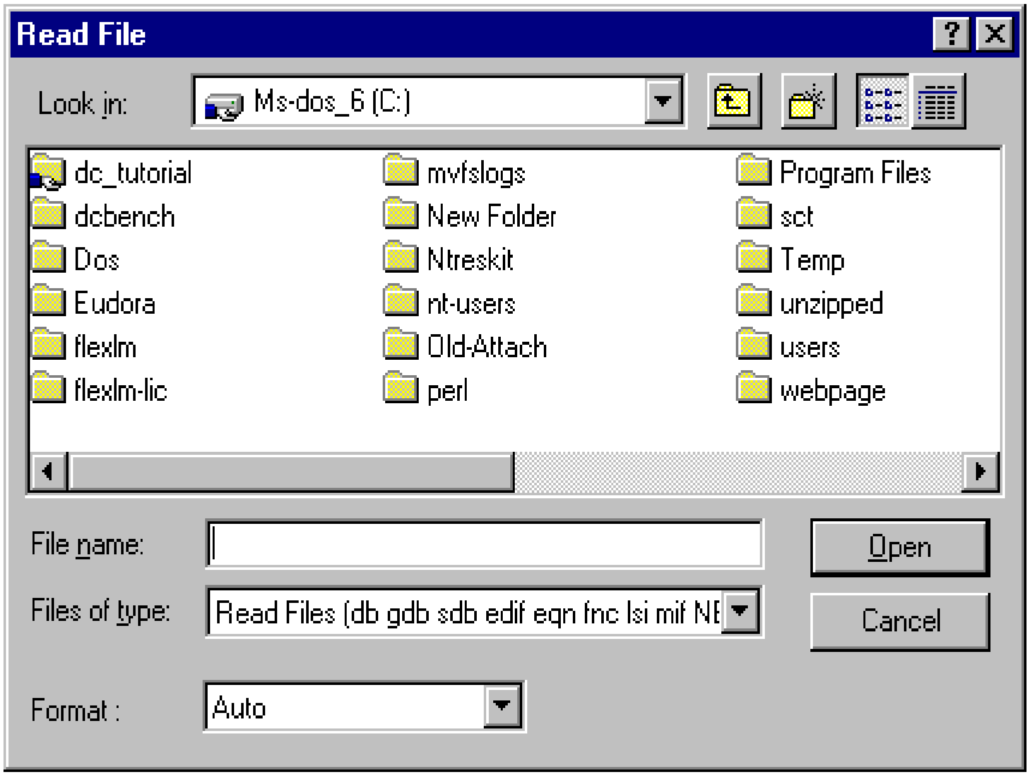
Directory: /remote/techp1/tutorial/db

../ (Move up one directory)
ALARM_BLOCK.db
ALARM_COUNTER.db
ALARM_SM_2.db
ALARM_STATE_MACHINE.db
COMPARATOR.db
CONVERTOR.pla
CONVERTOR_CKT.db
HOURS_FILTER.db
MUX.db

File Format: DB

Figure 7-8 shows the Read File dialog box you use to read in designs when you run Design Analyzer under Windows NT OS.

Figure 7-8   Design Analyzer Read File Dialog Box in Windows NT OS



Look in: Ms-dos_6 (C:)

File name:

Files of type: Read Files (db gdb sdb edif eqn fnc lsi mif NE

Format : Auto

You can also open the db directory by double-clicking the entry in the Read File list.

4. Scroll down to view the remaining files.

   The Read File window lists the files that can be read in from your directory. Only files with suffixes defined by the view_read_file_suffix variable in the system .synopsys_dc.setup file are displayed. This variable defines the suffixes used for design formats accepted by Design Compiler (for example, .vhd for a VHDL file, and .pla for a PLA design file).

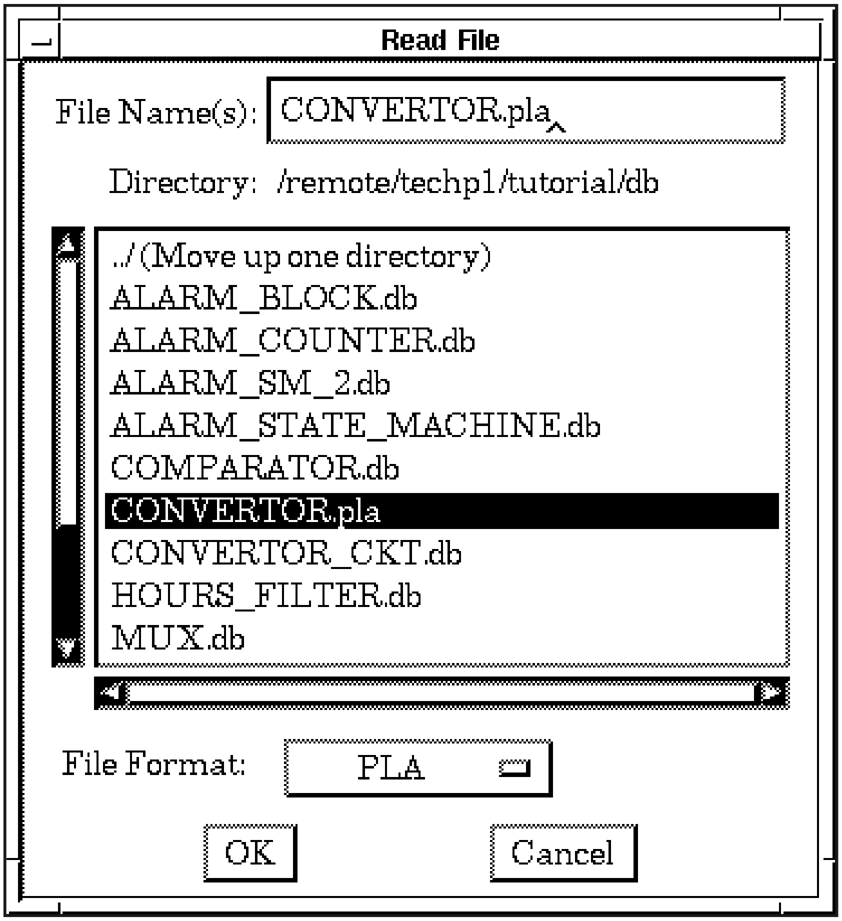
   For more information about view_read_file_suffix and other Synopsys variables, see the Synopsys online man pages for view_read_file_suffix.

5. Select the CONVERTOR.pla file.

   Figure 7-9 shows the Design Analyzer Read File window used to read in the CONVERTOR design in UNIX. The File Name(s) field displays the file name, and the File Format field changes to display PLA.

   The File Format field in the Read File window displays the design file format. The default is db. If you select a file with a format other than db, Design Analyzer automatically displays the correct design format in this field.

Figure 7-9 Reading In the CONVERTOR Design in UNIX



Read File

File Name(s): CONVERTOR.pla

Directory: /remote/techp1/tutorial/db

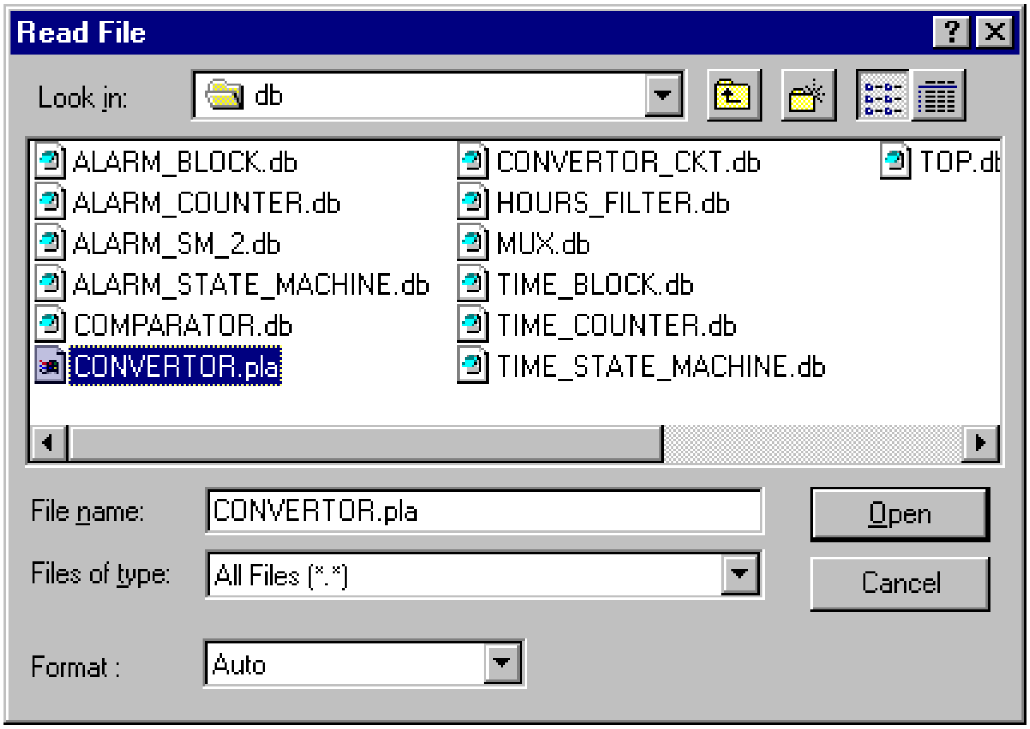
```
../ (Move up one directory)
ALARM_BLOCK.db
ALARM_COUNTER.db
ALARM_SM_2.db
ALARM_STATE_MACHINE.db
COMPARATOR.db
CONVERTOR.pla
CONVERTOR_CKT.db
HOURS_FILTER.db
MUX.db
```

File Format: PLA

Figure 7-10 shows the Design Analyzer Read File dialog box used to read in the CONVERTOR design in Windows NT OS. The File Name(s) field displays the file name, and the File Format field changes to display PLA.

The “Look in:” field shows the name of the directory that contains the design file—in this case, db. The CONVERTOR.pla file is the selected one in the db directory.

Figure 7-10 Reading In the CONVERTOR Design in Windows NT OS



Look in: db

File name: CONVERTOR.pla

Files of type: All Files (*.*)

Format : Auto

6. Choose Setup > Command Window.

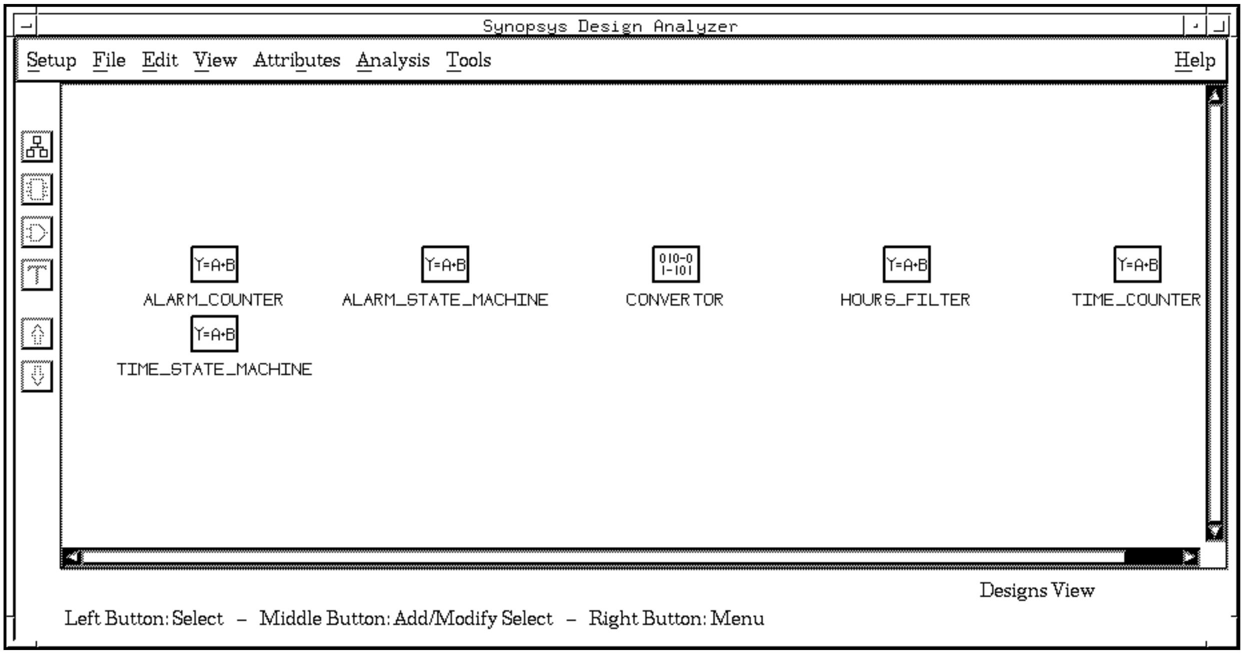
   The Command Window opens.

7. Click OK in the Read File window.

   Design Analyzer reads in the file and closes the Read File window. The Command Window displays the activities.

   Design Analyzer reports reading in CONVERTOR.pla and displays its icon with the other icons for the third-level designs in the Synopsys Design Analyzer window, shown in Figure 7-11.

Figure 7-11 Synopsys Design Analyzer Window



8. Minimize the Command Window.

Reading In the Second Hierarchy Level

The next level up in the hierarchy contains six designs:

ALARM_BLOCK

ALARM_SM_2

COMPARATOR

CONVERTOR_CKT

TIME_BLOCK

MUX

Analyzing the VHDL Designs

You can analyze these design files in one invocation because they are in the same format.

To analyze the six designs in the second level of hierarchy,

1. Choose File > Analyze.

   The Analyze File window appears.

2. Select one design by using the left mouse button. Select ALARM_BLOCK.vhd.
3. Select the remaining designs using the middle mouse button:

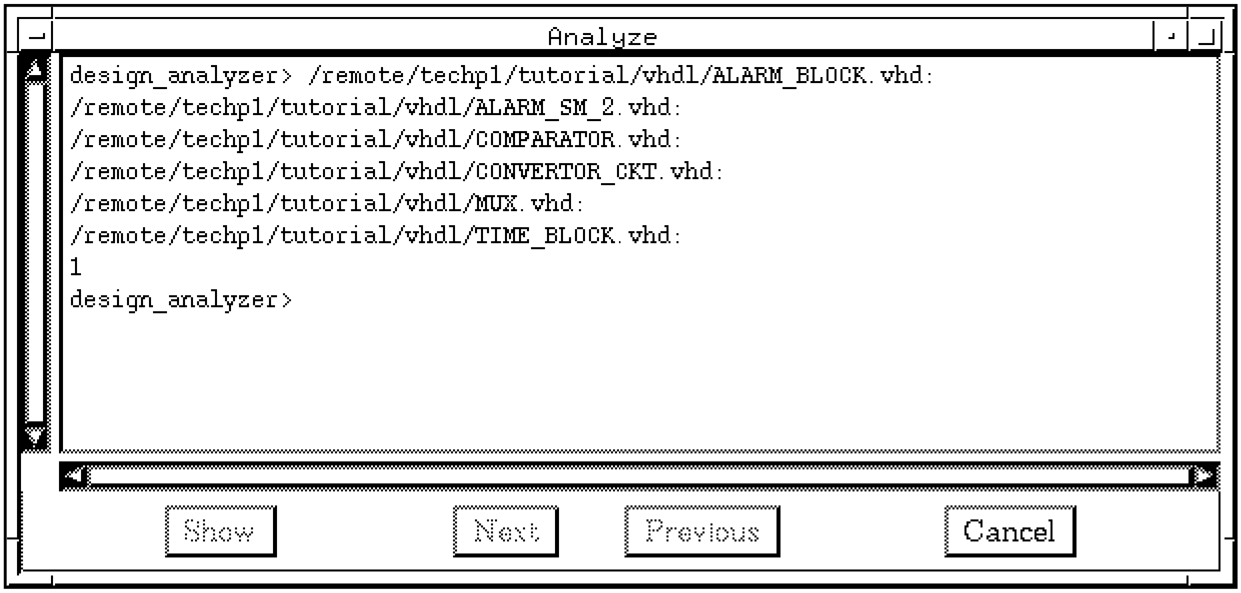
   ALARM_SM_2.vhd
   COMPARATOR.vhd
   CONVERTOR_CKT.vhd
   MUX.vhd
   TIME_BLOCK.vhd

   Each name concatenates in the File Name(s) field.

4. Click OK.

   The Analyze window appears and displays the activities of the analyze command, as shown in Figure 7-12.

Figure 7-12   Analyze Window Displaying Analyze Command Activities



```
design_analyzer> /remote/techp1/tutorial/vhdl/ALARM_BLOCK.vhd:
/remote/techp1/tutorial/vhdl/ALARM_SM_2.vhd:
/remote/techp1/tutorial/vhdl/COMPARATOR.vhd:
/remote/techp1/tutorial/vhdl/CONVERTOR_CKT.vhd:
/remote/techp1/tutorial/vhdl/MUX.vhd:
/remote/techp1/tutorial/vhdl/TIME_BLOCK.vhd:
1
design_analyzer>
```

5. Click Cancel to close the Analyze window.

Elaborating the VHDL Designs

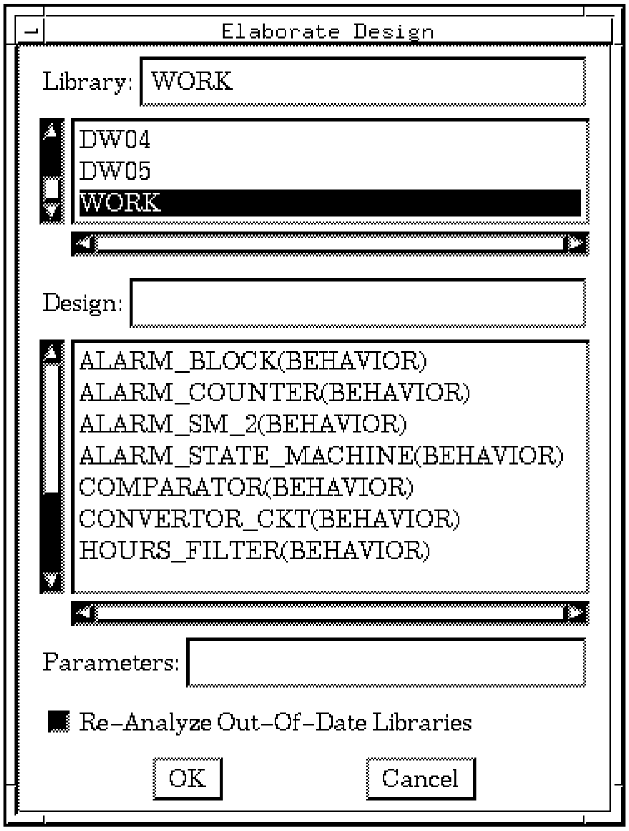
You must elaborate each design individually.

To elaborate the five VHDL designs,

1. Choose File > Elaborate.

   The Elaborate Design window, shown in Figure 7-13, appears. The Library field displays WORK.

Figure 7-13   Elaborate Design Window



Library: WORK

DW04
DW05
WORK

Design:

ALARM_BLOCK(BEHAVIOR)
ALARM_COUNTER(BEHAVIOR)
ALARM_SM_2(BEHAVIOR)
ALARM_STATE_MACHINE(BEHAVIOR)
COMPARATOR(BEHAVIOR)
CONVERTOR_CKT(BEHAVIOR)
HOURS_FILTER(BEHAVIOR)

Parameters:

■ Re-Analyze Out-Of-Date Libraries

2. Select ALARM_BLOCK(BEHAVIOR), and click OK.

   The Elaborate window appears and displays the activities of the elaborate command.

3. Repeat the elaboration process (from Step 1) for

   COMPARATOR (BEHAVIOR)
   CONVERTOR_CKT (BEHAVIOR)
   MUX (BEHAVIOR)
   TIME_BLOCK (BEHAVIOR)
   ALARM_SM_2 (BEHAVIOR)

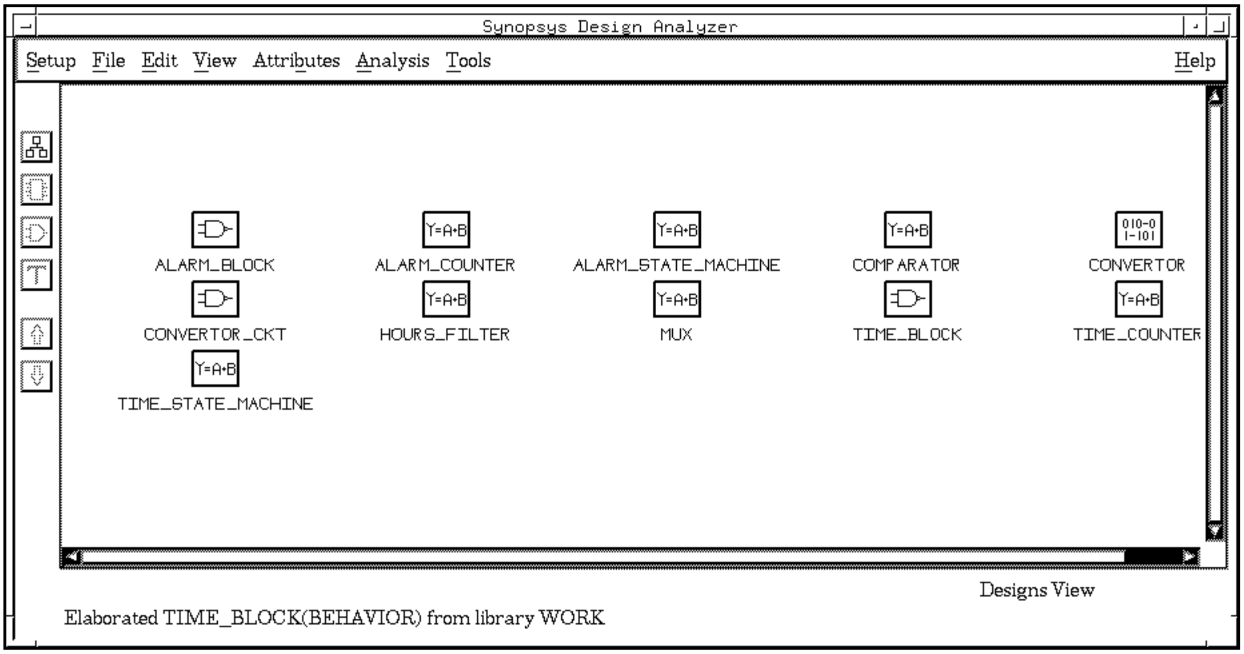
4. Click Cancel to close the Elaborate window.

The Synopsys Design Analyzer window displays the icons for the second-level and third-level designs.

ALARM_BLOCK, TIME_BLOCK, and CONVERTOR_CKT contain no unmapped logic, so the NAND gate icon is used for these three designs. The Y=A+B icon represents designs that are not yet mapped to gates.

Figure 7-14 shows the Synopsys Design Analyzer window for UNIX.

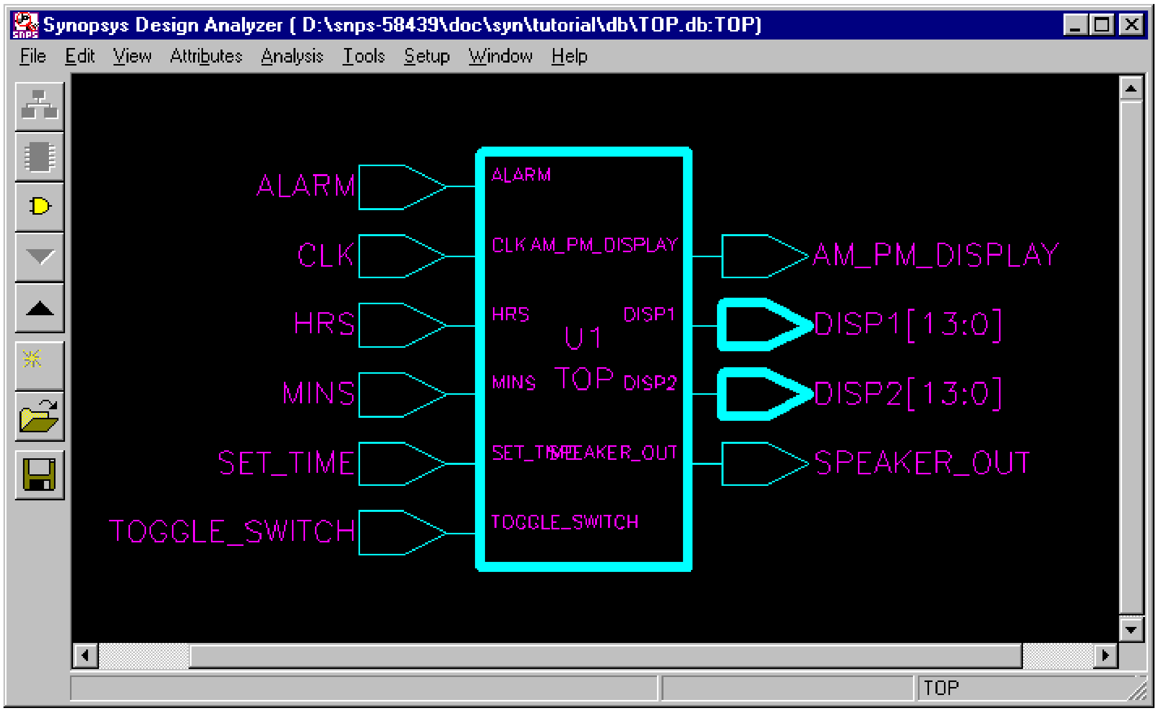
Figure 7-14 Synopsys Design Analyzer Window for UNIX Showing Second and Third Level Design Icons



Elaborated TIME_BLOCK(BEHAVIOR) from library WORK

Figure 7-15 shows the Synopsys Design Analyzer window for Windows NT OS.

Figure 7-15 Synopsys Design Analyzer Window for Windows NT OS Showing Second and Third Level Designs



Reading In the Top-Level Design

Use analyze and elaborate to read in the top-level design.

To analyze TOP,

1. Choose File > Analyze to open the Analyze File window.
2. Select TOP.vhd.
3. Click OK.

   The Analyze window appears and displays the activities.
4. Click Cancel to close the Analyze window.

To elaborate TOP,

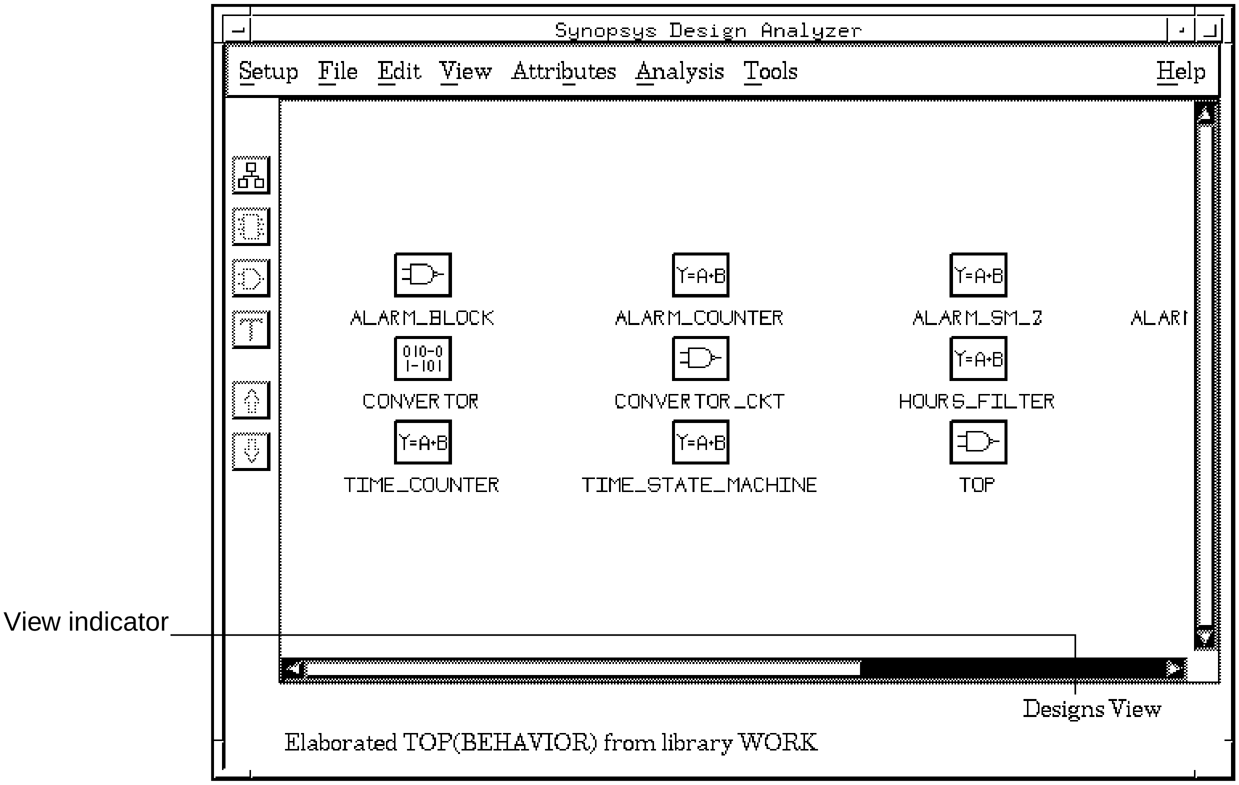
1. Choose File > Elaborate to display the Elaborate Design window.
2. Select TOP (BEHAVIOR).
3. Click OK.

   The Elaborate window appears and displays the activities.
4. Click Cancel to close the Elaborate window.

After you read in TOP, the entire design hierarchy is read in and the Designs view displays the 13 icons for the Alarm Clock design.

To view all 13 icons when using Design Analyzer in UNIX, either scroll horizontally or resize the Synopsys Design Analyzer window, as indicated in Figure 7-16.

Figure 7-16 Synopsys Design Analyzer for UNIX Displaying All Designs



Elaborated TOP(BEHAVIOR) from library WORK

Setting Attributes Using Design Analyzer

After you read a design into Design Compiler, you specify the design environment. Specify the design environment by setting attributes that define information such as drive strengths on the ports, when signals arrive on the ports, or the load driven by the output ports.

This section describes the tasks you perform to set the attributes for the Alarm Clock design example, using Design Analyzer.

This section includes the following:

- About Setting Attributes for the TOP Design

- Setting the Drive Strength on Input Ports
- Setting the Drive Strength for CLK
- Setting the Load on Output Ports
- Setting Other Attributes at the Top Level

About Setting Attributes for the TOP Design

You set attributes on design objects by using the Symbol view.

To display the Symbol view,

1. Select TOP.
2. Click the down arrow.
3. Click the Symbol View button.

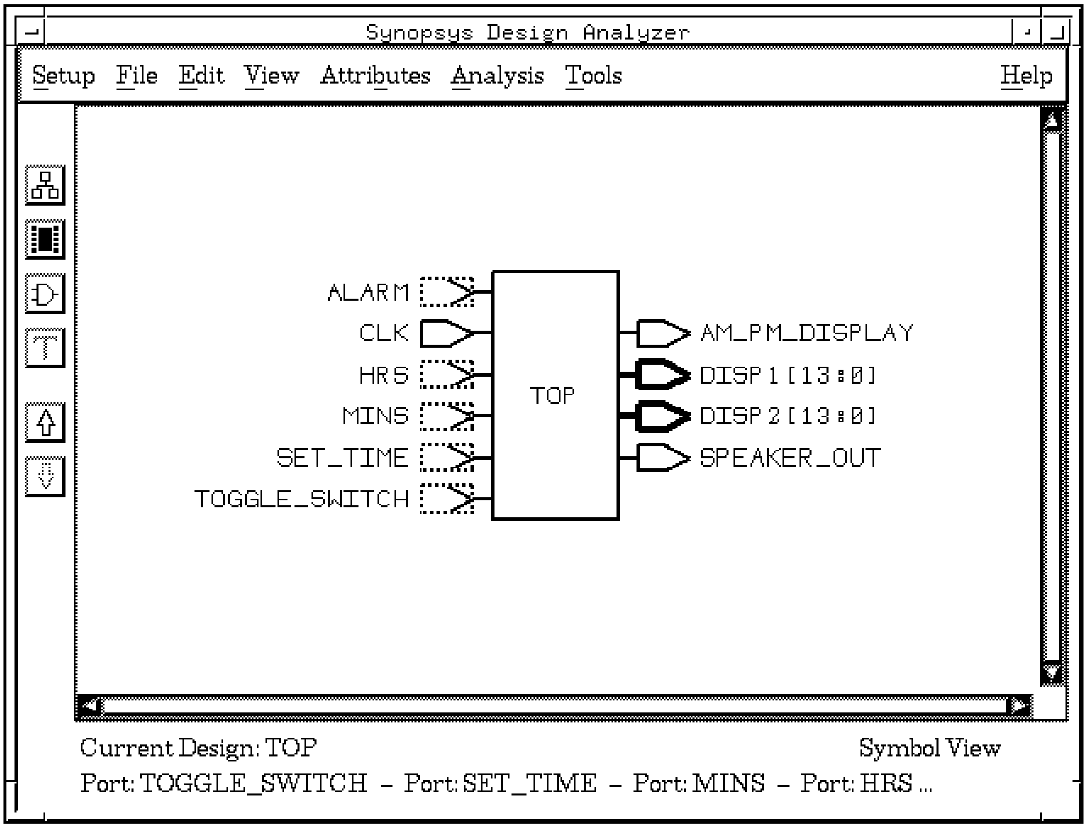
Setting the Drive Strength on Input Ports

Assume that all input ports except CLK have a drive strength of 0.08 (units are determined by the target library). You can set these input ports simultaneously by selecting all the appropriate ports.

To select ports,

1. Use the left mouse button to select the input port ALARM.
2. Use the middle mouse button to select the other input ports, except CLK. (In the next task, you set the drive strength for CLK by using a different method.) See Figure 7-17.

Figure 7-17 Synopsys Design Analyzer Window for UNIX Showing Selected Ports



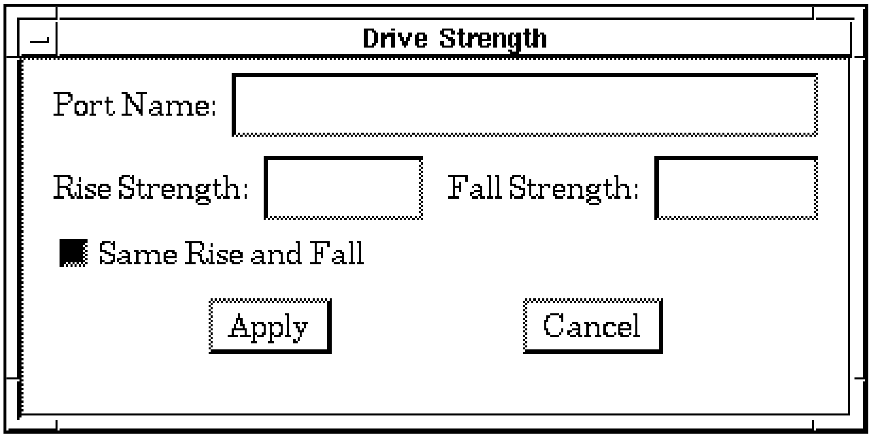
Current Design: TOP

Port: TOGGLE_SWITCH – Port: SET_TIME – Port: MINS – Port: HRS ...

To set drive strengths,

1. Choose Attributes > Operating Environment > Drive Strength to display the Drive Strength window, as shown in Figure 7-18.

Figure 7-18   Drive Strength Window



If you select one port, the Port Name field displays the name of the port when you open the Drive Strength window. If you select more than one port, the Port Name field is blank. Values entered in the window are set for all selected ports.

2. Type the value 0.08 in the Rise Strength field. The Fall Strength field is set automatically to 0.08 because the “Same Rise and Fall” option is set by default,.
3. Click Apply to set the values.

You can set port drive values equal to the drive strength of library cell output pins. When you do not know the value of the pin you want in the library, you use the drive_of command to find out and set the drive value. Assume that the drive strength on CLK needs to be equal to the drive strength of pin Z of the driver cell B4I, a buffer in the target library.

Note:

> This tutorial uses set_drive to set drive strength. However, set_driving_cell offers an advantage over the set_drive command. The set_driving_cell command associates an input with a driving

cell rather than with a specific drive value. See the *Design Compiler Reference Manual* for more information about Design Compiler commands.

Setting the Drive Strength for CLK

Rather than set the drive strength to 0.08, in this exercise, you set the drive strength of CLK equal to pin Z of B4I.

To set drive strength for CLK,

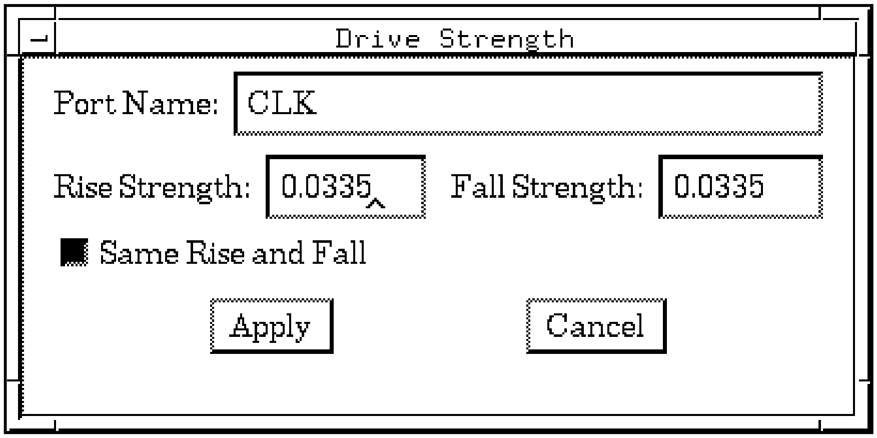
1. Select CLK in the Symbol view.
2. Click in the Rise Strength field, and press Ctrl-u to delete the value 0.00.
3. In the Rise Strength field, enter the following:

   ```
   drive_of (class/B4I/Z)
   ```

   Design Analyzer duplicates this command in the Fall Strength field. (Assume that B4I is a balanced driver—the rise and fall drive values are equal).
4. Click Apply.

   The computed rise and fall values (0.0335) appear, as shown in Figure 7-19.

Figure 7-19 Computed Rise and Fall Values Shown in the Drive Strength Window



Port Name: CLK

Rise Strength: 0.0335 Fall Strength: 0.0335

■ Same Rise and Fall

Changing the Drive Strength

You can change attribute values after they are set. Assume that the drive strength set previously for input port SET_TIME is incorrect. Change it with the following procedure.

To change the drive strength for SET_TIME,

1. Select the SET_TIME input port.

   The Drive Strength window changes to reflect the values for SET_TIME.

2. Replace the contents of the Rise Strength field with 0.06.

   The Fall Strength field is updated simultaneously with this value.

3. Click Apply to set the new drive strength.

4. Click Cancel to close the Drive Strength window.

Setting the Load on Output Ports

For this exercise, define loads for the output ports, as discussed in the following sections:

- Setting the Load for SPEAKER_OUT.
- Setting the Load on Bus Bits.
- Setting the Load for AM_PM_DISPLAY.

Setting the Load for SPEAKER_OUT

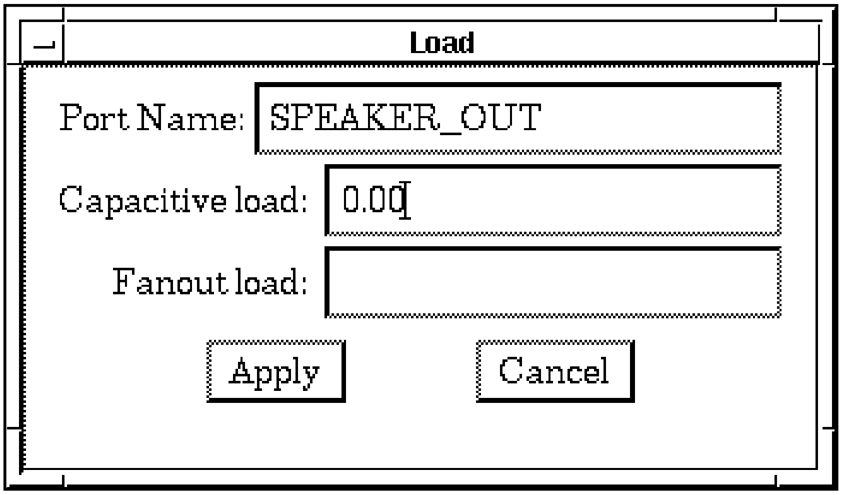
Load values are used to model the capacitive load on the output ports of the constrained module.

You can set port load values equal to the load values of library cell input pins. When you don’t know the library cell value you want, use the load_of command to determine and set a load value. Assume port SPEAKER_OUT drives a load of five inverters. Assume also that the inverters are the same as cell IVA (an inverter in the target library class).

To set the load for SPEAKER_OUT,

1. Select the SPEAKER_OUT port.
2. Choose Attributes > Operating Environment > Load to open the Load window, as shown in Figure 7-20.

Figure 7-20   Load Window



Port Name: SPEAKER_OUT

Capacitive load: 0.00

Fanout load:

3. Replace the contents of the Capacitive load field with

   `load_of (class/IVA/A) * 5`

   Leave a space before and after the asterisk (*). Note that pin A is the input pin of IVA.

4. Click Apply.

   The computed capacitive load value 7.50 appears.

   SPEAKER_OUT drives a load of five inverters. The load value of each inverter is 1.5 (load of cell IVA, pin A in the target library), so the load value for SPEAKER_OUT is calculated as
   1.5 * 5 = 7.50.

5. Click Cancel to dismiss the Load window.

Setting the Load on Bus Bits

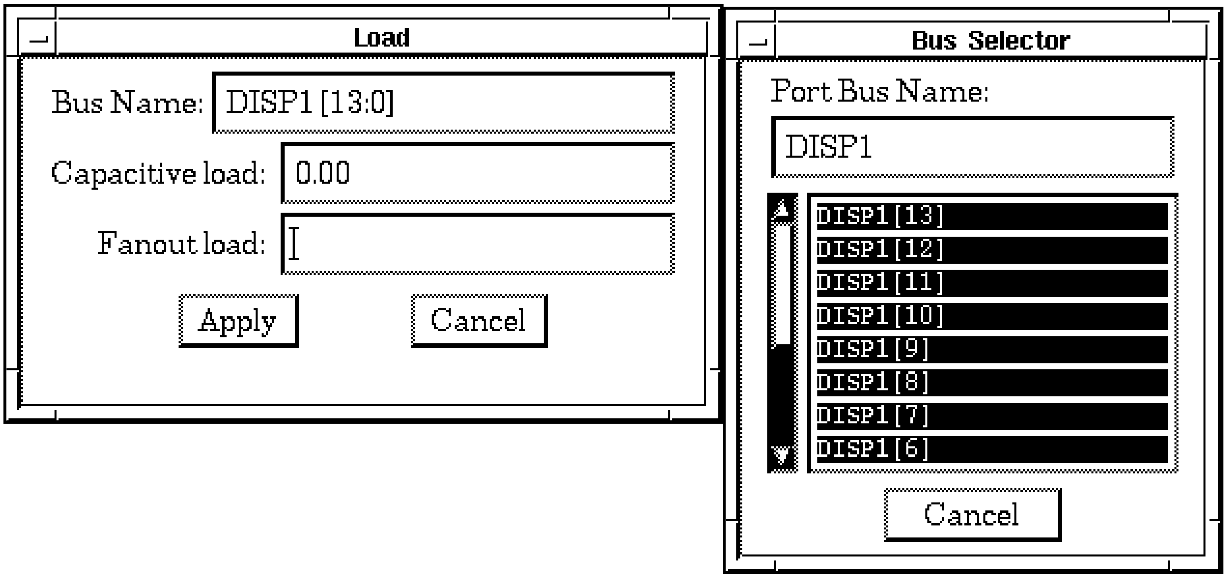
Assume that ports DISP1 and DISP2 each drive a load of 3 (standard loads) and AM_PM_DISPLAY drives a load of 2 (standard loads).

To set load values on DISP1,

1. Select DISP1.
2. Choose Attributes > Operating Environment > Load.

   Two windows appear—Load and Bus Selector—as shown in Figure 7-21. If the windows overlap, move one beside the other so both are fully visible.

Figure 7-21 Load and Bus Selector Windows



Bus Name: DISP1 [13:0]

Capacitive load: 0.00

Fanout load:

Port Bus Name: DISP1

DISP1[13]
DISP1[12]
DISP1[11]
DISP1[10]
DISP1[9]
DISP1[8]
DISP1[7]
DISP1[6]

   Use the Bus Selector window to select the bits on which to set attributes. In this window, all bits are selected. If you want to set attributes on a single bit, click that bit, and then type the values in the Load window.

   For this exercise, set the same attributes for the entire bus.

3. Click Cancel in the Bus Selector window.
4. Type the value 3 in the Capacitive load field of the Load window.

5. Click Apply.

   The load value is set for DISP1.

To set load values on DISP2,

1. Select DISP2 in the Design Analyzer window.
2. Type 3 in the Capacitive load field.
3. Click Apply.

Setting the Load for AM_PM_DISPLAY

Set the load on the remaining output port to 2.0.

To set the capacitive load for AM_PM_DISPLAY,

1. Select AM_PM_DISPLAY.
2. Type 2 in the Capacitive load field.
3. Click Apply.
4. Click Cancel in the Load window.

Setting Other Attributes at the Top Level

Some attributes are set at the top level of a design but are not associated with a particular input or output port. Such attributes can have a global effect on optimization of the design and the hierarchical blocks the design contains.

For this exercise, set attributes on the TOP design as discussed in the following sections:

- Setting the Wire Load for the Design.

- Setting the Operating Conditions for the Design.

Setting the Wire Load for the Design

Design Compiler optimization uses net fanout as a basis for estimating interconnect wire length from the wire load model. Design Compiler uses this information to calculate interconnect wiring and transition delays.

The wire load model for a design depends on the estimated die size of the design. Wire load models are defined in the target library.

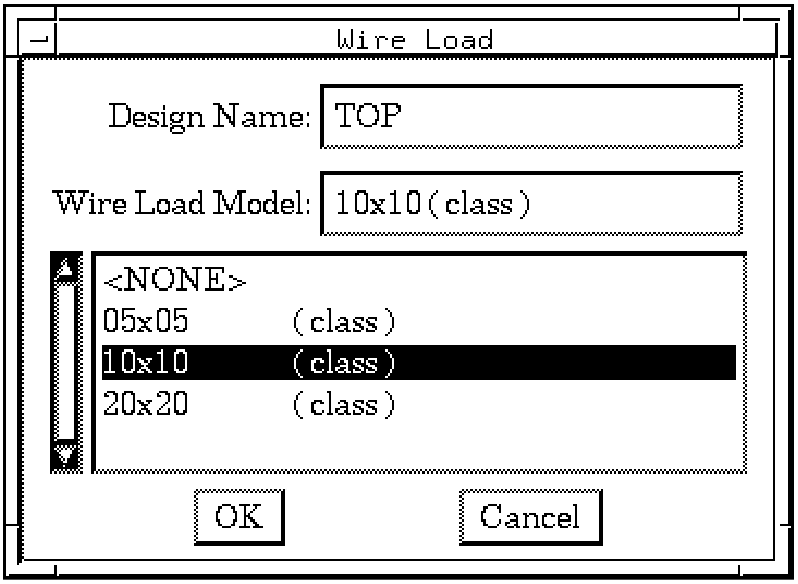
Design Compiler uses area as a basis for automatically selecting the wire load tables if your ASIC libraries support this feature.

To set the wire load on TOP,

1. Select TOP.
2. Choose Attributes > Operating Environment > Wire Load.

   The Wire Load window, shown in Figure 7-22, opens, listing the wire load models and the target library.

Figure 7-22 Wire Load Window



Design Name: TOP

Wire Load Model: 10x10 (class)

```
<NONE>
05x05       ( class )
10x10       ( class )
20x20       ( class )
```

3. Select 10x10 (class).

   As defined in the library file, 10x10 corresponds to a die size of 1 mm x 1 mm.

4. Click OK to set the wire load model and close the window.

Setting the Operating Conditions for the Design

Operating conditions are the temperature, process, and voltage in which the design operates. The target library defines the operating conditions. Library vendors define default operating conditions, which can differ from one vendor to another.

Common default operating conditions are

- Temperature – 25° C
- Process – 1

- Voltage – 5

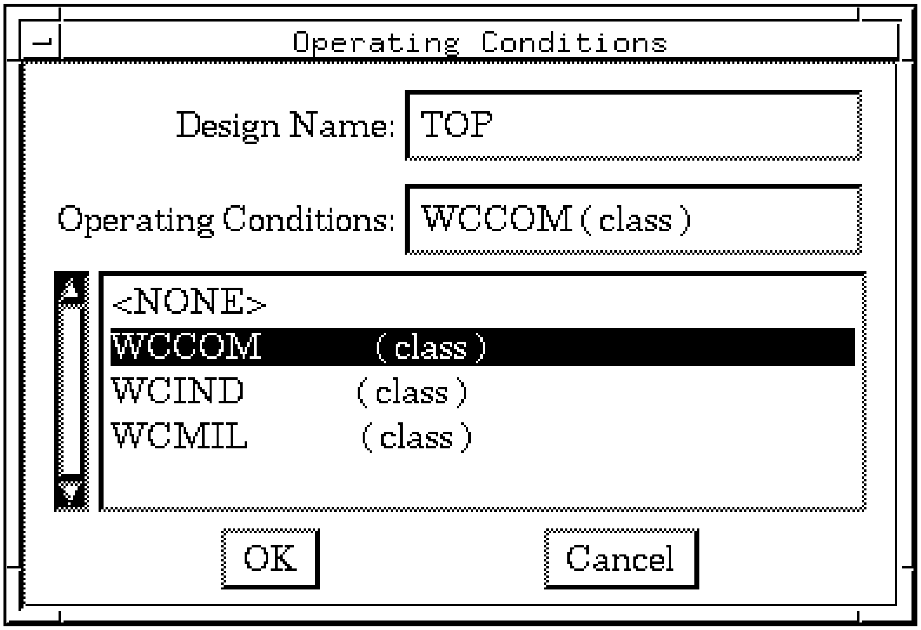
The Design Compiler static timing analyzer models the effects of variation in the drive strength, arrival time, and load values on a circuit’s timing characteristics. In a similar way, you can analyze a design for best-case, nominal-case, and worst-case performance or operating conditions.

To set the operating conditions for the design,

1. Choose Attributes > Operating Environment > Operating Conditions.

   The Operating Conditions window, shown in Figure 7-23, appears and lists operating conditions from the target library. Each set of operating conditions is followed by the name of the target library in parentheses.

Figure 7-23 Operating Conditions Window



Design Name: TOP

Operating Conditions: WCCOM (class)

2. Select WCCOM (class) in the Operating Conditions window.

   For this design, assume that operating conditions are worst-case commercial (WCCOM), as defined in the target library, class.db. As specified in the library file, the temperature is 70.0 ,Fthe process is 1.5, and the voltage is 4.75.

3. Click OK.

   The operating conditions are set.

Saving the Design Using Design Analyzer

After setting attributes on the design, save the design to preserve your attribute settings.

To save TOP design in .db format,

1. Choose File > Save As to open the Save Design window.

   The File Name field displays TOP.db.

2. Change to your tutorial/db/ directory.
3. Enter TOP_attributes.db in the File Name field to save this file in your db directory.
4. Verify that the Save All Designs in Hierarchy option is set to on.
5. Click OK.

   Design Analyzer saves the design and attribute settings.

Then quit Design Analyzer (as described in the following section) or go on to the next chapter.

To quit Design Analyzer,

- Enter quit in the Design Analyzer Command Window text field.

  or

  Choose File > Quit.

When you quit Design Analyzer, licenses checked out for your Design Analyzer session are automatically checked back in.

Using dc_shell to Set the Design Environment

This section includes the following subsections:

- Starting dc_shell

Explains how to start the dc_shell interface to Design Compiler in UNIX and in the Windows NT OS.

- Reading In a Hierarchical Design Using dc_shell

  Contains tasks that specify the command or commands you enter at the dc_shell prompt to read in the entire design, level by level. Exercises to read in the design begin with the VHDL package, followed by the three hierarchy levels from lowest to TOP.

- Setting Attributes Using the dc_shell Interface

  Contains tasks (and the dc_shell commands to carry them out) for setting the drive strength on input ports, setting the load on output ports, and setting other attributes at the top level of the design.

- Saving the Design Using dc_shell

  Explains how to save the design environment settings, using the dc_shell interface.

Note:

Commands are given in both dcsh mode and dctcl mode. However, before you can run dcsh-mode commands or dctcl-mode commands, you must ensure that you are using the correct setup file. Your tutorial directory contains setup files for both modes. The dcsh-mode file name is .synopsys_dcsh.setup, and the dctcl-mode file name is .synopsys_dctcl.setup. Copy the appropriate file to .synopsys_dc.setup, depending on which mode you intend use when running the dc_shell commands.

At any point in these exercises, you can save the design with the settings you specified. Then you can read the design into Design Analyzer and perform some or all of the remaining tasks with Design Analyzer. You might want to do this, for example, to collectively analyze a group of designs.

You can complete the exercises in this chapter in about two hours.

Starting dc_shell

Before launching dc_shell, set up the tutorial as described in Chapter 5, “Setting Up the Tutorial.”

This section includes the following:

- Starting dc_shell in UNIX
- Starting dc_shell in the Windows NT OS

Starting dc_shell in UNIX

dc_shell provides two Design Compiler command modes with two corresponding scripting languages. The *dcsh mode* uses a command language developed by Synopsys, and the *Tcl mode* is based on the Tool Command Language (Tcl). To start Design Compiler, you enter the dc_shell command. The dcsh mode is the default mode. Use the -tcl_mode switch to run in the Tcl mode.

To start the dc_shell command-line interface,

- At the operating system command prompt, type

```
mysystem% dc_shell
```

which launches dc_shell in dcsh mode and displays the dc_shell command-line prompt

```
dc_shell>
```

or type

```
mysystem% dc_shell -tcl_mode
```

which launches dc_shell in Tcl mode and displays the dc_shell command-line prompt

```
dc_shell-t>
```

Starting dc_shell in the Windows NT OS

To run dc_shell in the Windows NT OS from a command prompt window, make sure the PATH statement includes the path to the Synopsys installation directory.

Note:

> This requirement applies also if you use the Design Analyzer interface and launch Design Analyzer from within dc_shell.

Prerequisite Task

If the PATH statement does not already include the Synopsys executable directory path, modify your PATH variable before you start dc_shell to include the following directory path:

c:\synopsys\msvc50\syn\bin

You can use the Windows NT OS set command or the System Properties page to set the PATH variable.

Note:

If your Synopsys software is installed in a directory other than c:\synopsys, specify the actual installation directory.

Starting dc_shell

You can start dc_shell in either the dcsh or dctcl mode.

To start dc_shell using a command prompt window,

1. Open a command prompt window.
2. At the command prompt, type either

   ```
   c:\> dc_shell
   ```

   which launches dc_shell in dcsh mode and displays the dc_shell command-line prompt

   ```
   dc_shell>
   ```

   or type

   ```
   c:\> dc_shell -tcl_mode
   ```

   which launches dc_shell in dctcl mode and displays the dc_shell command-line prompt

   ```
   dc_shell-t>
   ```

Reading In a Hierarchical Design Using dc_shell

This section provides conceptual information underlying the tasks you perform. Then it describes the set of exercises that you undertake to read in the Alarm Clock hierarchical design example using dc_shell commands. Commands are presented in dcsh and dctcl mode.

Here are the contents of this section:

- Reading In a Hierarchical Design
- Saving the Design at Any Point Using dc_shell
- Reading In the VHDL Package Using dc_shell
- Reading In the Lowest Hierarchy Level
- Reading In the PLA Design
- Reading In the Second Hierarchy Level
- Reading In the Top-Level Design
- Analyzing the Top-Level Design
- Elaborating the VHDL Designs

Throughout these exercises, notice that both dc_shell modes display the number 1 following output produced during command execution. The number 1 is a completion code indicating that the command completed successfully.

Reading In a Hierarchical Design

One way to make a large or complex design easier to manipulate is to divide it into smaller components. Each component performs a function of the design. The components are called subdesigns. A design that contains one or more subdesigns is a hierarchical design. Hierarchical designs can have several levels—each containing one or more subdesigns. The design that calls or references the subdesigns in the hierarchy is the top-level design. The Alarm Clock design is hierarchical, and TOP is the top-level design.

The design is a VHDL netlist and references the subdesigns in the hierarchy.

Viewing the Top-Level Design File in UNIX

Under UNIX, to view the contents of the top-level design file, change to your tutorial/vhdl directory and enter

```
% cat TOP.vhd
```

Viewing the Top-Level Design File in the Windows NT OS

Under Windows NT OS, to view the contents of the top-level design file, use the Windows NT Explorer to change to your tutorial\vhdl directory and open the TOP.vhd file in WordPad or another word processor.

Saving the Design at Any Point Using dc_shell

At the end of each section of the tutorial is an explanation of how to save your work on the sample design up to that point. Because you might want to save the design at earlier intervals—that is, before completing the set of tutorial exercises—this section gives a brief introduction to the write command, which you can use for this purpose.

To save design settings, you use the dc_shell write command. This command writes the design to disk, including any subdesigns in the hierarchy. Here is the syntax for the write command:

```
write [-format format] [-hierarchy] [-no_implicit]
      [-modified] [-output file] [-library library_name]
      [design_list] [-names_file mapping_name_files]
      [-donot_expand_dw]
```

For a complete explanation of the write command and its parameters, see the *Design Compiler User Guide*.

Reading In the VHDL Package Using dc_shell

If you use VHDL format, you must read in a VHDL package before you read in the design files. The package is synopsys.vhd and is located in the vhdl directory.

If you do not use VHDL format, skip to the next section, “Reading In the Lowest Hierarchy Level."

To read in the VHDL package,

- Enter the one of the following commands at the dc_shell prompt:

  ```
  dc_shell> read -format vhdl {“./vhdl/synopsys.vhd”}
  ```

  or

  ```
  dc_shell-t> read_file -format vhdl [list {./vhdl/
  synopsys.vhd}]
  ```

Example Output

After executing the read command successfully, dc_shell displays the following output:

```
Loading db file ‘/usr/synopsys/myarch/libraries/syn/standard.sldb’
Loading db file ‘/usr/synopsys/myarch/libraries/syn/gtech.db’
Loading vhdl file ‘/usr/synopsys/tutorial/vhdl/synopsys.vhd’
Reading in the Synopsys vhdl primitives.
/bohm/tutorial/vhdl/synopsys.vhd:
Information: Saving the package ‘synopsys’. (HDL-202)
No designs were read
{}
```

The following sections describe using the analyze and elaborate commands to read VHDL designs.

Reading In the Lowest Hierarchy Level

To ensure that all references are resolved correctly (and to eliminate occurrence of unresolved references), read in the designs starting with the lowest (third) level subdesigns and work your way up to the top-level design.

The lowest level of hierarchy contains these designs:

ALARM_COUNTER

ALARM_STATE_MACHINE

CONVERTOR (2)

HOURS_FILTER

TIME_COUNTER

TIME_STATE_MACHINE

Use the read command to read in CONVERTOR because it is in PLA format.

Analyzing the VHDL Designs

Use the analyze and elaborate commands to read in the five VHDL files. You can analyze these files in one invocation because they have the same format. The WORK directory is initially empty. As you analyze designs, Design Compiler stores the resulting files in the WORK directory.

To analyze the lowest hierarchy level,

- Enter one of the following dc_shell commands at the command prompt:

```
dc_shell> analyze -format vhdl -lib WORK {./vhdl/ALARM_COUNTER.vhd,
./vhdl/ALARM_STATE_MACHINE.vhd, ./vhdl/HOURS_FILTER.vhd, ./vhdl/
TIME_COUNTER.vhd,./vhdl/TIME_STATE_MACHINE.vhd}
```

or

```
dc_shell-t> analyze -format vhdl -lib WORK [list ./vhdl/ALARM_COUNTER.vhd
./vhdl/ALARM_STATE_MACHINE.vhd ./vhdl/HOURS_FILTER.vhd ./vhdl/
TIME_COUNTER.vhd ./vhdl/TIME_STATE_MACHINE.vhd]
```

Example Output

When you analyze the lowest level of the VHDL design using the analyze command, dc_shell generates the following output:

```
/bohm/tutorial/vhdl/ALARM_COUNTER.vhd:
/bohm/tutorial/vhdl/ALARM_STATE_MACHINE.vhd:
/bohm/tutorial/vhdl/HOURS_FILTER.vhd:
/bohm/tutorial/vhdl/TIME_COUNTER.vhd:
/bohm/tutorial/vhdl/TIME_STATE_MACHINE.vhd:
1
```

Note:

In this output example, and in all cases shown in the tutorial, the numeral 1 concluding the output is a completion code indicating that the command completed successfully.

Elaborating the VHDL Designs

Elaborate the five VHDL designs individually. The elaborate command translates the intermediate design files created by analyze into .db format. Elaborating a given design makes it the current one.

In analyzing the VHDL designs for the tutorial, you specify BEHAVIOR as the architecture. (In VHDL, the default architecture is the most recently analyzed architecture.) You use the -lib parameter of the analyze command to specify the name of the library to which the work is mapped. By default, Design Compiler looks in the WORK library for the design to be built. The -lib parameter allows you to temporarily change this directory. However, for the tutorial, use the WORK directory.

To elaborate the designs,

- Enter each of the following commands at the dc_shell prompt:

```
dc_shell> elaborate ALARM_COUNTER -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate ALARM_STATE_MACHINE -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate HOURS_FILTER -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate TIME_COUNTER -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate TIME_STATE_MACHINE -arch “BEHAVIOR” -lib WORK -update
```

or

```
dc_shell-t> elaborate ALARM_COUNTER -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate ALARM_STATE_MACHINE -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate HOURS_FILTER -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate TIME_COUNTER -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate TIME_STATE_MACHINE -arch {BEHAVIOR} -lib WORK -update
```

Example Output

When you elaborate a VHDL design, using the elaborate command, dc_shell generates output similar to the output shown below. Elaborating a given design makes it the current one. The elaborate commands for the VHDL designs include the -update keyword, which directs the compiler to automatically reanalyze out-of-date intermediate files if the source is available.

Here is an example of the elaborate command and the output it produces:

```
dc_shell> elaborate ALARM_COUNTER -arch “BEHAVIOR” -lib WORK -update

Inferred memory devices in process
             in routine ALARM_COUNTER line 11 in file
                 ’/bohm/tutorial/vhdl/ALARM_COUNTER.vhd’.
 ===========================================================================
   Register Name   |  Type         | Width | Bus | MB | AR | AS | SR | SS | ST
```

```
=============================================================================
    AM_PM_OUT_reg | Flip-flop     | 1     |   - |  -  |  N |  N |  N  | N  |  N
    HOURS_OUT_reg | Flip-flop     | 4     |  Y  |  N  | N  |  N |  N  | N  |  N
  MINUTES_OUT_reg | Flip-flop     | 6     |  Y  |  N  | N  |  N |  N  | N  |  N
===============================================================================
Current design is now ’ALARM_COUNTER’
1
```

Reading In the PLA Design

You use the read command—not the analyze and elaborate commands—for formats other than VHDL and Verilog. The CONVERTOR.pla design is in PLA format.

To read in the CONVERTOR block of the Alarm Clock design,

- Enter

  ```
  dc_shell> read -format pla {“./vhdl/CONVERTOR.pla”}
  ```

  or

  ```
  dc_shell-t> read_file -format pla [list {./vhdl/
  CONVERTOR.pla}]
  ```

Example Output

When you read in the CONVERTOR block by using the read command, you specify the input format as PLA. The dc_shell compiler executes the command and generates the output shown below.

```
dc_shell> read -format pla {“./vhdl/CONVERTOR.pla”}


Loading pla file ‘/bohm/tutorial/vhdl/CONVERTOR.pla
Current design is now ‘/bohm/tutorial/vhdl/
CONVERTOR.db:CONVERTOR’ {“CONVERTOR”}
```

Reading In the Second Hierarchy Level

The next level up in the hierarchy contains six designs:

ALARM_BLOCK

ALARM_SM_2

COMPARATOR

CONVERTOR_CKT

TIME_BLOCK

MUX

Analyzing the VHDL Designs

To analyze all designs in the second hierarchy level,

- Enter one of the following commands at the dc_shell prompt:

```
dc_shell> analyze -format vhdl -lib WORK {./vhdl/ALARM_BLOCK.vhd,
./vhdl/ALARM_SM_2.vhd, ./vhdl/COMPARATOR.vhd, ./vhdl/CONVERTOR_CKT.vhd,
./vhdl/MUX.vhd, ./vhdl/TIME_BLOCK.vhd}
```

or

```
dc_shell-t> analyze -format vhdl -lib WORK [list ./vhdl/ALARM_BLOCK.vhd
./vhdl/ALARM_SM_2.vhd ./vhdl/COMPARATOR.vhd ./vhdl/CONVERTOR_CKT.vhd
./vhdl/MUX.vhd ./vhdl/TIME_BLOCK.vhd]
```

Elaborating the VHDL Designs

You must elaborate each design individually. The elaborate command translates the intermediate design files created by analyze into .db format. Elaborating a given design makes it the current one.

In analyzing the VHDL designs for the tutorial, you specify BEHAVIOR as the architecture. (In VHDL, the default architecture is the most recently analyzed architecture.) You use the -lib parameter of the analyze command to specify the name of the library to which the work is mapped. By default, Design Compiler looks in the WORK library for the design to be built. The -lib parameter allows you to temporarily change this directory. However, for the tutorial, use the WORK directory.

To elaborate the analyzed designs in the second hierarchy level,

- Enter

```
dc_shell> elaborate ALARM_BLOCK -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate COMPARATOR -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate CONVERTOR_CKT -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate MUX -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate TIME_BLOCK -arch “BEHAVIOR” -lib WORK -update
dc_shell> elaborate ALARM_SM_2 -arch “BEHAVIOR” -lib WORK -update
```

or

```
dc_shell-t> elaborate ALARM_BLOCK -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate COMPARATOR -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate CONVERTOR_CKT -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate MUX -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate TIME_BLOCK -arch {BEHAVIOR} -lib WORK -update
dc_shell-t> elaborate ALARM_SM_2 -arch {BEHAVIOR} -lib WORK -update
```

Example Output

This example shows the output for the last pair of commands. When you elaborate an analyzed design, dc_shell generates output similar to the following output, produced by analyzing the ALARM_SM_2 design.

```
dc_shell> elaborate ALARM_SM_2 -arch “BEHAVIOR” -lib WORK -update
```

```
Inferred memory devices in process ‘SYNCH’
                 in routine ALARM_SM_2 line 32 in
                       file ‘/bohm/tutorial/vhdl/ALARM_SM_2.vhd’.
========================================================================
   Register Name | Type       | Width | Bus | MB | AR | AS | SR | SS | ST
==========================================================================
Current State    | Flip-flop  |   1   |  -  | -  | N  |  N |  N |  N | N
==========================================================================
Current design is now ‘ALARM_SM_2’
1
```

Reading In the Top-Level Design

Use analyze and elaborate to read in the top-level design.

Analyzing the Top-Level Design

To analyze the top-level VHDL design, you specify BEHAVIOR as the architecture. (In VHDL, architecture defaults to the most recently analyzed architecture.) You use the -lib parameter to specify the work library where the resulting files are to be stored and where Design Compiler looks for the designs when building them. (By default, Design Compiler looks in the WORK library for the design to be built.) The -lib parameter allows you to temporarily change this directory. However, for the tutorial, use the WORK directory.

To analyze the TOP design,

- Enter one of the following commands at the dc_shell prompt:

```
dc_shell> analyze -format vhdl -lib WORK {“./vhdl/TOP.vhd”}
```

or

```
dc_shell-t> analyze -format vhdl -lib WORK [list {./vhdl/TOP.vhd}]
```

Example Output

When you have successfully analyzed the top-level design, dc_shell generates the following output:

```
/bohm/tutorial/vhdl/TOP.vhd:
1
```

Elaborating the VHDL Designs

To elaborate the TOP design,

- Enter one of the following commands at the dc_shell prompt:

```
dc_shell> elaborate TOP -arch “BEHAVIOR” -lib WORK -update
```

or

```
dc_shell-t> elaborate TOP -arch {BEHAVIOR} -lib WORK -update
```

Example Output

When you elaborate the TOP design, dc_shell generates the following output:

```
Current design is now ‘TOP’
1
```

Setting Attributes Using the dc_shell Interface

After you read a design into Design Compiler, you specify the design environment. Specify the design environment by setting attributes that define information such as drive strengths on the ports, when signals arrive on the ports, or the load driven by the output ports.

This section describes the tasks you perform to set attributes for the Alarm Clock design example using the dc_shell command-line interface. This section includes the following:

- Setting the Drive Strength on Input Ports
- Setting the Drive Strength for CLK

- Setting the Load on Output Ports
- Setting Other Attributes at the Top Level

Setting the Drive Strength on Input Ports

Assume that all input ports except CLK have drive strength 0.08 (units are determined by the target library). Assume that the drive strength on CLK needs to be equal to the drive strength of pin Z of the driver cell B4I in the target library.

You can set port drive values equal to the drive strength of library cell output pins. When you do not know the value of the pin you want in the library, you use the drive_of command to find out and set the drive value.

Note:

> This tutorial uses set_drive to set drive strength. However, set_driving_cell offers an advantage over the set_drive command. The set_driving_cell command associates an input with a driving cell rather than a specific drive value. See the *Design Compiler Reference Manual* for more information about Design Compiler commands.

These commands set the drive strengths for the various ports by setting the rise and fall options to 0.08.

```
dc_shell> set_drive -rise .08 “ALARM”
dc_shell> set_drive -fall .08 “ALARM”
dc_shell> set_drive -rise .08 “HRS”
dc_shell> set_drive -fall .08 “HRS”
dc_shell> set_drive -rise .08 “MINS”
dc_shell> set_drive -fall .08 “MINS”
dc_shell> set_drive -rise .08 “SET_TIME”
dc_shell> set_drive -fall .08 “SET_TIME”
dc_shell> set_drive -rise .08 “TOGGLE_SWITCH”
dc_shell> set_drive -fall .08 “TOGGLE_SWITCH”
```

or

```
dc_shell-t> set_drive -rise .08 {ALARM}
dc_shell-t> set_drive -fall .08 {ALARM}
dc_shell-t> set_drive -rise .08 {HRS}
dc_shell-t> set_drive -fall .08 {HRS}
dc_shell-t> set_drive -rise .08 {MINS}
dc_shell-t> set_drive -fall .08 {MINS}
dc_shell-t> set_drive -rise .08 {SET_TIME}
dc_shell-t> set_drive -fall .08 {SET_TIME}
dc_shell-t> set_drive -rise .08 {TOGGLE_SWITCH}
dc_shell-t> set_drive -fall .08 {TOGGLE_SWITCH}
```

Note:

> If you do not specify the -rise or -fall option, both will be set. Also, you can specify all the ports sequentially as arguments to a single set_drive command.

Example Output

The following output is displayed when you set the rise drive strength on port ALARM:

```
Performing set_drive on port ‘ALARM’.
1
```

The following output is displayed when you set the fall drive strength on port ALARM:

```
Performing set_drive on port ‘ALARM’.
1
```

Setting the Drive Strength for CLK

Rather than set the drive strength to 0.08 for this exercise, you set the drive strength of CLK equal to pin Z of B4I. Assume that B4I is a balanced driver—that is, the rise and fall drive values are equal. The resulting computed rise and fall values are 0.0335.

To set the drive strength for CLK equal to pin Z of B4I,

- Enter one of the following pairs of commands at the dc_shell prompt:

  ```
  dc_shell> set_drive -rise drive_of (class/B4I/Z) “CLK”
  dc_shell> set_drive -fall drive_of (class/B4I/Z) “CLK”
  ```

  or

  ```
  dc_shell-t> set_drive -rise [drive_of {class/B4I/Z}]
  {CLK}
  dc_shell-t> set_drive -fall [drive_of {class/B4I/Z}]
  {CLK}
  ```

Example Output

When you set either the rise or fall drive strength of clock equal to pin Z of B4I, dc_shell displays the following output:

```
Performing drive_of on port ‘Z’.
Performing set_drive on port ‘CLK’
1
```

Setting the Load on Output Ports

For this exercise, you define the loads for the output ports, as discussed in the following sections:

- Setting the Load for SPEAKER_OUT
- Setting the Load on Bus Bits

- Setting the Load for AM_PM_DISPLAY

Setting the Load for SPEAKER_OUT

Load values are used to model the capacitive load on the output ports of the constrained module.

You can set port load values equal to the load values of library cell input pins. When you don’t know the library cell value you want, use the load_of command to find out and set a load value. Assume port SPEAKER_OUT drives a load of five inverters. Assume also that the inverters are the same as cell IVA (an inverter in the target library class).

To set the load for SPEAKER_OUT,

- Enter one of the following commands from the dc_shell prompt:

```
dc_shell> set_load load_of (class/IVA/A) * 5 “SPEAKER_OUT”
```

or

```
dc_shell-t> set_load [expr [load_of {class/IVA/A}] * 5] {SPEAKER_OUT}
```

Example Output

When you set the load for SPEAKER_OUT, dc_shell displays the following output:

```
Performing load_of on port ‘A’
Performing set_load on port ‘SPEAKER_OUT’
1
```

Setting the Load on Bus Bits

Assume that ports DISP1 and DISP2 each drive a load of 3 (standard loads) and AM_PM_DISPLAY drives a load of 2 (standard loads).

You can enter these commands collectively, using a script file. See Appendix A, “Tutorial Script Files.”

To set the load values on DISP1,

- Enter the following commands from the dc_shell prompt:

```
dc_shell> set_load 3.0 “DISP1[13]”
dc_shell> set_load 3.0 “DISP1[12]”
dc_shell> set_load 3.0 “DISP1[11]”
dc_shell> set_load 3.0 “DISP1[10]”
dc_shell> set_load 3.0 “DISP1[9]”
dc_shell> set_load 3.0 “DISP1[8]”
dc_shell> set_load 3.0 “DISP1[7]”
dc_shell> set_load 3.0 “DISP1[6]”
dc_shell> set_load 3.0 “DISP1[5]”
dc_shell> set_load 3.0 “DISP1[4]”
dc_shell> set_load 3.0 “DISP1[3]”
dc_shell> set_load 3.0 “DISP1[2]”
dc_shell> set_load 3.0 “DISP1[1]”
dc_shell> set_load 3.0 “DISP1[0]”
```

or

```
dc_shell-t> set_load 3.0 {DISP1[13]}
dc_shell-t> set_load 3.0 {DISP1[12]}
dc_shell-t> set_load 3.0 {DISP1[11]}
dc_shell-t> set_load 3.0 {DISP1[10]}
dc_shell-t> set_load 3.0 {DISP1[9]}
dc_shell-t> set_load 3.0 {DISP1[8]}
dc_shell-t> set_load 3.0 {DISP1[7]}
dc_shell-t> set_load 3.0 {DISP1[6]}
dc_shell-t> set_load 3.0 {DISP1[5]}
dc_shell-t> set_load 3.0 {DISP1[4]}
dc_shell-t> set_load 3.0 {DISP1[3]}
dc_shell-t> set_load 3.0 {DISP1[2]}
dc_shell-t> set_load 3.0 {DISP1[1]}
dc_shell-t> set_load 3.0 {DISP1[0]}
```

Shortcut:

Instead of setting the load values individually, you can use a wildcard. For example, you can enter set_load 3.0 “DISP1[*]” and obtain the same results.

To set the load values on DISP2,

- Enter the following commands from the dc_shell prompt, or run the script containing them (see Appendix A, “Tutorial Script Files”):

```
dc_shell> set_load 3.0 “DISP2[13]”
dc_shell> set_load 3.0 “DISP2[12]”
dc_shell> set_load 3.0 “DISP2[11]”
dc_shell> set_load 3.0 “DISP2[10]”
dc_shell> set_load 3.0 “DISP2[9]”
dc_shell> set_load 3.0 “DISP2[8]”
dc_shell> set_load 3.0 “DISP2[7]”
dc_shell> set_load 3.0 “DISP2[6]”
dc_shell> set_load 3.0 “DISP2[5]”
dc_shell> set_load 3.0 “DISP2[4]”
dc_shell> set_load 3.0 “DISP2[3]”
dc_shell> set_load 3.0 “DISP2[2]”
dc_shell> set_load 3.0 “DISP2[1]”
dc_shell> set_load 3.0 “DISP2[0]”
```

or

```
dc_shell-t> set_load 3.0 {DISP2[13]}
dc_shell-t> set_load 3.0 {DISP2[12]}
dc_shell-t> set_load 3.0 {DISP2[11]}
dc_shell-t> set_load 3.0 {DISP2[10]}
dc_shell-t> set_load 3.0 {DISP2[9]}
dc_shell-t> set_load 3.0 {DISP2[8]}
dc_shell-t> set_load 3.0 {DISP2[7]}
dc_shell-t> set_load 3.0 {DISP2[6]}
dc_shell-t> set_load 3.0 {DISP2[5]}
dc_shell-t> set_load 3.0 {DISP2[4]}
dc_shell-t> set_load 3.0 {DISP2[3]}
dc_shell-t> set_load 3.0 {DISP2[2]}
```

```
   dc_shell-t> set_load 3.0 {DISP2[1]}
   dc_shell-t> set_load 3.0 {DISP2[0]}
```

Example Output

```
Performing set_load on port ‘DISP1[13]’.
1
```

Setting the Load for AM_PM_DISPLAY

Set the load on the remaining output port to 2.0.

To set the capacitive load for AM_PM_DISPLAY,

- Enter one of the following commands from the dc_shell prompt:

  ```
  dc_shell> set_load 2.0 “AM_PM_DISPLAY”
  ```

  or

  ```
  dc_shell-t> set_load 2.0 {AM_PM_DISPLAY}
  ```

Example Output

When you set the load on the AM_PM_DISPLAY output port, dc_shell displays the following output:

```
Performing set_load on port ‘AM_PM_DISPLAY’.
1
```

Setting Other Attributes at the Top Level

Some attributes are set at the top level of a design but are not associated with a particular input or output port. Such attributes can have a global effect on optimization of the design and the hierarchical blocks the design contains.

For this exercise, set attributes on the TOP design as discussed in the following sections:

- Setting the Wire Load for the Design.
- Setting the Operating Conditions for the Design.

Setting the Wire Load for the Design

Design Compiler optimization uses net fanout as a basis for estimating interconnect wire length from the wire load model. Design Compiler uses this information to calculate interconnect wiring and transition delays.

The wire load model for a design depends on the estimated die size of the design. Wire load models are defined in the target library.

Design Compiler uses area as a basis for automatically selecting the wire load tables if your ASIC libraries support this feature.

To set the wire load,

- Enter one of the following commands at the dc_shell prompt:

  ```
  dc_shell> set_wire_load “10x10” -library “class”
  ```

  or

  ```
  dc_shell-t> set_wire_load {10x10} -library {class}
  ```

Example Output

```
Using wire_load model ‘10x10’ found in library ‘class’.
```

Setting the Operating Conditions for the Design

Operating conditions are the temperature, process, and voltage in which the design operates. The target library defines the operating conditions. Library vendors define default operating conditions, which can differ from one vendor to another. Common default operating conditions are

- Temperature – 25° C
- Process – 1
- Voltage – 5

The Design Compiler static timing analyzer models the effects of variation in the drive strength, arrival time, and load values on a circuit's timing characteristics. In a similar way, you can analyze a design for best-case, nominal-case, and worst-case performance or operating conditions.

To set the operating conditions,

- Enter one of the following commands from the dc_shell prompt:

```
dc_shell> set_operating_conditions -library “class” “WCCOM”
```

or

```
dc_shell-t> set_operating_conditions -library {class} {WCCOM}
```

Example Output

When you set the operating conditions for the design, dc_shell displays the following confirmation output:

```
Using operating conditions ‘WCCOM’ found in library ‘class’.
```

Saving the Design Using dc_shell

After you set attributes on the design, save the design to preserve the settings.

To save the design,

- Enter one of the following commands at the dc_shell prompt:

```
dc_shell> write -format db -hierarchy -output “./db/
TOP_attributes.db” {“TOP.db:TOP“}
```

or

```
dc_shell-t> write -format db -hierarchy -output {./db/TOP_attributes.db} [list
{TOP.db:TOP}]
```

In response, dc_shell displays the following confirmation message:

```
Writing to file /bohm/tutorial/db/TOP_attributes.db
```

Then quit dc_shell, or go on to the next chapter.

To quit dc_shell,

- At the command line prompt, type quit or exit.

8

Defining Optimization Goals and Setting Constraints

During optimization of a design, Design Compiler algorithms assess how best to implement the design. You direct Design Compiler decisions by defining optimization goals before you optimize. Your optimization goals are called constraints.

Constraints are measurable circuit characteristics for timing, area, and power that you set on a design. Design Compiler checks your constraint goals during optimization and tries to meet them while synthesizing the design to your technology library.

Your technology library contains important specifications of timing, area, and power. During optimization, Design Compiler constructs complex models and makes detailed calculations, using specifications in the technology library and your design constraints. For accurate results, define constraint values that are as realistic as possible.

You can also use constraints to specify internal design timing, logical and electrical connections of ports, and subdesign interfaces.

Allow about one hour to complete the exercises in this chapter.

This chapter includes the following sections:

- Before You Begin

  Explains the preliminary tasks you perform before undertaking the exercises you use to set the optimization goals. Preliminary tasks are provided for both interfaces—Design Analyzer and dc_shell.

- Removing Existing Constraints

  Explains, for both interfaces, how to remove existing constraints on the design, if any, in order to prepare for setting them anew.

- About Setting Design Constraints

  Explains what design constraints are and how to assess realistic goals for a design prior to setting constraints for it, in order to get the best results from Design Compiler.

- Using Design Analyzer to Set Design Constraints

  Contains the complete set of tasks and exercises you perform to set the design constraints for optimizing the sample design. This section begins with an explanation of how to remove existing constraints. Then it addresses how to set the various design constraints, check the design, and resolve multiple instances of a design created through multiple references to it. This section concludes with instructions about how to save your design.

  For a list of the exercises, see the introduction to the section.

- Using dc_shell to Set Design Constraints

  Contains the complete set of tasks and exercises you perform to set the design constraints for optimizing the sample design. The dc_shell commands are given in both dcsh and dctcl modes. This section begins with an explanation of how to remove existing constraints. Then it addresses how to set the various design constraints, check the design, and resolve multiple instances of a design created through multiple references to it. This section concludes with instructions about how to save your design.

  For a list of the exercises, see the introduction to the section.

Before You Begin

For each interface, this section explains what you need to do before you begin the exercises described in this chapter.

For either interface, before you begin, complete the exercises in Chapter 7, “Setting the Design Environment.”

Design Analyzer Preliminaries

For Design Analyzer, complete these preliminary tasks:

1. If Design Analyzer is not running, invoke it from your tutorial directory. See “Starting Design Analyzer” in Chapter 2.
2. Read in TOP_attributes.db—which you created by following the steps in Chapter 7, “Setting the Design Environment”—from your tutorial/db/ directory, using File > Read.
3. Generate the Symbol view for TOP, as described in “About Setting Attributes for the TOP Design” in Chapter 7.

The dc_shell Command-Line Interface Preliminaries

For dc_shell, complete these preliminary tasks:

1. If dc_shell is not running, invoke it from your tutorial directory. See "About Reading In a Hierarchical Design" in Chapter 7 and "Starting dc_shell" in Chapter 4. You can use either the dcsh or dctcl mode.
2. Read in TOP_attributes.db (which you create by following the steps in Chapter 7, "Setting the Design Environment") from your tutorial/db/ directory, using the following command:

   ```
   dc_shell> read -format db {“./db/TOP_attributes.db”}
   ```

   or

   ```
   dc_shell-t> read_file -format db [list {./db/
   TOP_attributes.db}]
   ```

Removing Existing Constraints

In the tutorial you have not yet defined constraints on TOP. Before you do, it is good practice to remove any existing constraints.

To remove existing constraints,

- Enter the following dc_shell command in the Design Analyzer Command Window text field or at the dc_shell command prompt (dcsh or dctcl mode):

  ```
  remove_constraint -all
  ```

The remove_constraint command does not remove the attributes set as part of the design environment in Chapter 7, “Setting the Design Environment.”

About Setting Design Constraints

Constraints are your performance goals for a design and often define maximum area and timing goals, such as maximum or minimum delay, and clock specifications. However, constraints can also define other requirements, such as maximum allowable power or how easily a design must be routed (porosity).

For details on all constraints, see the *Design Compiler Reference Manual: Constraints and Timing.*

To get the best results from Design Compiler, set your constraints to values that are close to your design goals. If timing goals are set unrealistically low (for example, 0), optimization adds buffers to critical paths or duplicates logic on heavily loaded nets. Trying to meet an unrealistic timing goal can result in a significant increase in area. Use realistic timing goals so optimization produces the smallest circuit that most closely satisfies the constraint goal.

If you don’t set a timing goal, the Design Compiler default is to apply only design rule constraints to the design during optimization. Design rule constraints are requirements imposed by the technology to which you map your design.

Determining Realistic Goals

Usually the clock period, timing numbers, and area of the circuit are provided in the design specification. In such a case, goals are clearly defined at the outset.

However, goals might not be available in a design specification, or perhaps only a subdesign needs to be optimized. When optimizing a subdesign, you might know goals for the entire design but not for the subdesign.

When realistic goals are unknown, map the design or subdesign to gates without setting constraints. Mapping without constraints can help determine the current design speed. Use this design speed to determine a starting point value for constraints. As you recompile the design, you can refine your constraints.

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

For the remainder of this chapter, assume that the constraints you set in the exercises are realistic for the design.

About Defining Goals for Design TOP

The clock for design TOP has a period of 1 second. If you define a clock period of 1 second as a design goal, Design Compiler easily meets the goal because the delay times for library cells are in nanoseconds.

To set a more restrictive time constraint, assume that the alarm clock should operate at a period of 25 ns, or 40 MHz. In addition, the signal should reach the output ports before the next clock cycle (before 25 ns). To ensure that this goal is met, constrain the output ports to require the signals to reach these ports before 20 ns.

Suppose the design area must not exceed 1100 gate equivalents. You do not, however, need to set an area constraint. Optimization produces the smallest circuit that meets the timing constraints.

Using Design Analyzer to Set Design Constraints

This section contains a description of the exercises in the following sections:

- Setting Clock Constraints Using Design Analyzer

  Describes a clock and explains that for combinational modules not fed by clocks, you must create virtual clocks in the design. Provides steps for assigning clock-related values (constraints and attributes) to the clock object of the Alarm Clock design, using Design Analyzer.

- Setting Delay Constraints Using Design Analyzer

  Describes what input delays and output delays are and explains how to assess constraints to define them for a design. Provides steps for setting delays constraints on output ports, using Design Analyzer.

- Checking the Design, and Exploring and Correcting Errors, Using Design Analyzer

Presents a set of exercises you undertake to check the design for problems, display areas warned about, display the pin names layer, and explore other error messages.

- Resolving Multiple Design Instances Using Design Analyzer

  Explains how multiple instances of subdesigns can occur through multiple references to the same subdesign. Also discusses different approaches to resolving the problem.

- Saving the Design Using Design Analyzer

  Explains how to save the design, using Design Analyzer before you quit or go on to the next chapter.

Setting Clock Constraints Using Design Analyzer

Sequential circuits always have clocks. Constrain sequential designs by defining input and output delays relative to a clock.

A combinational module is not fed by a clock. Therefore, you must create virtual clocks in the design before setting input or output delays. To constrain your tutorial design, create a virtual clock before defining input and output delays.

A clock has a source that can be an input port of the design or the output pin of a component in the design. A clock object can be attached to a clock source. For the Alarm Clock design example, the clock source is port CLK.

In this exercise, you assign clock-related values (constraints and attributes) to the clock object. Clock objects appear in the Schematic view as a small waveform symbol next to the source to which they are attached.

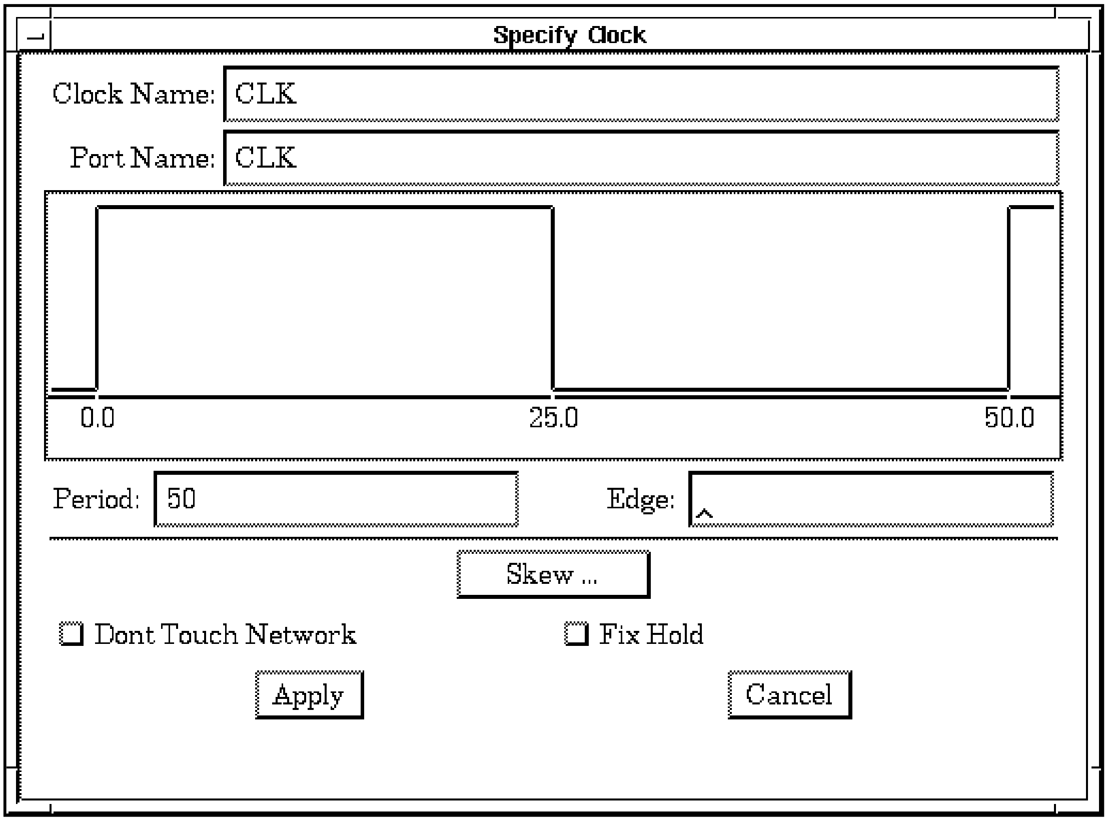
Before you perform this task, read “About Setting Design Constraints” on page 8-5.

To create the clock object for CLK,

1. Select CLK in the Symbol view.
2. Choose Attributes > Clocks > Specify.

   The Specify Clock window, as shown in Figure 8-1, opens.

Figure 8-1 Specify Clock Window



Clock Name: CLK

Port Name: CLK

Period: 50

Edge:

□ Dont Touch Network

□ Fix Hold

3. Type 25 in the Period field.

When the value in this field is applied, a setup constraint is set on each D pin of the flip-flops driven by CLK.

The Fix Hold option causes Design Compiler to fix violated hold times. Usually you request that Design Compiler first fix setup time violations. Then, after compilation, you check for hold-time violations. If hold-time violations exist after compilation, you recompile the design with Fix Hold selected.

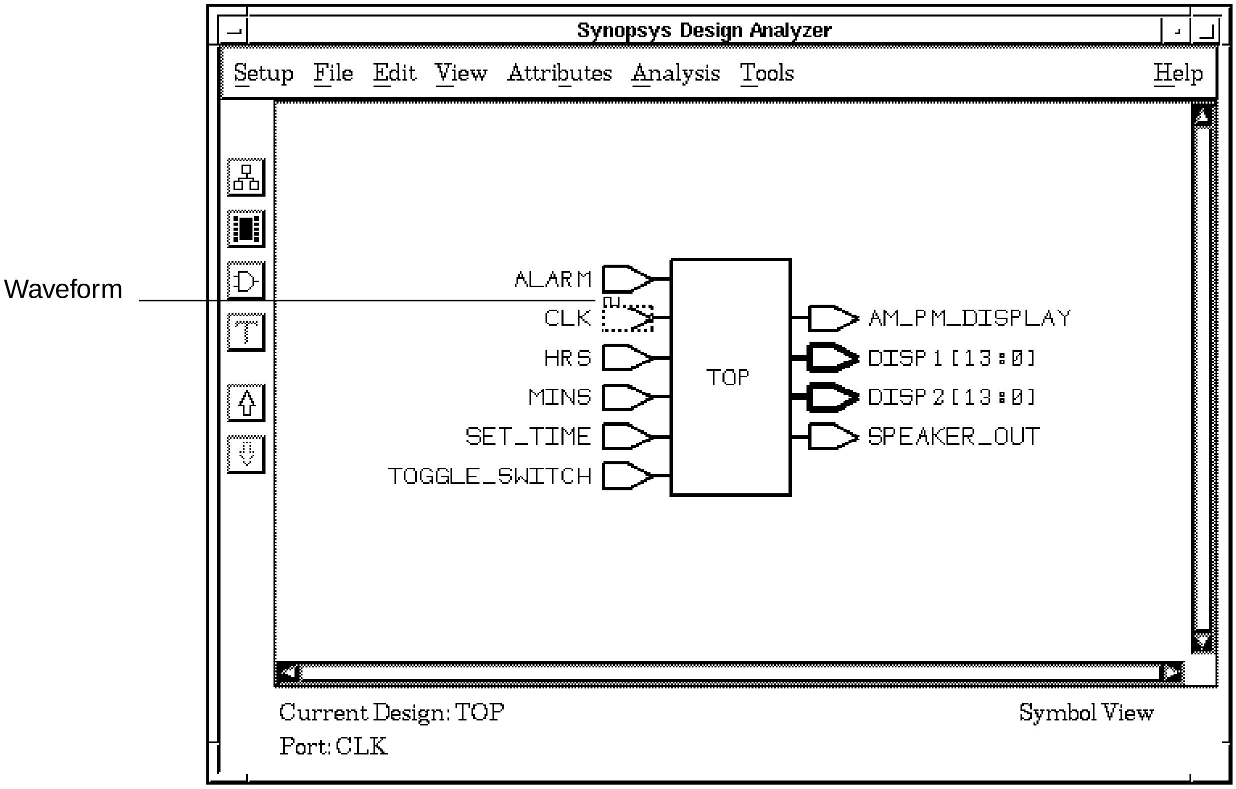
The Fix Hold option is not needed in this exercise.

4. Click Apply.
5. Click Cancel to close the Specify Clock window.

   The clock constraint is set. Note that the values in the Specify Clock window are not changed.

   The clock object is created on CLK. Note the small waveform symbol attached to the CLK port. See Figure 8-2.

Figure 8-2 Clock Object



Setting Delay Constraints Using Design Analyzer

Input delays model the external delays arriving at the input ports of the constrained module. Input delays are defined relative to a real or virtual clock.

Output delays model the external delays leaving the output ports of the constrained module. Output delays must be defined relative to a real or virtual clock to act as a path constraint. The output delay corresponds to the time before the next rising edge.

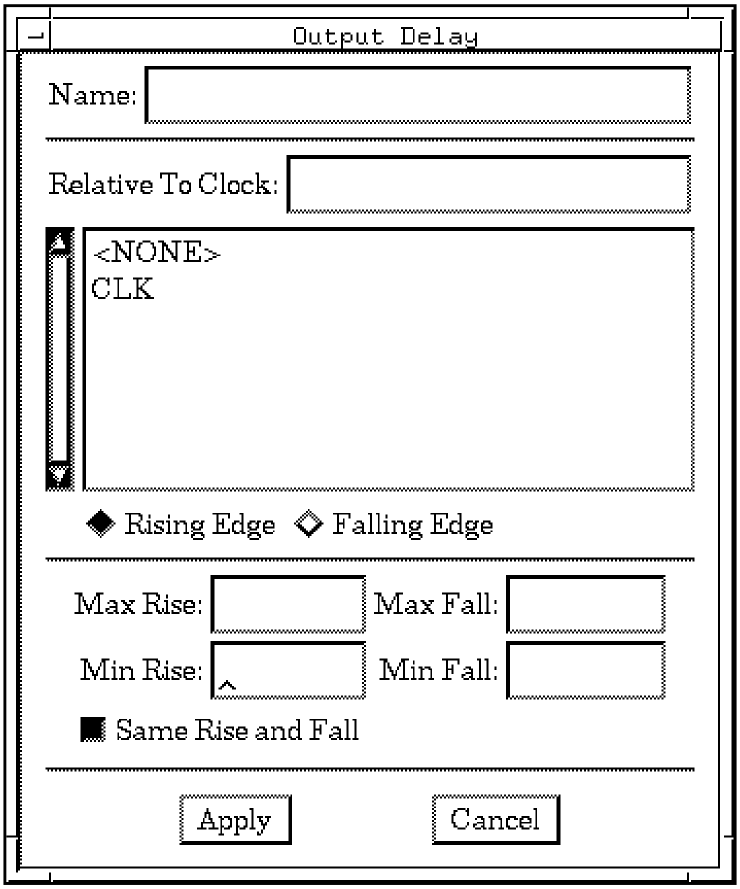
The period of the clock is 25 ns, and the outputs are required to arrive at 20 ns. This timing relationship defines an output delay of 5 ns for all output ports. Thus, the delay constraint on the output ports is 5 ns.

To set the delay constraints on the output ports,

1. Select the four output ports of TOP.
2. Choose Attributes > Operating Environment > Output Delay.

   The Output Delay window, as shown in Figure 8-3, opens.

Figure 8-3 Output Delay Window



<NONE>
CLK

◆ Rising Edge ◇ Falling Edge

■ Same Rise and Fall

3. Select CLK.

CLK becomes highlighted, and the Relative To Clock field displays the CLK value.

4. Enter 5 in the Max Rise and Max Fall fields.

5. Click Apply.

6. Click Cancel to close the Output Delay window.

   Design Analyzer sets the constraints on the output ports and displays this message:

   ```
   Output delay attributes set on selected paths
   ```

Checking the Design, and Exploring and Correcting Errors, Using Design Analyzer

After you set constraints on a design and before you compile the design, check the design to identify and correct any problems. This process checks the internal representation of the design for correctness and issues appropriate warnings or errors.

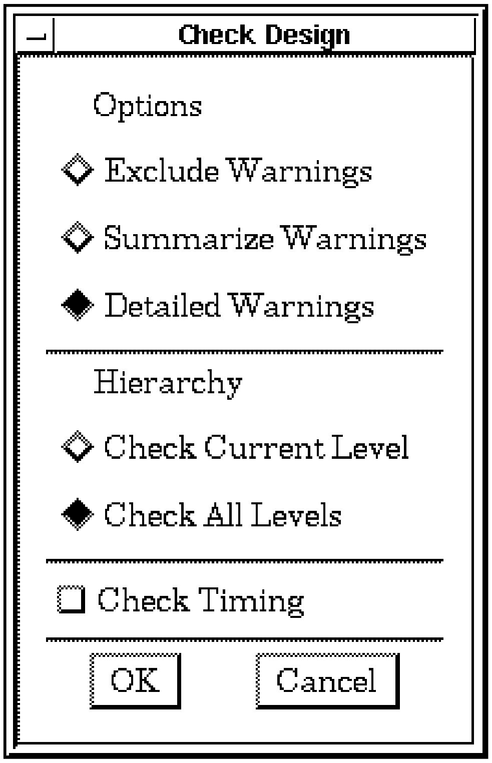
Check the Design

To check the design,

1. Choose Analysis > Check Design.

   The Check Design window, shown in Figure 8-4, appears.

Figure 8-4   Check Design Window



Check Design

Options

◇ Exclude Warnings

◇ Summarize Warnings

◆ Detailed Warnings

Hierarchy

◇ Check Current Level

◆ Check All Levels

□ Check Timing

By default, Detailed Warnings and Check All Levels are selected. These options provide explicit warning and error information and check all subdesigns in the hierarchy.

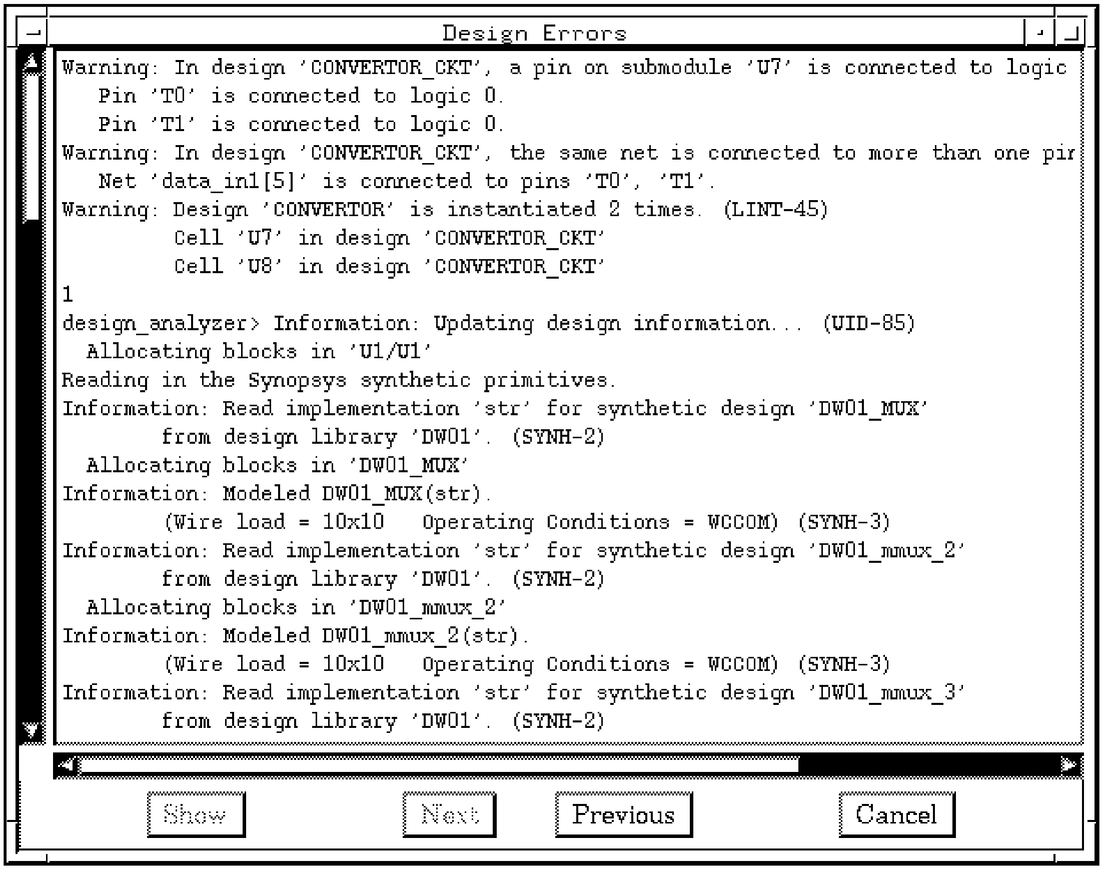
2. Click Check Timing to select it.

   Check Timing checks the timing attributes placed on the design.

3. Click OK.

   The Design Errors window, shown in Figure 8-5, appears.

Figure 8-5   Design Errors Window



```
Warning: In design 'CONVERTOR_CKT', a pin on submodule 'U7' is connected to logic
   Pin 'T0' is connected to logic 0.
   Pin 'T1' is connected to logic 0.
Warning: In design 'CONVERTOR_CKT', the same net is connected to more than one pir
   Net 'data_in1[5]' is connected to pins 'T0', 'T1'.
Warning: Design 'CONVERTOR' is instantiated 2 times. (LINT-45)
         Cell 'U7' in design 'CONVERTOR_CKT'
         Cell 'U8' in design 'CONVERTOR_CKT'
1
design_analyzer> Information: Updating design information... (UID-85)
  Allocating blocks in 'U1/U1'
Reading in the Synopsys synthetic primitives.
Information: Read implementation 'str' for synthetic design 'DW01_MUX'
        from design library 'DW01'. (SYNH-2)
  Allocating blocks in 'DW01_MUX'
Information: Modeled DW01_MUX(str).
        (Wire load = 10x10   Operating Conditions = WCCOM) (SYNH-3)
Information: Read implementation 'str' for synthetic design 'DW01_mmux_2'
        from design library 'DW01'. (SYNH-2)
  Allocating blocks in 'DW01_mmux_2'
Information: Modeled DW01_mmux_2(str).
        (Wire load = 10x10   Operating Conditions = WCCOM) (SYNH-3)
Information: Read implementation 'str' for synthetic design 'DW01_mmux_3'
        from design library 'DW01'. (SYNH-2)
```

Warning messages are issued for CONVERTOR_CKT and CONVERTOR.

To read an entire warning message, use the scroll bar at the bottom of the window or drag the window border to make it larger.

Display the Area That Pertains to the Warning

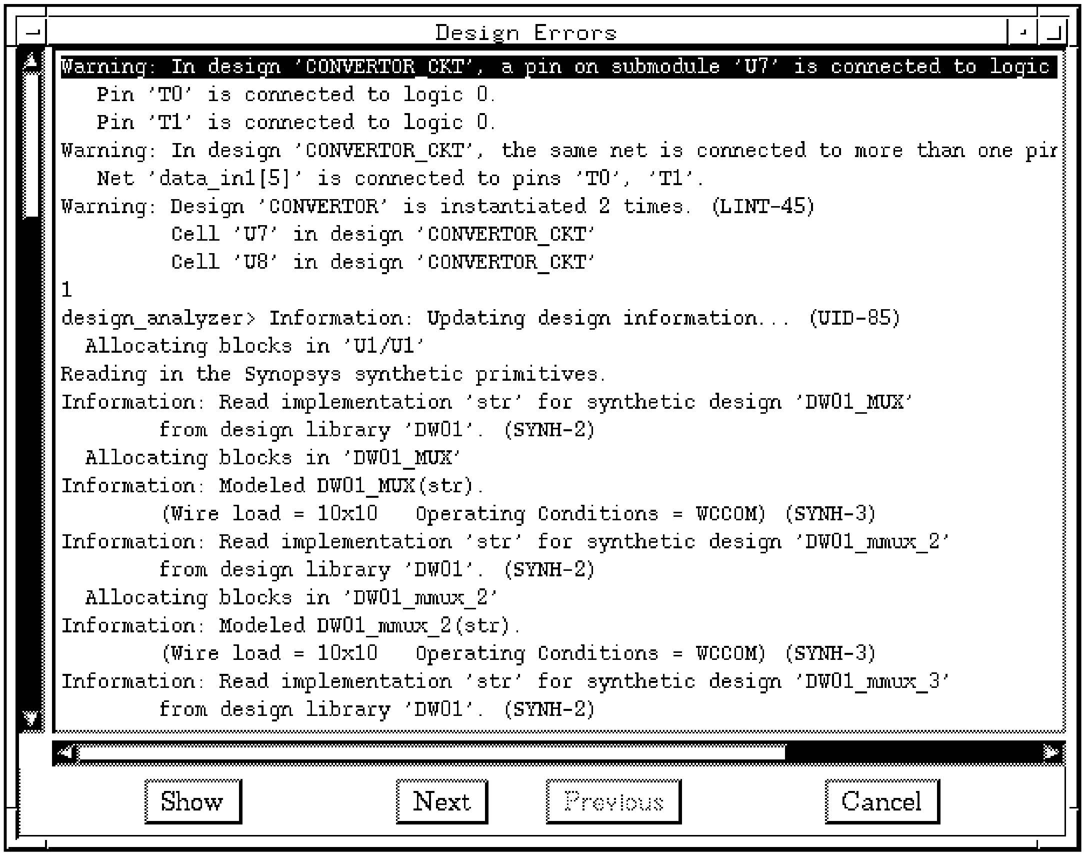
To display the area for which warnings are issued,

1. Move the Design Errors window beside the Design Analyzer window so that both windows are fully visible.

2. Select the first warning message in the Design Errors window shown in Figure 8-6:

   ```
   Warning: In design ‘CONVERTOR_CKT’, a pin submodule ‘U7’
   is connected to logic 1 or logic 0. (LINT-32)
   ```

Figure 8-6 Warning Message in Design Errors Window



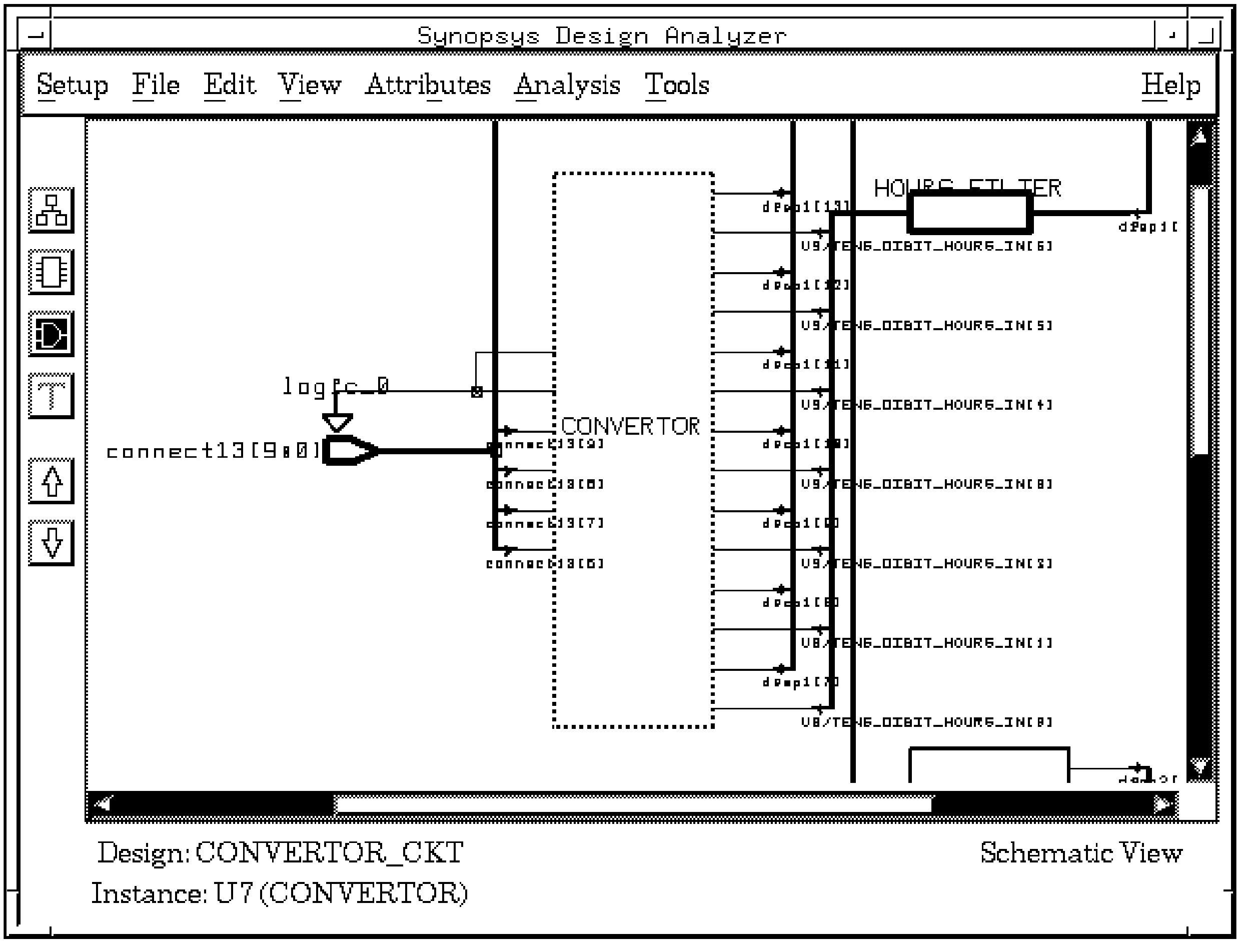
```
Warning: In design 'CONVERTOR_CKT', a pin on submodule 'U7' is connected to logic
   Pin 'T0' is connected to logic 0.
   Pin 'T1' is connected to logic 0.
Warning: In design 'CONVERTOR_CKT', the same net is connected to more than one pir
   Net 'data_in1[5]' is connected to pins 'T0', 'T1'.
Warning: Design 'CONVERTOR' is instantiated 2 times. (LINT-45)
         Cell 'U7' in design 'CONVERTOR_CKT'
         Cell 'U8' in design 'CONVERTOR_CKT'
1
design_analyzer> Information: Updating design information... (UID-85)
  Allocating blocks in 'U1/U1'
Reading in the Synopsys synthetic primitives.
Information: Read implementation 'str' for synthetic design 'DW01_MUX'
        from design library 'DW01'. (SYNH-2)
  Allocating blocks in 'DW01_MUX'
Information: Modeled DW01_MUX(str).
        (Wire load = 10x10   Operating Conditions = WCCOM) (SYNH-3)
Information: Read implementation 'str' for synthetic design 'DW01_mmux_2'
        from design library 'DW01'. (SYNH-2)
  Allocating blocks in 'DW01_mmux_2'
Information: Modeled DW01_mmux_2(str).
        (Wire load = 10x10   Operating Conditions = WCCOM) (SYNH-3)
Information: Read implementation 'str' for synthetic design 'DW01_mmux_3'
        from design library 'DW01'. (SYNH-2)
```

The warning message is highlighted, and the Show and Next buttons are enabled.

3. Click Show.

   Design Analyzer displays a close-up view of the design areas related to the warning message.

Figure 8-7   Close-Up of Design Areas in Design Analyzer Window


Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools
Help
CONVERTOR
logic_0
connect13[9:0]
HOURS_FILTER
U9/TENS_DIGIT_HOURS_IN[6]
U9/TENS_DIGIT_HOURS_IN[5]
U9/TENS_DIGIT_HOURS_IN[4]
U9/TENS_DIGIT_HOURS_IN[3]
U9/TENS_DIGIT_HOURS_IN[2]
U8/TENS_DIGIT_HOURS_IN[1]
U8/TENS_DIGIT_HOURS_IN[0]

Design: CONVERTOR_CKT

Instance: U7 (CONVERTOR)

Schematic View

CONVERTOR is selected in the Schematic view and the CONVERTOR’s name appears in the message area.

CONVERTOR has instance name U7 and is a submodule of CONVERTOR_CKT. (CONVERTOR is instantiated by CONVERTOR_CKT.)

Examine the Additional Messages

To examine the two additional messages issued for the design example,

1. Click Next in the Design Errors window to highlight the next message:

   ```
   Pin ‘T0’ is connected to logic 0.
   ```

   Pin T0 is selected in the schematic. Design Analyzer updates the schematic in the Design Analyzer window.

2. Click Next to highlight the next message:

   ```
   Pin ‘T1’ is connected to logic 0.
   ```

   Pin ‘T1’ is selected in the schematic.

3. Zoom in on the area that contains pins T0 and T1.

The pin names for T0 and T1 are not displayed in the schematic because pin names are turned off by default. Schematics are composed of transparent layers, each layer containing different information. A layer of information is visible only if it is turned on.

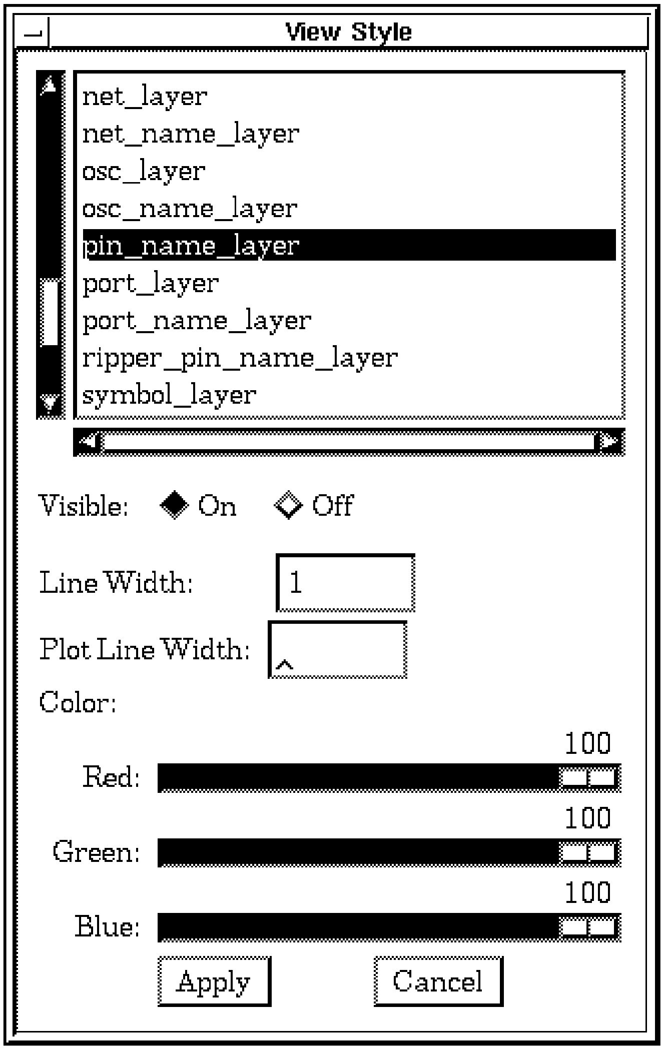
Display the Pin Names Layer

To display the pin names layer,

1. Choose View > Style.

   The View Style window, as shown in Figure 8-8, appears.

Figure 8-8   View Style Window



View Style

net_layer
net_name_layer
osc_layer
osc_name_layer
pin_name_layer
port_layer
port_name_layer
ripper_pin_name_layer
symbol_layer

Visible: ◆ On ◇ Off

Line Width: 1

Plot Line Width:

Color:

Red: 100

Green: 100

Blue: 100

Browse the list of layers. Each layer name describes the text or graphic it displays. Use the View Style window to review layer settings and set additional layers.

2. Scroll down the list to pin_name_layer, and select it.

   The Visible option for this layer is set to Off.

3. Click the Visible option On, and then click Apply.

The schematic displays pin names.

4. Click Cancel to close the View Style window.

Examine the Remaining Errors

To continue viewing errors,

1. Click Next in the Design Errors window.

   This message is related to the three messages you examined earlier:

```
Warning: In design ‘CONVERTOR_CKT’, the same net is connected to more than one
pin on submodule ‘U7’. (LINT-33)
Net ‘data_in1[5]’ is connected to pins ‘T0’, ‘T1’.
```

   This message and the previous three warning messages are issued because unconnected input ports are automatically connected to logic 0 by Design Compiler. Because these warnings do not reflect problems in the design, you can ignore them.

   If you’re using Verilog, the net name Logic0 is connected to T0 and T1.

2. Click Next in the Design Errors window.

```
Warning: Design ‘CONVERTOR’ is instantiated 2 times. (LINT-45)
Cell ‘U7’ in design ‘CONVERTOR_CKT’
Cell ‘U8’ in design ‘CONVERTOR_CKT’
```

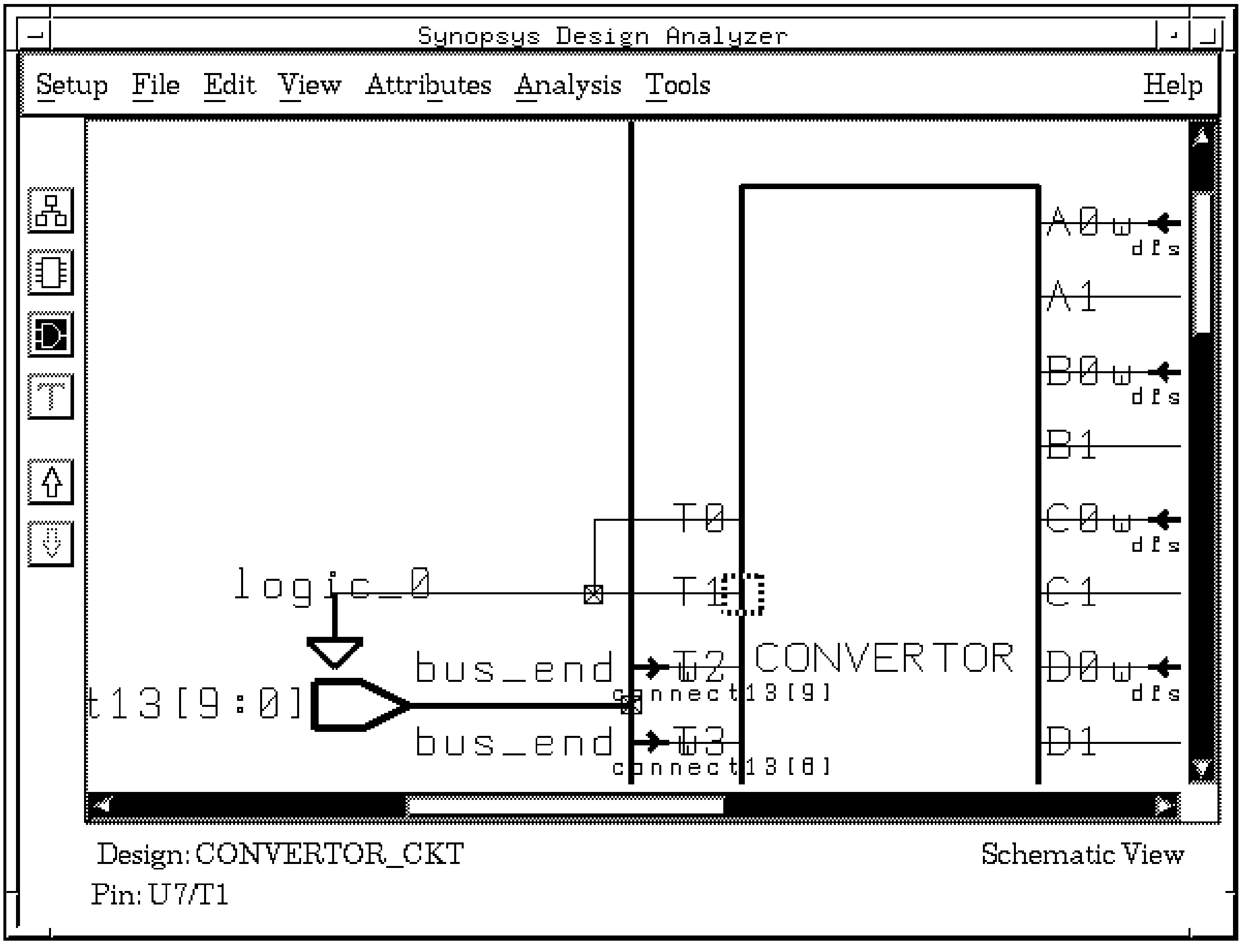
   This message is issued because CONVERTOR is referenced more than once in design CONVERTOR_CKT. Resolve the multiple instances of CONVERTOR before optimization, as shown in the next section, “Resolving Multiple Design Instances Using Design Analyzer.”

3. Click Cancel to close the Design Errors window.

Explore an Optional Exercise

Cell names U7 and U8 (instances of CONVERTOR) are not visible in the schematic. The cell_name_layer is Off by default. Turn the cell names on, and display them in the schematic, as shown in Figure 8-9.

Figure 8-9   Cell Names in the Design Analyzer Window



Design: CONVERTOR_CKT
Schematic View
Pin: U7/T1

Resolving Multiple Design Instances Using Design Analyzer

Hierarchical designs can reference a subdesign more than once. When the same subdesign is referenced more than once in a design, multiple instances exist in the design. Resolve multiple design instances before you compile.

You can resolve multiple instances in three ways:

- Use the compile-once-don’t-touch method (set_dont_touch command).

  This method preserves the subdesign during optimization of the design by setting the dont_touch attribute on cells or references.

- Remove the hierarchy (ungroup command).

  This method removes the hierarchy and generates designs with unique names for all cells and references in the current design hierarchy.

- Make each instance unique by making a copy for each instance (uniquify command).

  This method generates designs with unique names for all cells and references in the current design hierarchy.

This exercise uses the third method of issuing the uniquify command to resolve multiple instances. The uniquify multiple command creates one copy of the subdesign each time it is referenced in the design and assigns a unique design name to the copy. References to the subdesign are updated to reflect the new names wherever they occur in the hierarchy. During compilation, the Design Compiler optimization maps each instance to its unique environment.

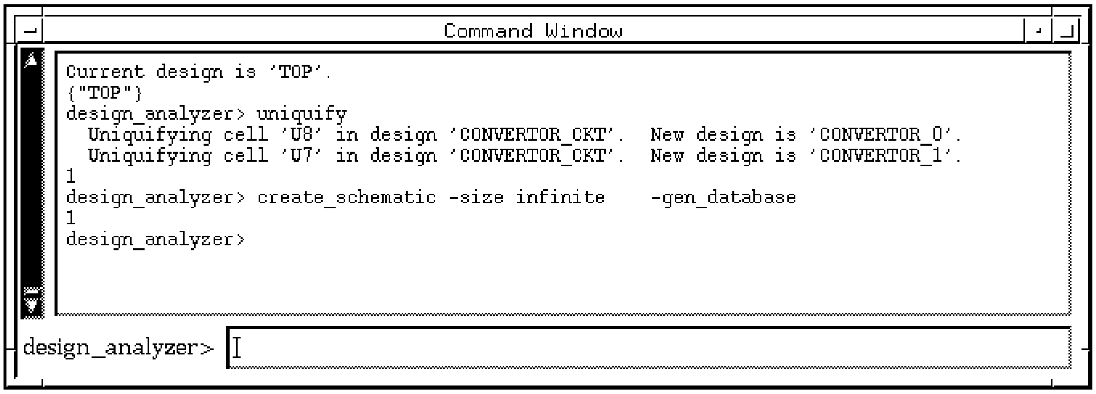
For the Alarm Clock design, the uniquify option is the most effective method because each instance of CONVERTOR needs to be optimized with respect to its environment and the hierarchy for the alarm clock needs to be preserved. The set_dont_touch and ungroup commands appear in “Using Alternatives to uniquify in Design Analyzer” in Chapter 9.

To resolve multiple references to a subdesign,

1. Click the Up Arrow twice in the Schematic view of CONVERTOR_CKT.
2. Select the Designs view, and then select TOP design.
3. Choose Edit > Uniquify > Hierarchy.

   The uniquify command runs for TOP. The results appear in the Command Window, as shown in Figure 8-10.

Figure 8-10 Command Window Showing Uniquify Command Results

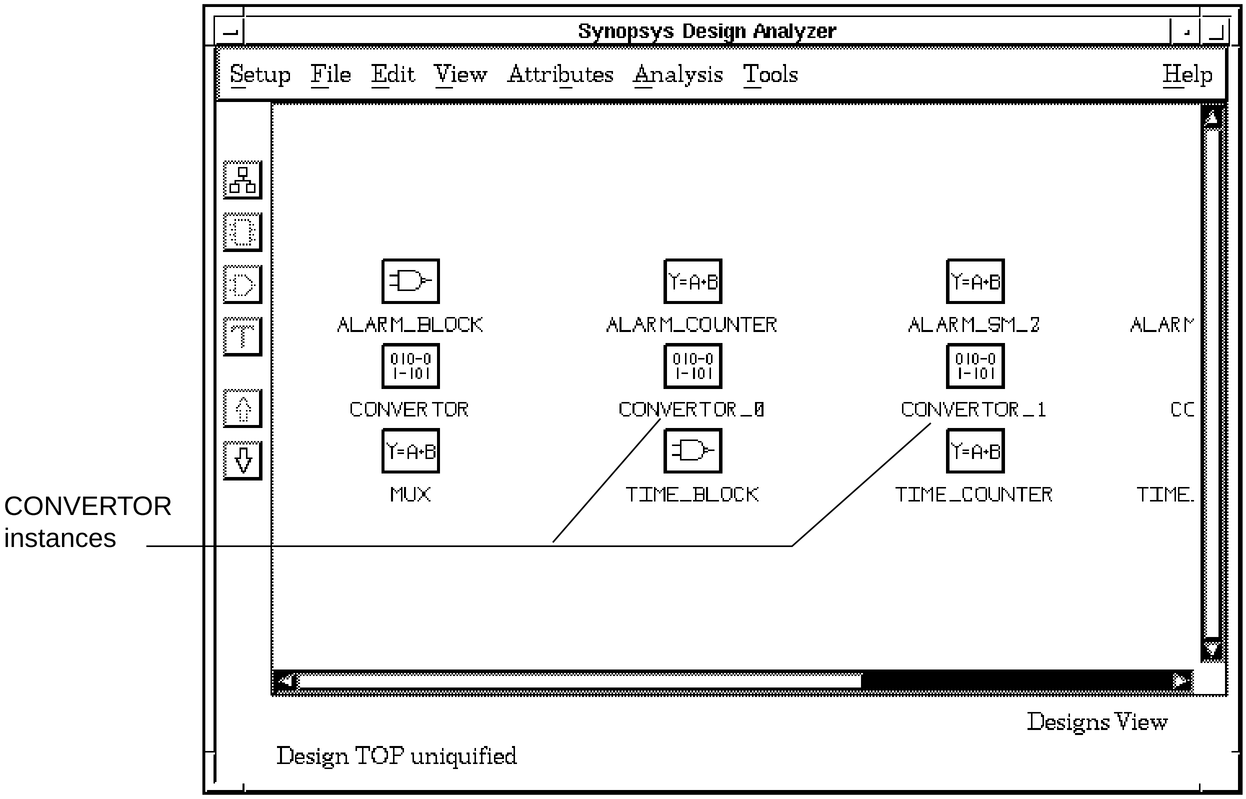


```
Current design is 'TOP'.
{"TOP"}
design_analyzer> uniquify
  Uniquifying cell 'U8' in design 'CONVERTOR_CKT'.  New design is 'CONVERTOR_0'.
  Uniquifying cell 'U7' in design 'CONVERTOR_CKT'.  New design is 'CONVERTOR_1'.
1
design_analyzer> create_schematic -size infinite    -gen_database
1
design_analyzer>
```

In the Designs view, the message “Design TOP uniquified” and two new PLA icons appear—CONVERTOR_0 and CONVERTOR_1—which correspond to the two instances of CONVERTOR in TOP. See Figure 8-11.

During optimization, CONVERTOR_0 and CONVERTOR_1 are optimized in the context of their different environments.

Figure 8-11 Design Analyzer WIndow Showing Two CONVERTOR Instances



Saving the Design Using Design Analyzer

Use this version of TOP as the starting point for the exercises in the next chapter.

To save the design file,

1. Select design TOP if it is not already selected.
2. Choose File > Save As.
3. Change to the db directory, if you are not in it.

4. Enter TOP_before_compile.db in the File Name field.
5. Click the “Save All Designs in Hierarchy” option to select it.
6. Click OK.

Then quit Design Analyzer, or go on to the next chapter.

To quit Design Analyzer,

- Enter quit in the Design Analyzer Command Window text field

  or

  Choose File > Quit.

Using dc_shell to Set Design Constraints

This section contains a description of the exercises in the following sections:

- Setting Clock Constraints Using dc_shell

  Describes a clock and explains that for combinational modules not fed by clocks, you must create virtual clocks in the design.

  Explains how to assign clock-related values (constraints and attributes) to the clock object of the Alarm Clock design, using dc_shell commands.

- Setting Delay Constraints Using dc_shell

Describes what input delays and output delays are and explains how to assess constraints to define them for a design. Specifies the dc_shell commands you use to set delay constraints and tells you where to find the tutorial script file of these commands, which you can run instead.

- Checking the Design Using dc_shell

  Explains how to run the check_design command to identify and correct any problems after you set constraints on a design and before you compile it.

- Resolving Multiple Design Instances Using dc_shell

  Explains how multiple instances of subdesigns can occur through multiple references to the same subdesign. Also discusses different approaches to resolving the problem, and gives commands for resolving multiple instances of a subdesign, using the uniquify command in dc_shell.

- Saving the Design Using dc_shell

  Explains how to save the design in dc_shell before you quit or go on to the next chapter.

Setting Clock Constraints Using dc_shell

Sequential circuits always have clocks. Constrain sequential designs by defining input and output delays relative to a clock.

A combinational module is not fed by a clock. You must create virtual clocks in the design before setting input or output delays. To constrain your tutorial design, create a virtual clock before defining input and output delays.

A clock has a source that can be an input port of the design or the output pin of a component in the design. A clock object can be attached to a clock source. For the Alarm Clock design example, the clock source is port CLK.

In this exercise, you assign clock-related values (constraints and attributes) to the clock object. Before you perform this task, read “About Setting Design Constraints” on page 8-5.

To create a clock object and then set a clock constraint, enter one of the following commands at the dc_shell prompt:

```
dc_shell> create_clock -name CLK -period 25 -waveform { 0 12.5 } { CLK }
```

or

```
dc_shell-t> create_clock -name CLK -period 25 -waveform [list 0 12.5 ] [list CLK ]
```

The -name option specifies the name of the clock, -period gives the period of the clock waveform in library time units, and -waveform gives the rise and fall edge times of the clock over an entire clock period in library time units.

In response, dc_shell displays the following output:

```
Performing create_clock on port, ’CLK’.
```

Setting Delay Constraints Using dc_shell

Input delays model the external delays arriving at the input ports of the constrained module. Input delays are defined relative to a real or virtual clock.

Output delays model the external delays leaving the output ports of the constrained module. Output delays must be defined relative to a real or virtual clock to act as a path constraint. The output delay corresponds to the time before the next rising edge.

The period of the clock is 25 ns, and the outputs are required to arrive at 20 ns. This timing relationship defines an output delay of 5 ns for all output ports. Thus, the delay constraint on the output ports is 5 ns.

These are the commands you use to set delay constraints on the output ports for the Alarm Clock design example.

Instead of entering these commands singly, you can use the optgoals.script or optgoals.tcl script file of these commands to set delay constraints on the output ports for this exercise. All the Design Compiler Tutorial scripts are available in the tutorial/appendix_A directory and are listed in Appendix A, “Tutorial Script Files.”

```
dc_shell> set_output_delay -clock CLK -max -rise 5 “SPEAKER_OUT”
dc_shell> set_output_delay -clock CLK -max -fall 5 “SPEAKER_OUT”
dc_shell> set_output_delay -clock CLK -max -rise 5 “AM_PM_DISPLAY”
dc_shell> set_output_delay -clock CLK -max -fall 5 “AM_PM_DISPLAY”
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP2[13]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP2[13]”
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP2[12]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP2[12]”
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP2[11]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP2[11]”
.
.
.
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP2[0]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP2[0]”
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP1[13]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP1[13]”
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP1[12]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP1[12]”
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP1[11]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP1[11]”
.
.
```

```
.
dc_shell> set_output_delay -clock CLK -max -rise 5 “DISP1[0]”
dc_shell> set_output_delay -clock CLK -max -fall 5 “DISP1[0]”
```

In Tcl mode, the commands are the same except that quotation marks are replaced by braces in the port names. For example,

```
dc_shell-t> set_output_delay -clock CLK -max -rise 5 {SPEAKER_OUT}
dc_shell-t> set_output_delay -clock CLK -max -fall 5 {SPEAKER_OUT}
dc_shell-t> set_output_delay -clock CLK -max -rise 5 {AM_PM_DISPLAY}
dc_shell-t> set_output_delay -clock CLK -max -fall 5 {AM_PM_DISPLAY}
.
.
.
dc_shell-t> set_output_delay -clock CLK -max -rise 5 {DISP1[0]}
dc_shell-t> set_output_delay -clock CLK -max -fall 5 {DISP1[0]}
```

Example Output

After each command is executed, dc_shell displays a confirmation message similar to the following one displayed when the output delay is set on port SPEAKER_OUT:

```
Performing set_output_delay on port ‘SPEAKER_OUT’.
```

Checking the Design Using dc_shell

After you set constraints on a design and before you compile the design, run the check_design command to identify and correct any problems.

Note:

Use Design Analyzer instead of dc_shell to see the schematic for a design before and after the synthesis process.

Check the Design

To check that the internal representation of the design is correct,

- Enter one of the following pairs of commands at the dc_shell prompt:

  ```
  dc_shell> check_design
  dc_shell> check_timing
  ```

  or

  ```
  dc_shell-t> check_design
  dc_shell-t> check_timing
  ```

  These commands issue appropriate warnings or errors.

Example Output

When the check_design command completes execution, dc_shell displays the following messages:

```
Warning: In design ‘CONVERTOR_CKT’, a pin on submodule ‘U7’ is connected to logic
1 or logic 0. (LINT-32)
    Net ‘data _in1[5]’ is connected to pins ‘T0’, ‘T1’.
Warning: Design ‘CONVERTOR’ is instantiated 2 times. (LINT-45)
         Cell ‘U7’ in design ‘CONVERTOR_CKT’
         Cell ‘U8’ in design ‘CONVERTOR_CKT’
1
```

A section of the informative, cautionary, and error messages the dc_shell displays when the check_timing command completes execution follows:

```
  Information: Updating design information...(UID-85)
     Allocating blocks in ‘U1/U1’
Reading in the Synopsys synthetic primitives.
                .
                .
                .
Warning: Design ‘TOP’ contains unmapped cells.
        Use report_cell to list unmapped cells in the design. (OPT-309)
```

```
Warning: The following end-points are not constrained for maximum delay.

End point
---------------
AM_PM_DISPLAY
DISP1[0]
DISP1[1]
DISP1[2]
DISP1[3]
DISP1[4]
DISP1[5]
DISP1[6]
DISP1[7]
DISP1[8]
DISP1[9]
DISP1[10]
DISP1[11]
DISP1[12]
DISP1[13]
DISP2[0]
DISP2[1]
DISP2[2]
DISP2[3]
    .
    .
    .
```

The warning message at the beginning of the output indicates that not all of the reported endpoints have been constrained. (After the endpoints are constrained, the warning message will not be issued.)

Display the Pin Names Layer

The following commands set the visual characteristics of the pin names layer. Because they set visible features of the pin layer, their effect has no bearing on the results you see when you use the dc_shell interface version of Design Compiler. You might want to run these commands from a script file to set the pin layer features for viewing within Design Analyzer. For example, to ensure consistent settings, you could store these commands in a script file that is executed by several design engineers working on the same design. See Appendix A, “Tutorial Script Files.”

```
dc_shell> set_layer pin_name_layer visible TRUE
dc_shell> set_layer pin_name_layer line_width 1
dc_shell> set_layer pin_name_layer plot_line_width 0
dc_shell> set_layer pin_name_layer red 65535
dc_shell> set_layer pin_name_layer green 65535
dc_shell> set_layer pin_name_layer blue 65535
```

These commands are the same in Tcl mode. For example,

```
dc_shell-t> set_layer pin_name_layer visible TRUE
```

Resolving Multiple Design Instances Using dc_shell

Hierarchical designs can reference a subdesign more than once. When the same subdesign is referenced more than once in a design, multiple instances exist in the design. Resolve multiple design instances before you compile.

You can resolve multiple instances in three ways:

- Use the compile-once-don’t-touch method.

  This method (set_dont_touch command) preserves the subdesign during optimization of the design by setting the dont_touch attribute on cells or references.

- Remove the hierarchy (ungroup command).

  This method removes the hierarchy and generates designs with unique names for all cells and references in the current design hierarchy.

- Make each instance unique by making a copy for each instance (uniquify command).

This method generates designs with unique names for all cells and references in the current design hierarchy.

This exercise uses the third method of issuing the uniquify command. The uniquify command creates one copy of the subdesign each time it is referenced in the design and assigns a unique design name to the copy. References to the subdesign are updated to reflect the new names wherever they occur in the hierarchy. During compilation, the Design Compiler optimization maps each instance to its unique environment.

For the Alarm Clock design, the uniquify option is the most effective method because each instance of CONVERTOR needs to be optimized with respect to its environment and the hierarchy for the alarm clock needs to be preserved.

The set_dont_touch and ungroup commands appear in optional exercises at the end of Chapter 9, “Compiling a Hierarchical Design.”

To resolve multiple references to a subdesign,

- Enter the following command at the dc_shell prompt:

  ```
  dc_shell> uniquify
  ```

  or

  ```
  dc_shell-t> uniquify
  ```

Example Output

When you execute the uniquify command, dc_shell displays the following output:

```
Uniquifying cell ‘U8’ in design ‘CONVERTOR_CKT’. New design is ‘CONVERTOR_0’.
Uniquifying cell ‘U7’ in design ‘CONVERTOR_CKT’. New design is ‘CONVERTOR_1’.
```

Saving the Design Using dc_shell

Use this version of TOP as the starting point for the exercises in the next chapter.

To save the design,

- Enter one of the following commands from the dc_shell prompt:

  ```
  dc_shell> write -format db -hierarchy -output “./db/
  TOP_before_compile.db” {“TOP_attributes.db:TOP”}
  ```

  or

  ```
  dc_shell-t> write -format db -hierarchy -output {./db
  TOP_before_compile.db} [list {TOP_attributes.db:TOP}]
  ```

Then quit dc_shell or go on to the next chapter.

To quit dc_shell,

- At the dcsh or dctcl mode command line prompt, type **quit** or **exit**.

9

Compiling a Hierarchical Design

Optimization is the step in the synthesis process that attempts to implement a combination of library cells that meets the functional, area, and speed requirements of the design. The Design Compiler compile command invokes optimization. The compile process modifies and optimizes the design as it attempts to create a circuit that meets the specified constraints.

This chapter includes the following sections:

- Before You Begin

  Explains how to read in the design, using either interface, Design Analyzer or dc_shell. This section also summarizes the design environment and attribute settings you have already made in previous chapters to prepare to compile the design.

- Compiling and Analyzing the Design Using Design Analyzer

Explains how to compile the design for optimization and how to evaluate and interpret the results. It discusses how to explore the design space to review the results. Then, it explains how to set tighter constraints and recompile the design. Finally, it explains how to generate reports and analyze them.

- Compiling and Analyzing the Design Using dc_shell

  Explains how to use dc_shell in both the dcsh and dctcl modes to compile and optimize a design, set tighter constraints, and generate reports.

- Optional Exercises

  Gives optional exercises that show you how to use dc_shell commands to manipulate instances and report their attributes. Other optional exercises show you how to resolve multiple instances, using the set_dont_touch and ungroup commands.

Before You Begin

Before beginning, complete the exercises in Chapter 8, “Defining Optimization Goals and Setting Constraints.”

Allow about one hour to complete the main exercises in this chapter. The optional exercises require about 45 minutes.

To prepare for the exercises in this chapter if you are using Design Analyzer,

1. Start the Design Analyzer interface from the tutorial directory.
2. Change to the db subdirectory.

3. Choose File > Read to read in TOP_before_compile.db from the db directory.

To prepare for the exercises in this chapter if you are using the dc_shell interface,

- Enter one of the following pairs of commands:

  ```
  dc_shell> remove_design -all
  dc_shell> read -format db {“./db/TOP_before_compile.db”}
  ```

  or

  ```
  dc_shell-t> remove_design -all
  dc_shell-t> read_file -format db [list {./db/
  TOP_before_compile.db}]
  ```

Review the tasks you performed and the constraints you set in Chapter 7, “Setting the Design Environment,” and Chapter 8, “Defining Optimization Goals and Setting Constraints.”

Here is a review of the attributes and constraints you have already set for the TOP_before_compile.db design. You have

- Set these attributes on TOP:
  - Drive strength values

    ALARM = 0.08

    CLK = 0.0335 (used drive_of command on pin Z of target library cell B4I to assign rise and fall values)

    HRS = 0.08

    MINS = 0.08

    SET_TIME = 0.06

    TOGGLE_SWITCH = 0.08

- Load values

  AM_PM_DISPLAY = 2

  DISP1 = 3

  DISP2 = 3

  SPEAKER_OUT = 7.5 (used load_of command on pin A of target library cell IVA)

- Wire load model, 10X10
- Operating conditions, WCCOM

- Set these constraints on TOP
  - Clock period of 25 on port CLK
  - Output delay constraint of 5 on all output ports
- Run uniquify to resolve multiple design instances

Compiling and Analyzing the Design Using Design Analyzer

This section contains a description of tasks and exercises in the following sections:

- Compiling the Design to Optimize It Using Design Analyzer
- Evaluating and Interpreting the Design Using Design Analyzer
- Exploring the Design Space Using Design Analyzer
- Setting Tighter Time Constraints Using Design Analyzer
- Recompiling the Design Using Design Analyzer

- Generating New Reports Using Design Analyzer
- Analyzing the New Reports Using Design Analyzer
- Saving the Newly Compiled Version Using Design Analyzer

Design Compiler automatically compiles all levels of the design hierarchy, upholding the hierarchical structure of the design.

During compilation, these five major phases of gate-level optimization of a module can occur in a design:

1. Initial sequential optimization
2. Combinational optimization
3. I/O pad optimization
4. Final sequential optimization
5. Localized adjusting

After compiling each module in the design, Design Compiler continues to optimize the circuit until the constraints are met. Sometimes this process requires recompiling subdesigns on a critical path. When the performance goals are achieved or when no further improvement can be made, the compile process stops.

By default, each subdesign in a hierarchical design is compiled separately. During compilation of subdesigns, Design Compiler optimization takes into account the unique boundary conditions of each subdesign.

Before compilation, resolve any multiple instances of the same design by using the uniquify set_dont_touch or ungroup command.

Compiling the Design to Optimize It Using Design Analyzer

After you set constraints and attributes on the Alarm Clock design and run check_design, the design is ready to be optimized with the compile command.

In this section, you

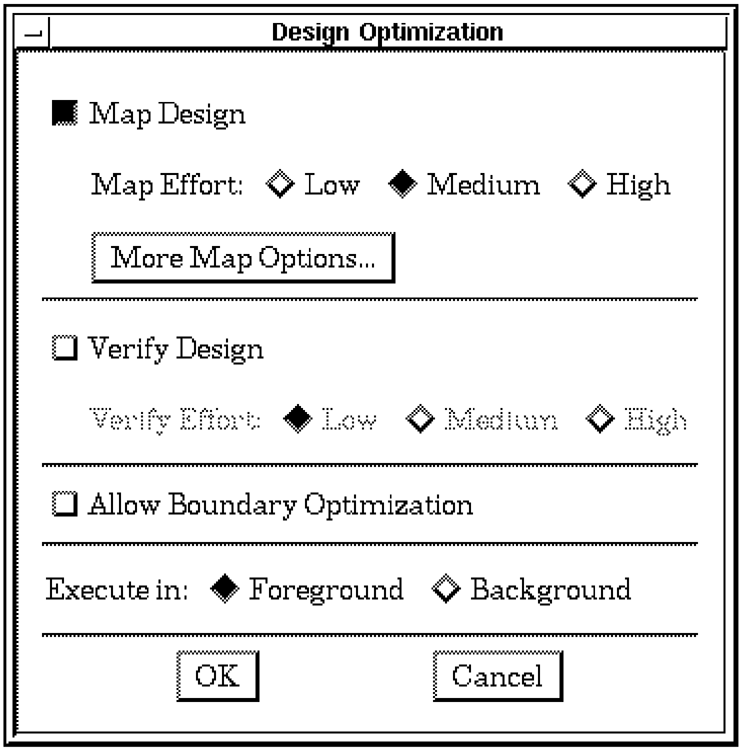
1. Open the Design Optimization window
2. Check and set options
3. Initiate optimization
4. Read in modified designs
5. Examine the design optimization messages

To open the Design Optimization window,

1. Select TOP in the Designs view.
2. Choose Tools > Design Optimization.

   The Design Optimization window, shown in Figure 9-1, appears.

Figure 9-1   Design Optimization Window



Design Optimization

■ Map Design

Map Effort: ◇ Low ◆ Medium ◇ High

□ Verify Design

Verify Effort: ◆ Low ◇ Medium ◇ High

□ Allow Boundary Optimization

Execute in: ◆ Foreground ◇ Background

To check and set options,

1. Note the default settings.

   Map Design (mapping) is selected.

   Medium Map Effort (the CPU effort used to map a design) is selected.

   Use these default settings. For most optimizations, the defaults are sufficient to meet defined constraints.

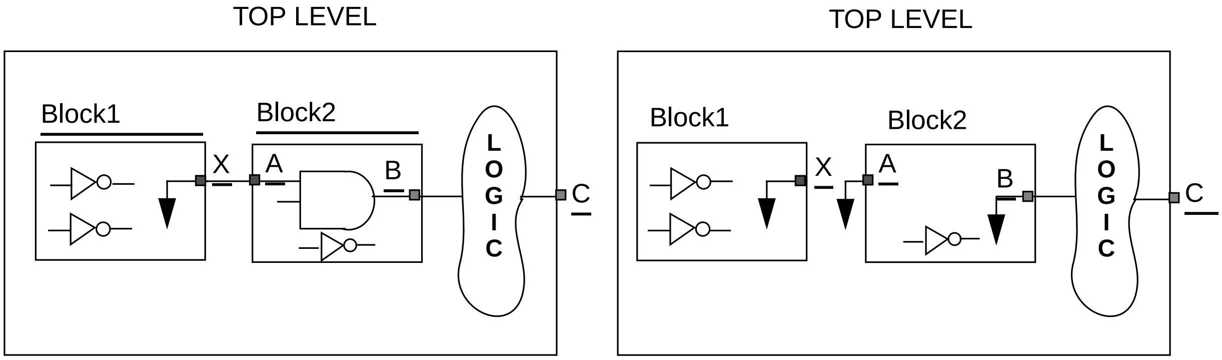
2. Click Verify Design to select it.

This checks that the new synthesized design is functionally equivalent to the original design. Select Verify Effort Low (the CPU effort used to verify the design).

3. Click Allow Boundary Optimization to select it.

   Boundary optimization allows for logic optimization across module boundaries. For example, if the signal to an input port of a subdesign is always logic 0, boundary optimization can simplify the logic in the subdesign, as shown in Figure 9-2.

Figure 9-2 Boundary Optimization (Design Analyzer)



Two input ports in the CONVERTOR module are not being used. With boundary optimization, the logic associated with these two input ports is optimized away.

Compilation can run in either the background or the foreground. In this exercise, you compile in the foreground.

Note:

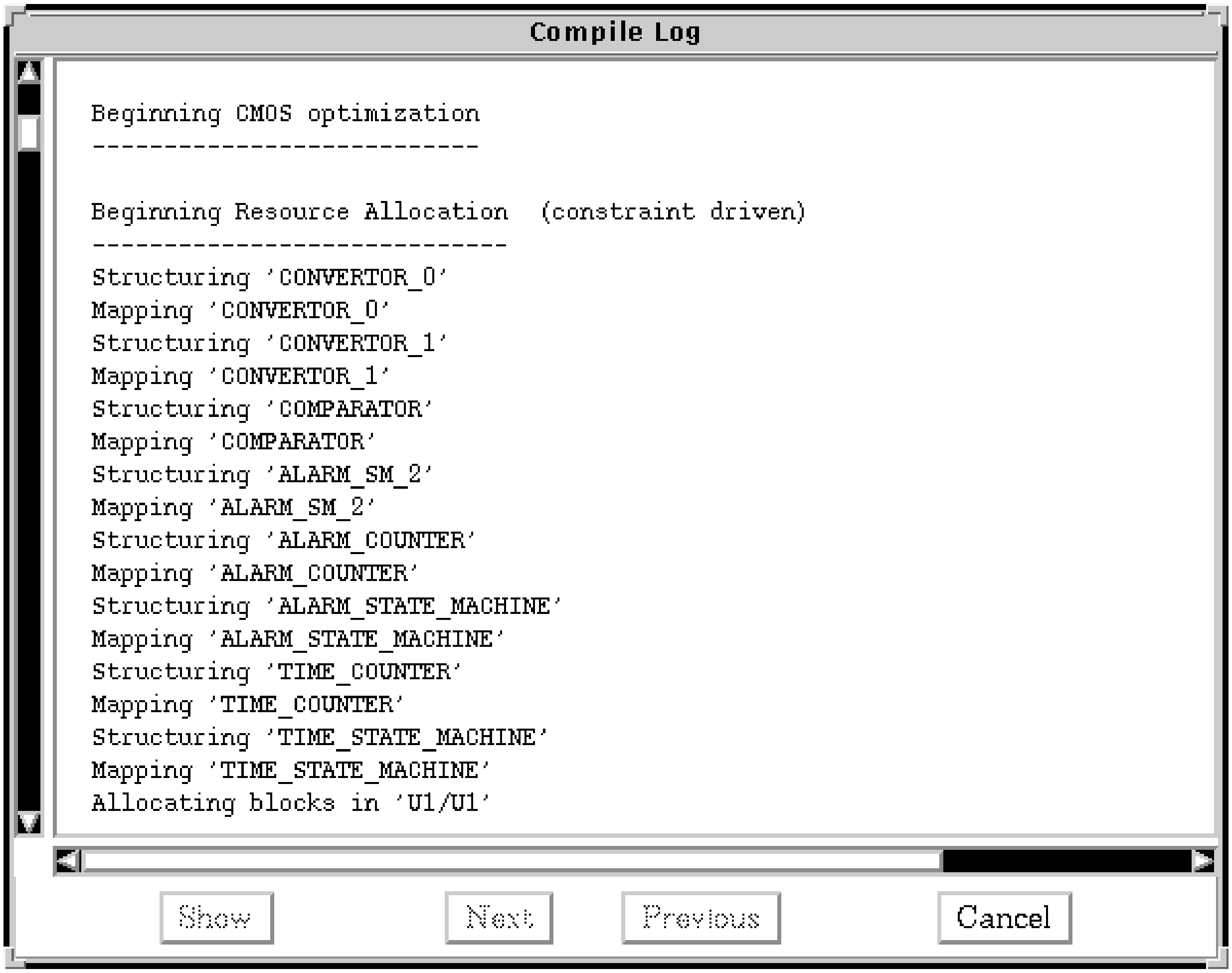
Compiling in the background has advantages. You can continue to work on the design while compiling it and you can save different versions of a compiled design in separate directories. Saving different versions allows you to select the one with the best results.

To initiate optimization in the foreground,

- Click "Execute in: Foreground" and in the Design Optimization window click OK to begin optimization.

After reading in the default variables and design for compile, Design Compiler begins CMOS optimization. Figure 9-3 shows the reported results. During this phase, Design Compiler attempts to meet the specified constraints.

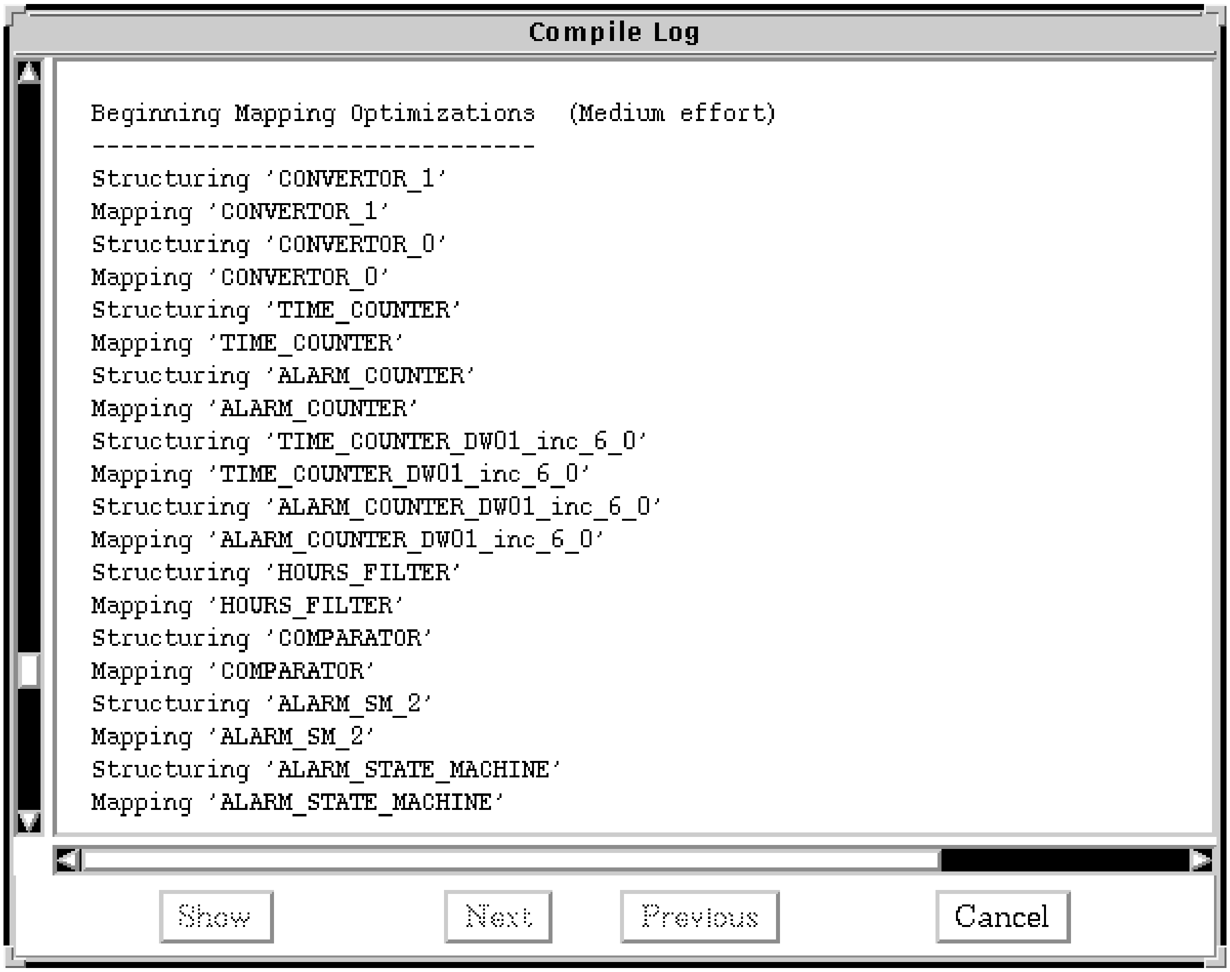
Figure 9-3   CMOS Optimization Phase



```
Beginning CMOS optimization
---------------------------

Beginning Resource Allocation  (constraint driven)
-----------------------------
Structuring 'CONVERTOR_0'
Mapping 'CONVERTOR_0'
Structuring 'CONVERTOR_1'
Mapping 'CONVERTOR_1'
Structuring 'COMPARATOR'
Mapping 'COMPARATOR'
Structuring 'ALARM_SM_2'
Mapping 'ALARM_SM_2'
Structuring 'ALARM_COUNTER'
Mapping 'ALARM_COUNTER'
Structuring 'ALARM_STATE_MACHINE'
Mapping 'ALARM_STATE_MACHINE'
Structuring 'TIME_COUNTER'
Mapping 'TIME_COUNTER'
Structuring 'TIME_STATE_MACHINE'
Mapping 'TIME_STATE_MACHINE'
Allocating blocks in 'U1/U1'
```

The process proceeds to the mapping optimizations phase; the logged results are shown in Figure 9-4.

Figure 9-4   Mapping Optimizations Phase Results



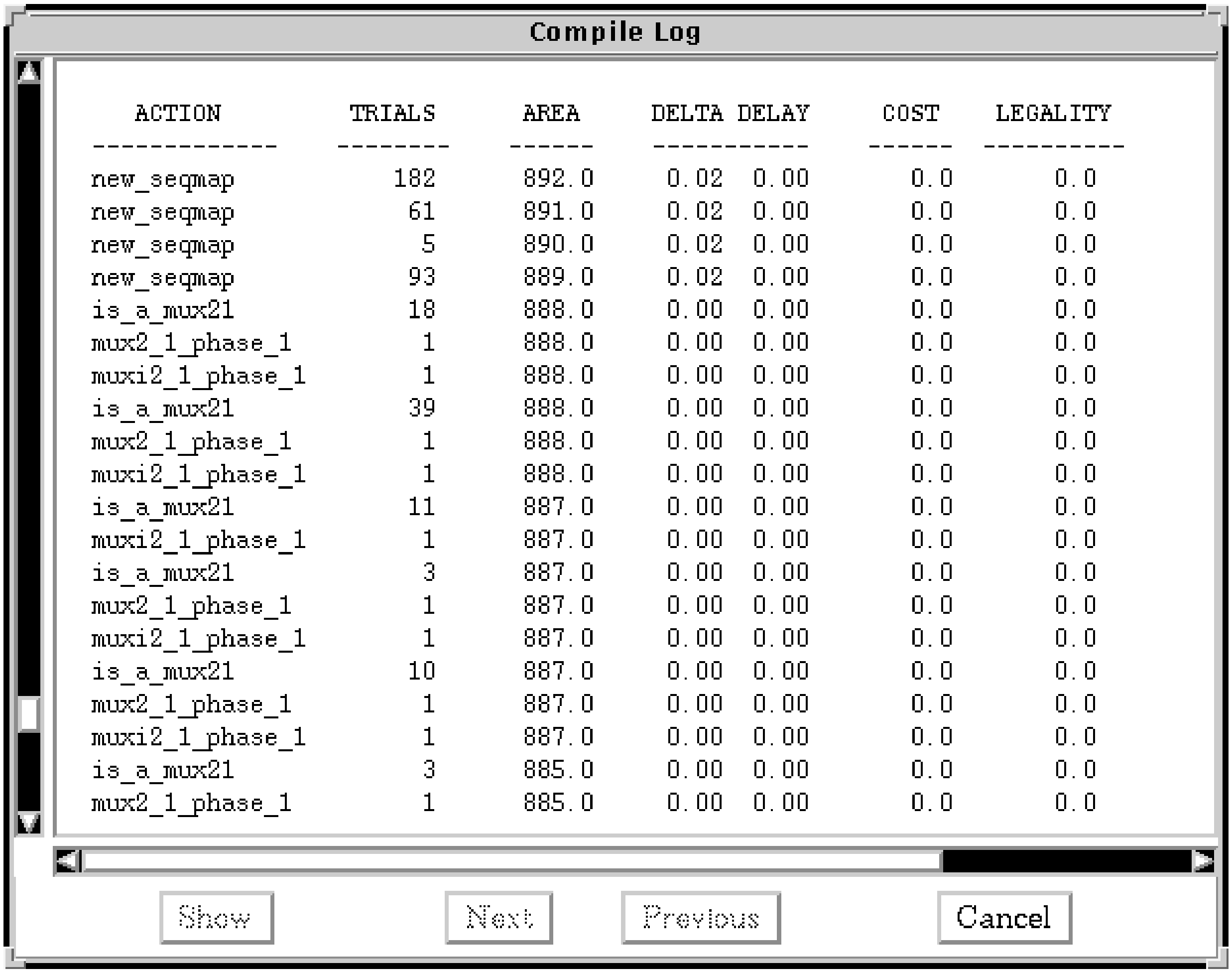
```
Beginning Mapping Optimizations  (Medium effort)
-------------------------------
Structuring 'CONVERTOR_1'
Mapping 'CONVERTOR_1'
Structuring 'CONVERTOR_0'
Mapping 'CONVERTOR_0'
Structuring 'TIME_COUNTER'
Mapping 'TIME_COUNTER'
Structuring 'ALARM_COUNTER'
Mapping 'ALARM_COUNTER'
Structuring 'TIME_COUNTER_DW01_inc_6_0'
Mapping 'TIME_COUNTER_DW01_inc_6_0'
Structuring 'ALARM_COUNTER_DW01_inc_6_0'
Mapping 'ALARM_COUNTER_DW01_inc_6_0'
Structuring 'HOURS_FILTER'
Mapping 'HOURS_FILTER'
Structuring 'COMPARATOR'
Mapping 'COMPARATOR'
Structuring 'ALARM_SM_2'
Mapping 'ALARM_SM_2'
Structuring 'ALARM_STATE_MACHINE'
Mapping 'ALARM_STATE_MACHINE'
```

During the mapping optimizations phase, Design Compiler does structuring and mapping. In the Compile log, note two newly created designs related to resource sharing. The subdesigns ALARM_COUNTER and TIME_COUNTER use a synthetic resource to implement a multiple addition operation. Only one resource is needed by both designs to implement all addition operations. This resource is the synthetic module DW01_inc_no (incrementor).

During the next stage, Design Compiler tries different strategies to meet the constraints and further improve the circuit.

Your results might vary slightly from those shown in Figure 9-5:

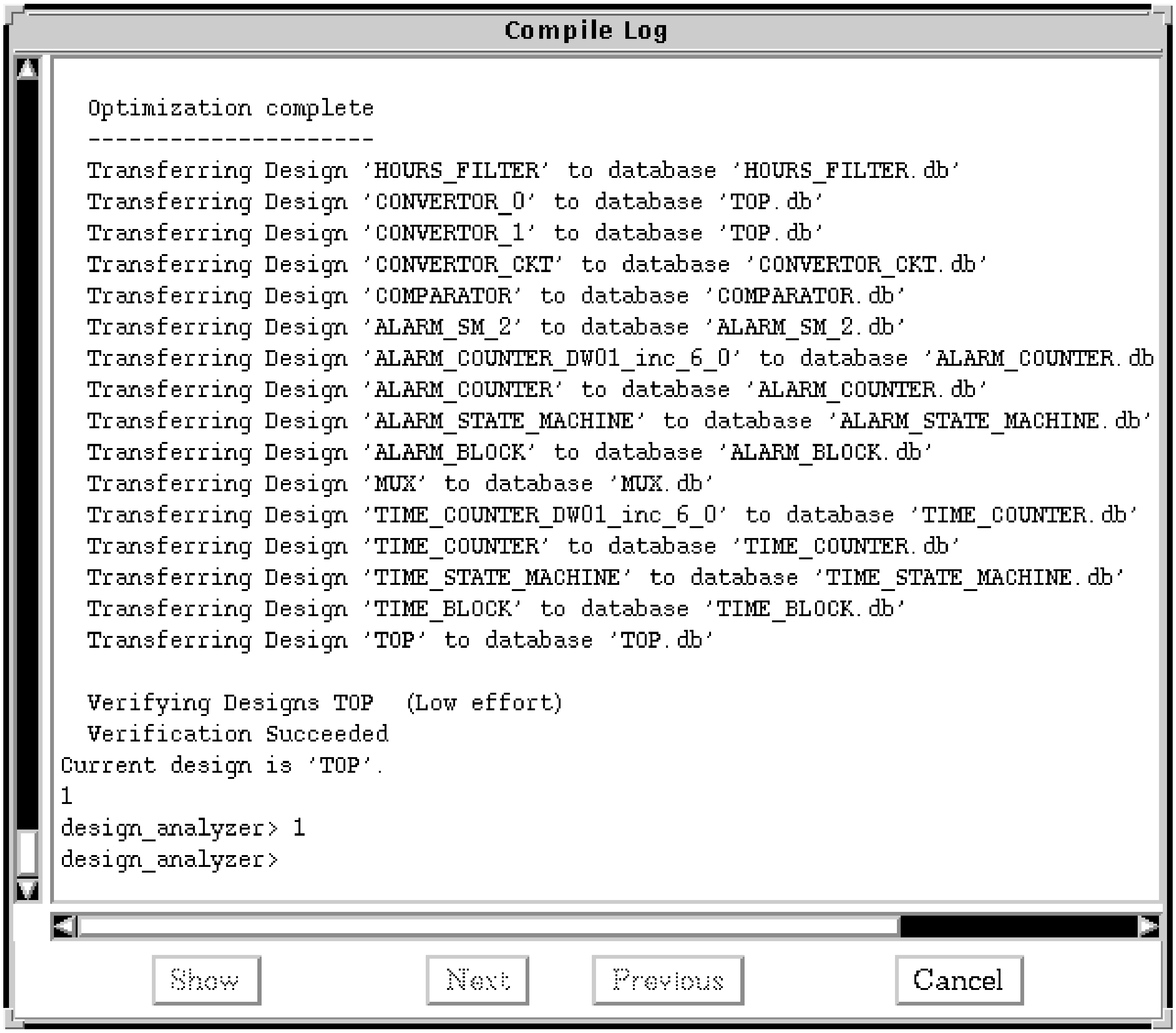
Figure 9-5 Compile Log Report



```
   ACTION          TRIALS       AREA      DELTA DELAY      COST    LEGALITY
-------------    --------     ------     -----------     ------  ----------
new_seqmap            182      892.0       0.02  0.00       0.0         0.0
new_seqmap             61      891.0       0.02  0.00       0.0         0.0
new_seqmap              5      890.0       0.02  0.00       0.0         0.0
new_seqmap             93      889.0       0.02  0.00       0.0         0.0
is_a_mux21             18      888.0       0.00  0.00       0.0         0.0
mux2_1_phase_1          1      888.0       0.00  0.00       0.0         0.0
muxi2_1_phase_1         1      888.0       0.00  0.00       0.0         0.0
is_a_mux21             39      888.0       0.00  0.00       0.0         0.0
mux2_1_phase_1          1      888.0       0.00  0.00       0.0         0.0
muxi2_1_phase_1         1      888.0       0.00  0.00       0.0         0.0
is_a_mux21             11      887.0       0.00  0.00       0.0         0.0
muxi2_1_phase_1         1      887.0       0.00  0.00       0.0         0.0
is_a_mux21              3      887.0       0.00  0.00       0.0         0.0
mux2_1_phase_1          1      887.0       0.00  0.00       0.0         0.0
muxi2_1_phase_1         1      887.0       0.00  0.00       0.0         0.0
is_a_mux21             10      887.0       0.00  0.00       0.0         0.0
mux2_1_phase_1          1      887.0       0.00  0.00       0.0         0.0
muxi2_1_phase_1         1      887.0       0.00  0.00       0.0         0.0
is_a_mux21              3      885.0       0.00  0.00       0.0         0.0
mux2_1_phase_1          1      885.0       0.00  0.00       0.0         0.0
```

When optimization is complete, the Compile Log window looks like the one shown in Figure 9-6:

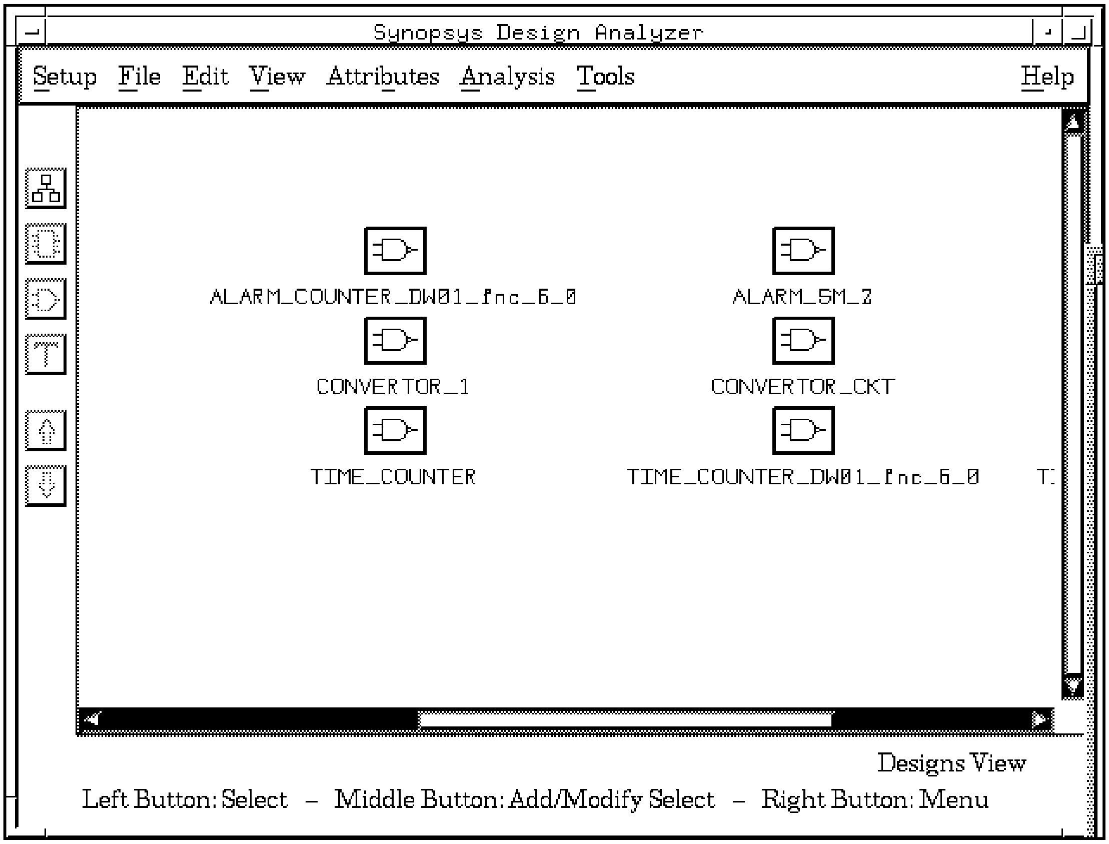
Figure 9-6   The Compile Log Window at Completion of Optimization



```
  Optimization complete
  ---------------------
  Transferring Design 'HOURS_FILTER' to database 'HOURS_FILTER.db'
  Transferring Design 'CONVERTOR_0' to database 'TOP.db'
  Transferring Design 'CONVERTOR_1' to database 'TOP.db'
  Transferring Design 'CONVERTOR_CKT' to database 'CONVERTOR_CKT.db'
  Transferring Design 'COMPARATOR' to database 'COMPARATOR.db'
  Transferring Design 'ALARM_SM_2' to database 'ALARM_SM_2.db'
  Transferring Design 'ALARM_COUNTER_DW01_inc_6_0' to database 'ALARM_COUNTER.db
  Transferring Design 'ALARM_COUNTER' to database 'ALARM_COUNTER.db'
  Transferring Design 'ALARM_STATE_MACHINE' to database 'ALARM_STATE_MACHINE.db'
  Transferring Design 'ALARM_BLOCK' to database 'ALARM_BLOCK.db'
  Transferring Design 'MUX' to database 'MUX.db'
  Transferring Design 'TIME_COUNTER_DW01_inc_6_0' to database 'TIME_COUNTER.db'
  Transferring Design 'TIME_COUNTER' to database 'TIME_COUNTER.db'
  Transferring Design 'TIME_STATE_MACHINE' to database 'TIME_STATE_MACHINE.db'
  Transferring Design 'TIME_BLOCK' to database 'TIME_BLOCK.db'
  Transferring Design 'TOP' to database 'TOP.db'

  Verifying Designs TOP  (Low effort)
  Verification Succeeded
Current design is 'TOP'.
1
design_analyzer> 1
design_analyzer>
```

After compilation finishes, Design Analyzer updates the Designs view, as shown in Figure 9-7.

Figure 9-7   Designs View With Modified Designs After Compilation


Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools
Help
ALARM_COUNTER_DW01_inc_6_0
ALARM_SM_2
CONVERTOR_1
CONVERTOR_CKT
TIME_COUNTER
TIME_COUNTER_DW01_inc_6_0
T
Designs View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Note the two new icons in the Design Analyzer window. These two icons represent the two synthetic modules, ALARM_COUNTER_DW01_inc_6_0 and TIME_COUNTER_DW01_inc_6_0, that implement the addition operation. The synthetic modules create a new level of hierarchy.

The icons are gate symbols because the designs are mapped. When all designs are mapped to gates, you can determine the area and speed of the design.

Evaluating and Interpreting the Design Using Design Analyzer

Generate reports to determine whether the design goals are met. For this design, you need to analyze the area and the constraints.

Your results might vary slightly from the reports shown.

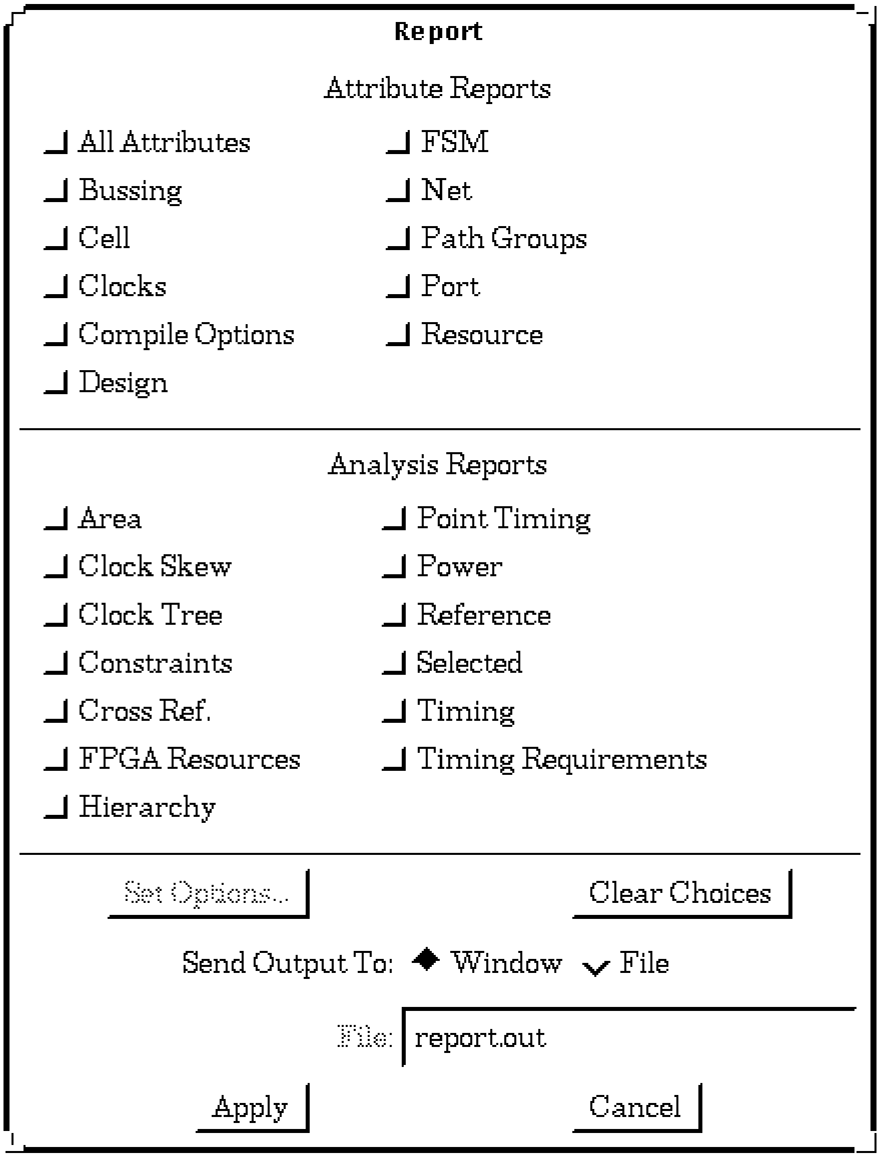
You can select and generate any number of attribute and analysis reports, which send output to a report window or a file you designate.

To open the report window,

1. Display the Symbol View of TOP.
2. Choose Analysis > Report.

   The Report window, as shown in Figure 9-8, opens.

Figure 9-8 Report Window



Report

Attribute Reports

□ All Attributes	□ FSM
□ Bussing	□ Net
□ Cell	□ Path Groups
□ Clocks	□ Port
□ Compile Options	□ Resource
□ Design	

Analysis Reports

□ Area	□ Point Timing
□ Clock Skew	□ Power
□ Clock Tree	□ Reference
□ Constraints	□ Selected
□ Cross Ref.	□ Timing
□ FPGA Resources	□ Timing Requirements
□ Hierarchy	

Set Options... Clear Choices

Send Output To: ◆ Window ✓ File

File: report.out

Apply Cancel

To generate area and constraint reports,

1. If options are set, click Clear Choices.
2. Under Analysis Reports, click Area and Constraints to select them.

3. Click the Send Output To: Window to select it.

   or

   Click "Send Output To: File" and send report results to a file.

4. Click Apply to generate the reports.

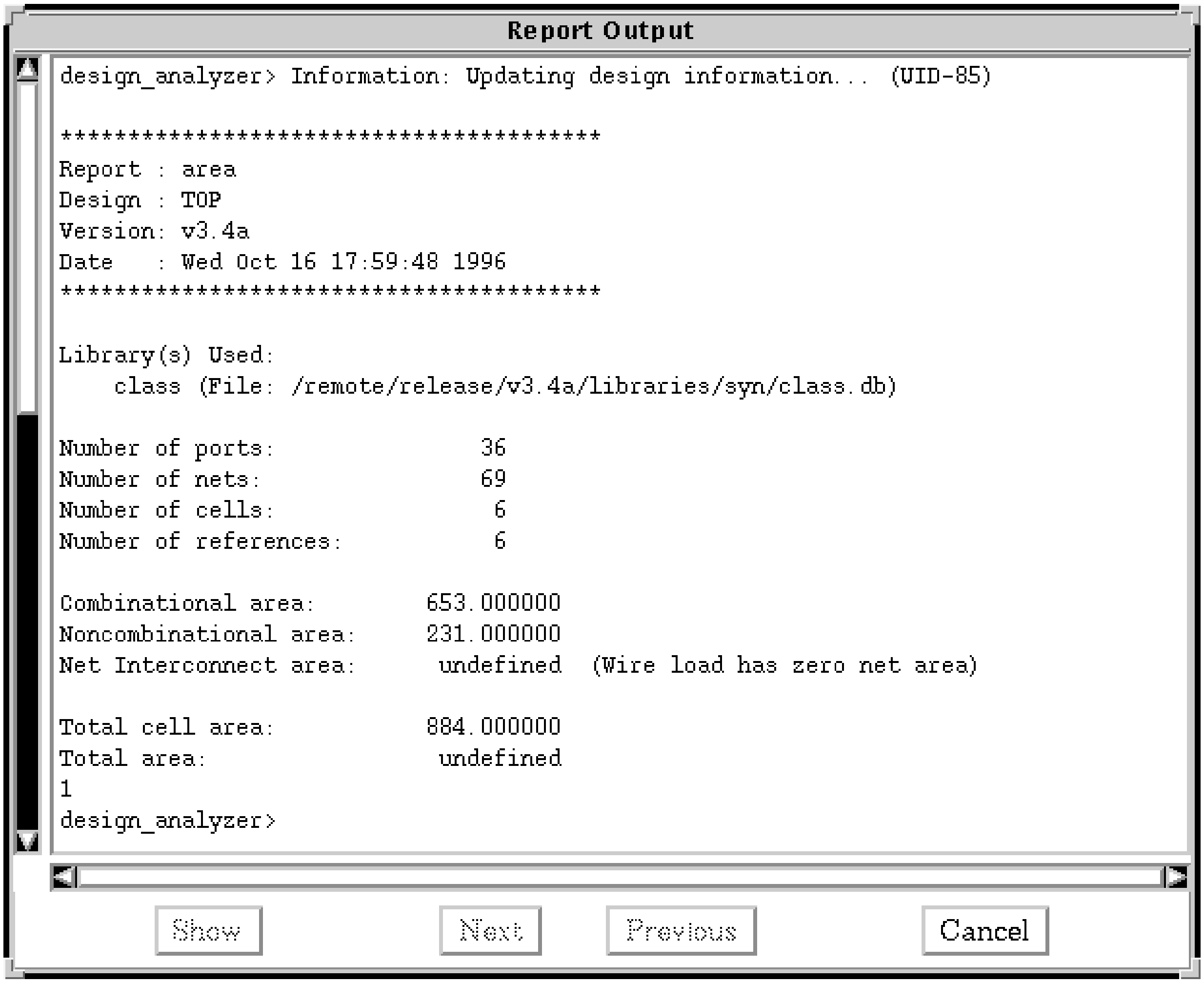
Analyzing the Area Report Using Design Analyzer

Use the area report to determine the circuit area. The area report lists the number of ports, nets, cells, and references in the design. The area of the design is divided into combinational, noncombinational, and net interconnect.

Some ASIC vendors provide a number that is used with the fanout value to estimate the net interconnect area. For this library, the information is not available. The net interconnect area is zero because the wire load model wire area is 0.

The area report, shown in Figure 9-9, indicates that the total cell area is 884 gate equivalents.

Figure 9-9   Area Report

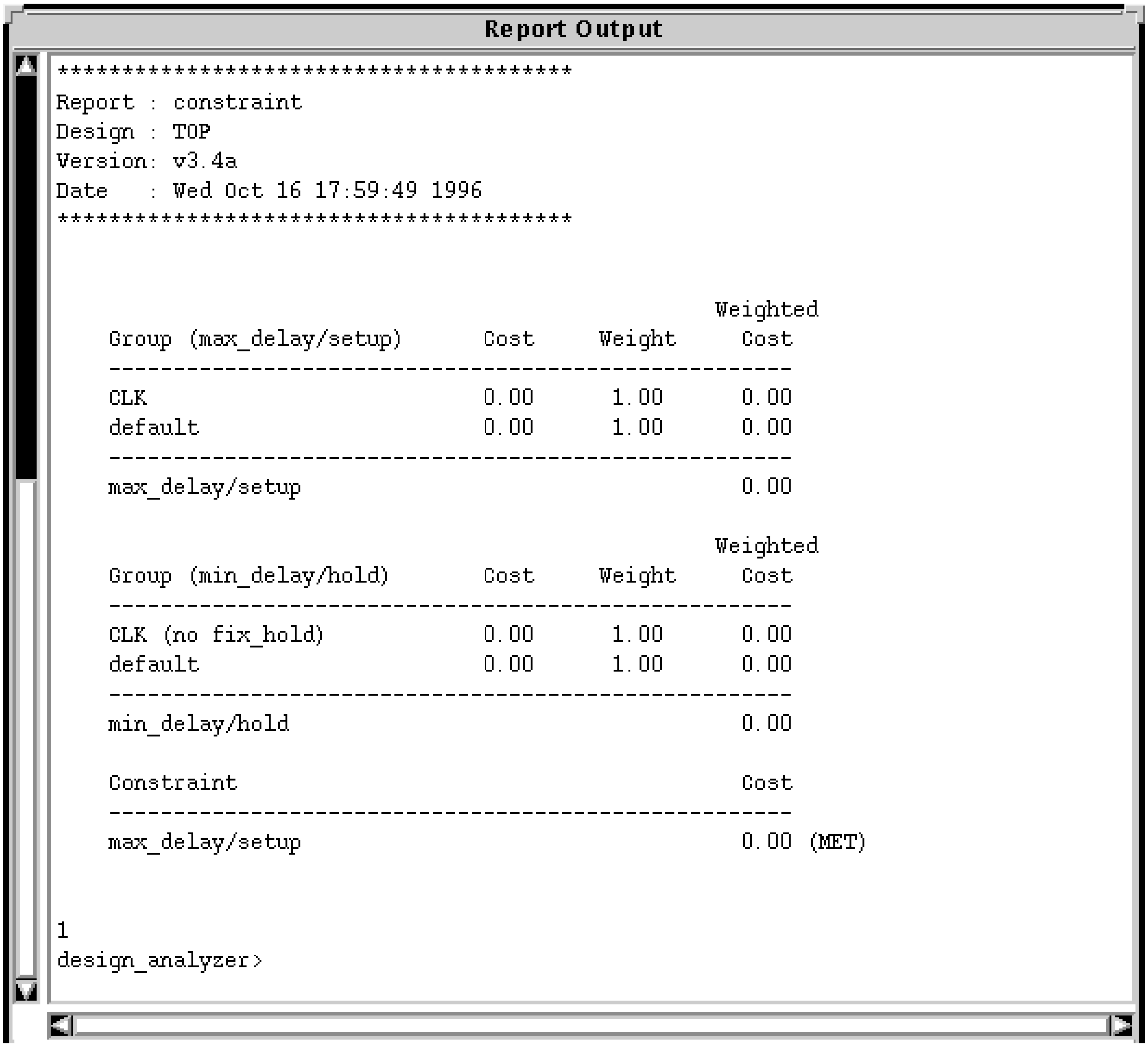


```
design_analyzer> Information: Updating design information... (UID-85)

****************************************
Report : area
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 17:59:48 1996
****************************************

Library(s) Used:
    class (File: /remote/release/v3.4a/libraries/syn/class.db)

Number of ports:               36
Number of nets:                69
Number of cells:                6
Number of references:           6

Combinational area:        653.000000
Noncombinational area:     231.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:           884.000000
Total area:                 undefined
1
design_analyzer>
```

Analyzing the Constraint Report Using Design Analyzer

Use the constraint report to determine whether constraints are met. The constraint report, shown in Figure 9-10, provides information about design rules and optimization constraints. The report includes all violations.

Figure 9-10 Constraints Report



Report Output

```
****************************************
Report : constraint
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 17:59:49 1996
****************************************



                                                     Weighted
    Group (max_delay/setup)      Cost     Weight     Cost
    -----------------------------------------------------
    CLK                          0.00      1.00      0.00
    default                      0.00      1.00      0.00
    -----------------------------------------------------
    max_delay/setup                                  0.00

                                                     Weighted
    Group (min_delay/hold)       Cost     Weight     Cost
    -----------------------------------------------------
    CLK (no fix_hold)            0.00      1.00      0.00
    default                      0.00      1.00      0.00
    -----------------------------------------------------
    min_delay/hold                                   0.00

    Constraint                                       Cost
    -----------------------------------------------------
    max_delay/setup                                  0.00 (MET)


1
design_analyzer>
```

In this constraint report, all the design constraints are met, including the design path delay of 20 ns and the clock period of 25 ns. You define the design path delay of 20 ns by setting the external output delay to 5 ns. Calculate the design path delay by subtracting the output delay from the clock period.

After examining the reports, click Cancel in the Report window and in the Report Output window to close the windows.

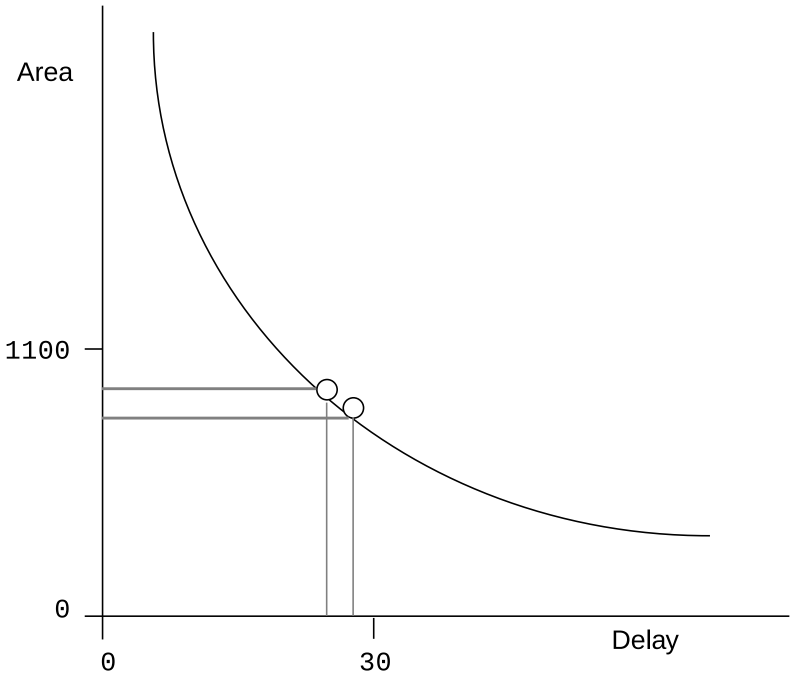
Exploring the Design Space Using Design Analyzer

Experimenting with area and speed to get the smallest or fastest design is called exploring the design space.

Experiment by compiling the design, using different combinations of constraints. Using Design Compiler, you can examine different implementations of the same design in a relatively short time.

Figure 9-11 displays a design space curve. The shape of the curve demonstrates the trade-off between area-efficient and speed-efficient circuits.

Figure 9-11 Design Space Curve



In the following exercise, you explore a smaller part of the design space curve. Recompiling your previously compiled design reduces compile time. However, if you need to take full advantage of all optimization steps, recompile the design from the source code.

Currently the design has an area of 884, a clock period of 25 ns, and a path delay of 20 ns. This design does not have an area constraint set; however, in Chapter 8, the upper limit for area was determined to be 1100. Because the design is not far under the area goal of 1100, there is some room to experiment. Still, you can determine whether the design can be made even faster by allowing optimization to give up some area results for faster performance.

Setting Tighter Time Constraints Using Design Analyzer

Set tighter time constraints on the optimized design. You can set a new clock period of 23 ns to define delay constraints of 18 ns on the output ports.

To set the new clock period,

1. Double-click the small waveform symbol above the input port CLK in the Symbol view.

   The Specify Clock window, shown in Figure 9-12, appears. It displays a period of 25.

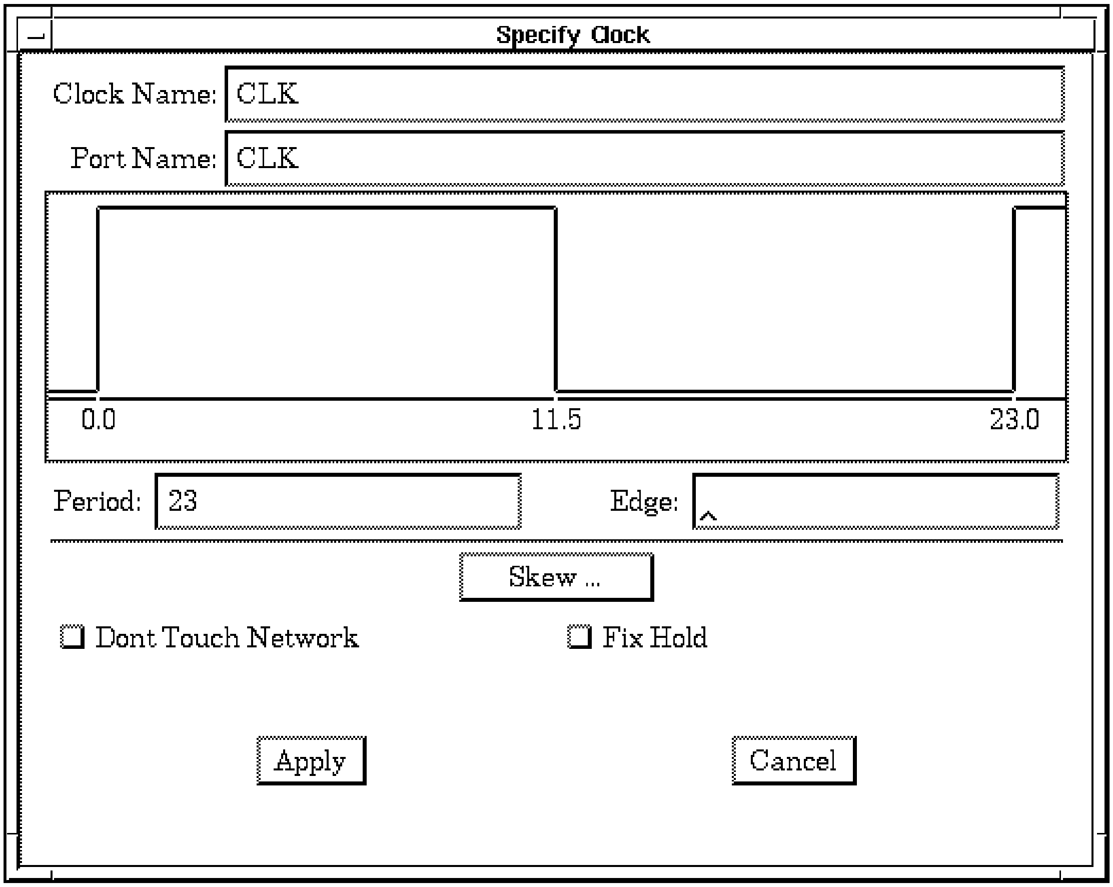
2. Change the Period field value to 23.

   Changing the clock period to 23 results in an expected path delay of 18 ns because the output delay of 5 ns is set relative to the clock.

3. Click Apply.

   The new clock constraint is set, as shown in Figure 9-12.

Figure 9-12   Specify Clock Window



Specify Clock

Clock Name: CLK

Port Name: CLK

Period: 23

Edge:

□ Dont Touch Network

□ Fix Hold

4. Click Cancel to close the Specify Clock window.

Recompiling the Design Using Design Analyzer

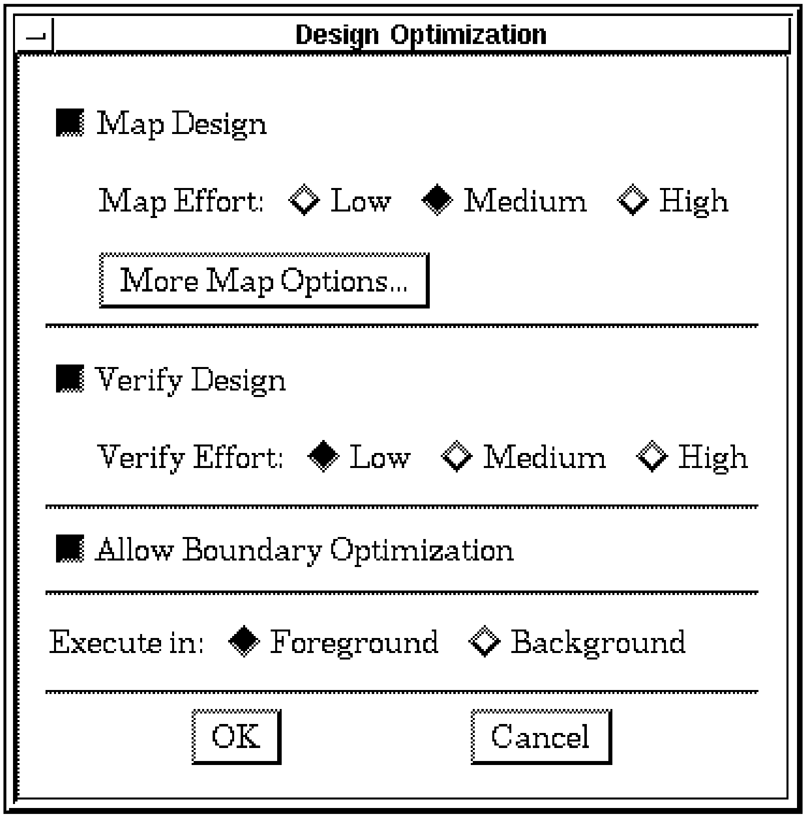
When you recompile the design, use the settings from the last optimization session. Only the clock constraint is changed.

To recompile the design,

1. Choose Tools > Design Optimization.
2. Click "Execute in: Foreground" to select it.

3. Leave the other settings from the previous compilation, as shown in Figure 9-13.

Figure 9-13 Design Optimization Settings for Recompilation



■ Map Design

Map Effort: ◇ Low ◆ Medium ◇ High

■ Verify Design

Verify Effort: ◆ Low ◇ Medium ◇ High

■ Allow Boundary Optimization

Execute in: ◆ Foreground ◇ Background

4. Click OK.

   The Compile Log window appears and displays the activities and results of the design optimization.

5. Browse the Compile Log window.

6. Click Cancel to close the window.

Generating New Reports Using Design Analyzer

After recompiling the design, generate new area and constraints reports.

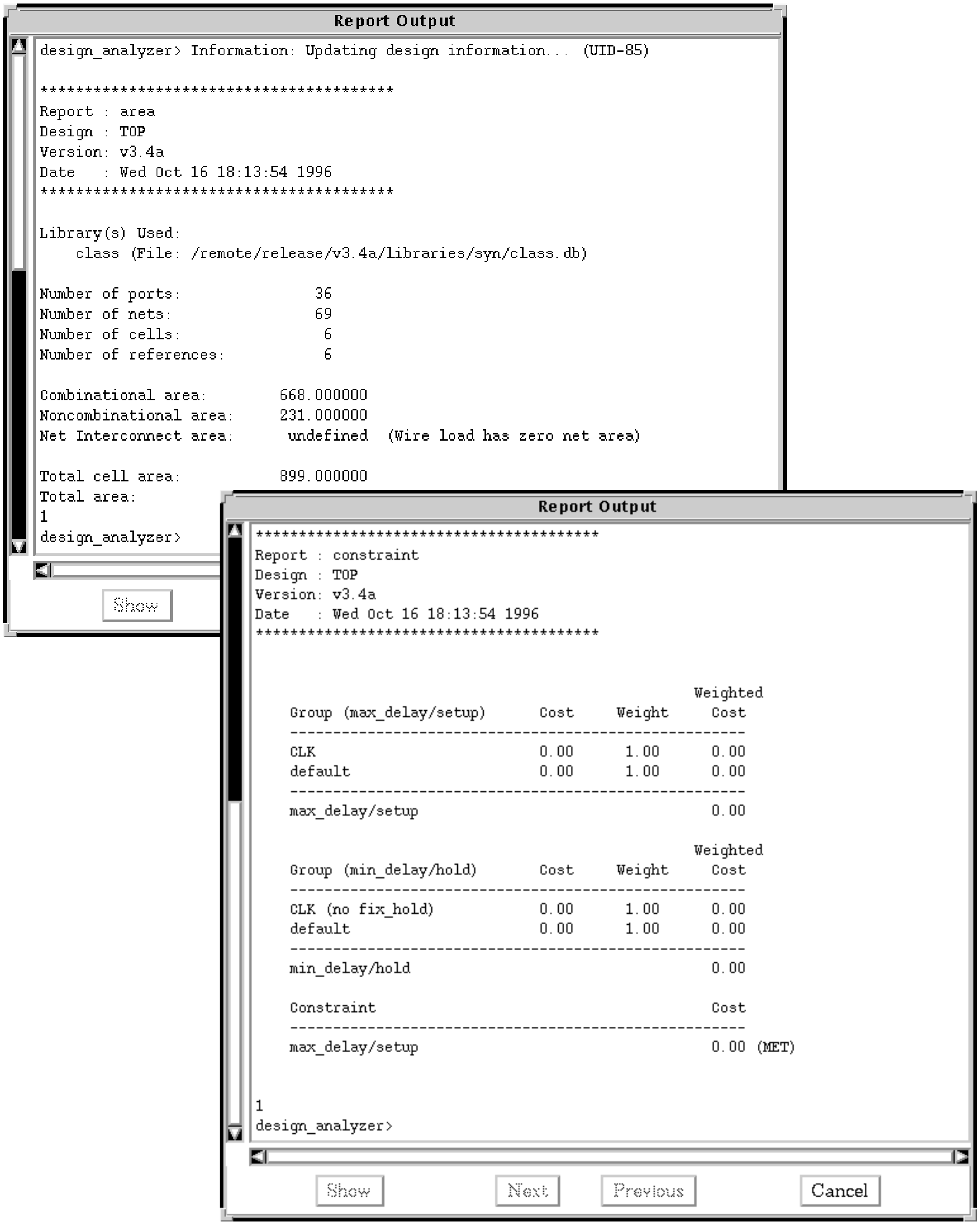
To generate reports,

1. Select TOP.
2. Choose Analysis > Report.
3. Click Area and Constraints to select them.
4. Click Apply to display the Report Output window.

Analyzing the New Reports Using Design Analyzer

The area and timing reports, shown in Figure 9-14, indicate that the second compile run yields better timing results and an increase in design area. The tighter clock constraint is met, but area increases from 884 gates to 899 gates. Even though the area increases in the faster design, the area goal of 1100 is still satisfied.

Figure 9-14   Area and Timing Reports



```
design_analyzer> Information: Updating design information... (UID-85)

****************************************
Report : area
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 18:13:54 1996
****************************************

Library(s) Used:
    class (File: /remote/release/v3.4a/libraries/syn/class.db)

Number of ports:              36
Number of nets:               69
Number of cells:               6
Number of references:          6

Combinational area:        668.000000
Noncombinational area:     231.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:           899.000000
Total area:
1
design_analyzer>
```

```
****************************************
Report : constraint
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 18:13:54 1996
****************************************



                                                   Weighted
    Group (max_delay/setup)      Cost     Weight     Cost
    -----------------------------------------------------
    CLK                          0.00      1.00      0.00
    default                      0.00      1.00      0.00
    -----------------------------------------------------
    max_delay/setup                                  0.00

                                                   Weighted
    Group (min_delay/hold)       Cost     Weight     Cost
    -----------------------------------------------------
    CLK (no fix_hold)            0.00      1.00      0.00
    default                      0.00      1.00      0.00
    -----------------------------------------------------
    min_delay/hold                                   0.00

    Constraint                                       Cost
    -----------------------------------------------------
    max_delay/setup                                  0.00 (MET)


1
design_analyzer>
```

To close the reports after examining them, click Cancel in each window.

Saving the Newly Compiled Version Using Design Analyzer

Having tightened timing constraints and increased performance, save this faster version as TOP_compiled.db. Use this design in the next chapter to further demonstrate reporting and analyzing design results.

To save the faster design,

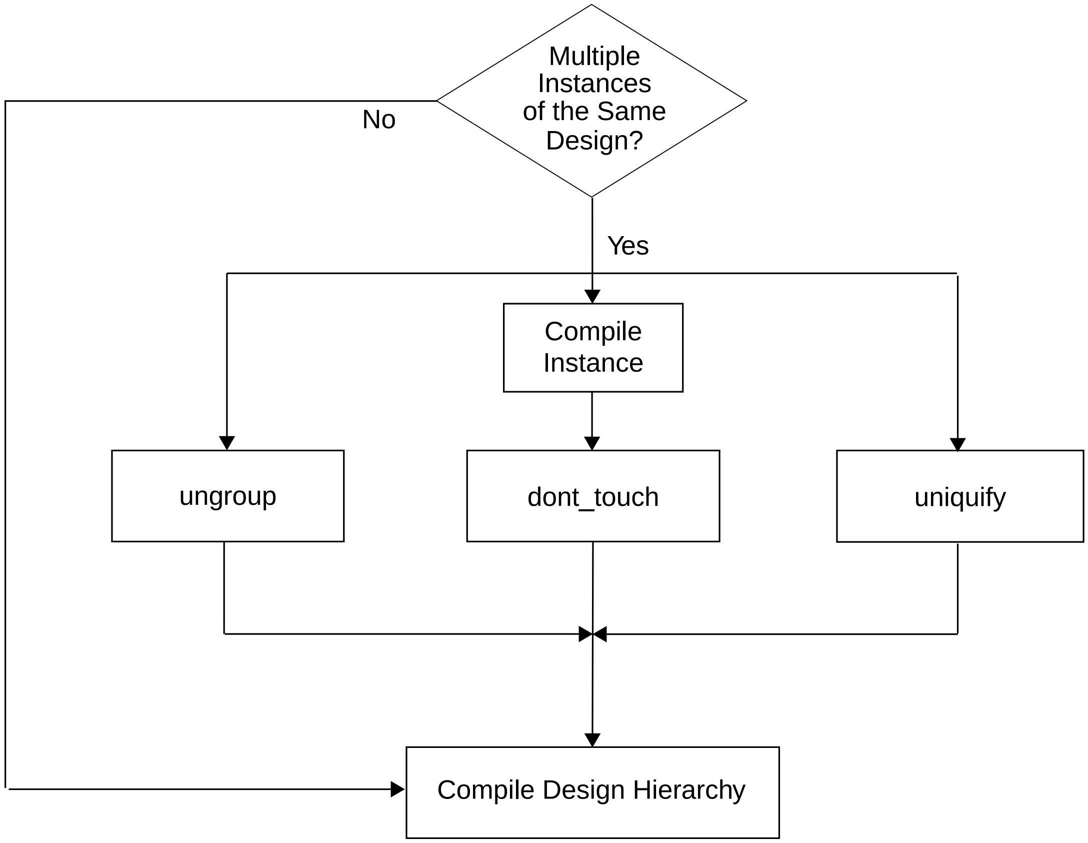
1. Choose File > Save As.
2. Change to the db directory.
3. Enter TOP_compiled.db in the File Name field.
4. Click the "Save All Designs in Hierarchy" option to select it.
5. Click OK.

Using Alternatives to uniquify in Design Analyzer

In Chapter 8, “Defining Optimization Goals and Setting Constraints,” uniquify is used to resolve the multiple instances of CONVERTOR. The following optional exercises describe how to resolve multiple design instances, using the set_dont_touch and ungroup commands.

The decision block in Figure 9-15 shows the options available to you when your design has multiple instances of the same design. You can use ungroup, dont_touch, or uniquify.

Figure 9-15 Decision Block



After one of the three options is successfully completed, run compile to optimize your design.

Undertaking the set_dont_touch Exercise

The set_dont_touch command sets the dont_touch attribute on a subdesign. Design Compiler optimization ignores any subdesign on which the dont_touch attribute is set.

Designers often use set_dont_touch on a subdesign after optimizing the subdesign separately from the design hierarchy. If a specific desired result is achieved in separate optimization, you can choose to avoid changing the design in subsequent optimization sessions. This method of handling subdesigns is called the compile-once-don’t-touch method.

Before you begin,

1. Delete any designs in the Designs view.
2. Read in the design TOP_attributes.db.
3. Select TOP and run check_design with Check Timing not selected.

   The warnings displayed earlier in this chapter are regenerated.

Before you apply the dont_touch attribute, optimize CONVERTOR to gates so that timing information is available.

To optimize CONVERTER,

1. Display the Symbol view for CONVERTOR.
2. Choose Tools > Design Optimization.
3. Click OK to use the default options in the Design Optimization window.

   The Compile Log window appears and displays the activities of compile.

   When the compilation finishes, design CONVERTOR is mapped to gates. The CONVERTOR icon in the Designs view changes from PLA to a NAND gate. This mapped design replaces the PLA description for CONVERTOR.

4. Click Cancel to close the Compile Log window.

To set the dont_touch attribute on CONVERTOR,

1. Choose Attributes > Optimization Directives > Design.

   Attribute values of the selected design are displayed or modified in the Design Attributes window.

2. Select Don’t Touch.
3. Click Apply.

   The dont_touch attribute is set on CONVERTOR.

4. Click Cancel to close the Design Attributes window.

To check the TOP design again,

1. Select TOP in the Designs view.
2. Choose Analysis > Check Design.
3. Make sure that Check Timing is not selected
4. Click OK.

   No warnings are issued about multiple design instances.

   A new warning is issued as a result of the optimization run earlier on CONVERTOR to map it to gates. Because ports A0 and D0 are identical, the two ports are shorted during compile. This warning is informational and does not reflect an error.

5. Click Cancel to close the Design Errors window.

   After the dont_touch attribute is set on the subdesign, optimize the top-level design TOP. The subdesign is not changed by compile during optimization of the hierarchical design.

Undertaking the ungroup Exercise

The ungroup command collapses the hierarchy of the selected design. The design circuitry is merged with the higher-level design during optimization.

Before you begin,

1. Delete any designs in the Designs View.
2. Read in the design TOP_attributes.db.
3. Select TOP.
4. Run check_design to generate the multiple instance warning message. Use the default options.

To set the ungroup attribute on CONVERTOR,

1. Select CONVERTOR in the Designs view.
2. Choose Attributes > Optimization Directives > Design.
3. Select Ungroup.
4. Click Apply.

   The ungroup attribute is set on CONVERTOR.

5. Click Cancel to close the Design Attributes window.

To check the TOP design again,

1. Select TOP in the Designs view.
2. Choose Analysis > Check Design.
3. Make sure Check Timing is not selected.
4. Click OK.

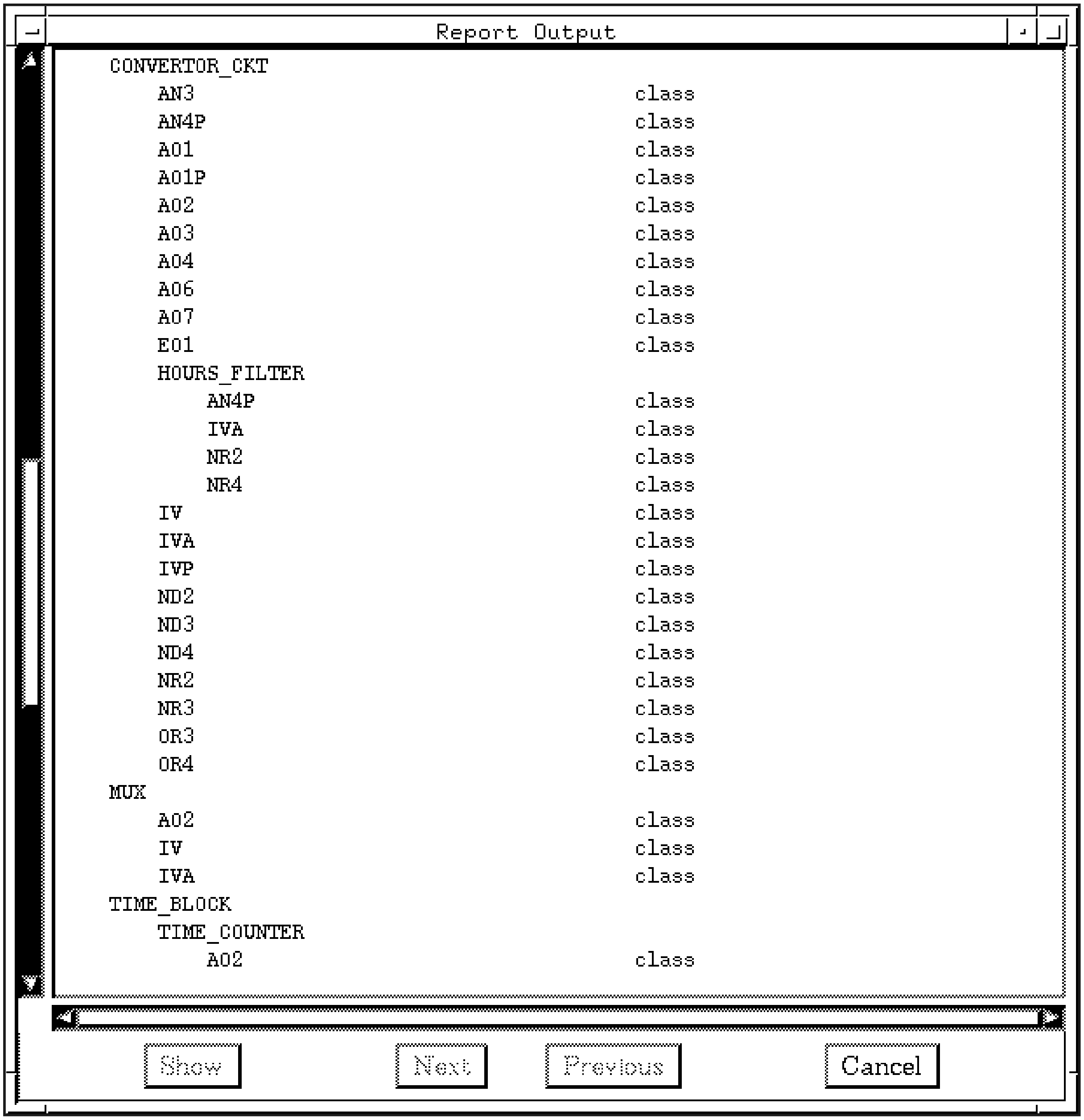
No warnings are issued about multiple design instances.

A new warning is issued as a result of the optimization run earlier on CONVERTOR to map it to gates. Because ports A0 and D0 are identical, the two ports are shorted during compile. This warning is informational and does not reflect an error.

5. Click Cancel to close the Design Errors window.

   When you optimize TOP, the CONVERTOR design merges with the top-level logic. After optimizing TOP
   (use Tools > Design Optimization), run a hierarchy report to view the results. Figure 9-16 shows the hierarchy report.

Figure 9-16   Hierarchy Report



```
CONVERTOR_CKT
    AN3                                    class
    AN4P                                   class
    A01                                    class
    A01P                                   class
    A02                                    class
    A03                                    class
    A04                                    class
    A06                                    class
    A07                                    class
    E01                                    class
    HOURS_FILTER
        AN4P                               class
        IVA                                class
        NR2                                class
        NR4                                class
    IV                                     class
    IVA                                    class
    IVP                                    class
    ND2                                    class
    ND3                                    class
    ND4                                    class
    NR2                                    class
    NR3                                    class
    OR3                                    class
    OR4                                    class
MUX
    A02                                    class
    IV                                     class
    IVA                                    class
TIME_BLOCK
    TIME_COUNTER
        A02                                class
```

Quit Design Analyzer, or go on to the next chapter.

Compiling and Analyzing the Design Using dc_shell

Design Compiler automatically compiles all levels of design hierarchy, upholding the hierarchical structure of the design. During compilation, these five major phases of gate-level optimization of a module can occur in a design:

1. Initial sequential optimization
2. Combinational optimization
3. I/O pad optimization
4. Final sequential optimization
5. Localized adjusting

After compiling each module in the design, Design Compiler continues to optimize the circuit until the constraints are met. Sometimes this process requires recompiling subdesigns on a critical path. When the performance goals are achieved or when no further improvement can be made, the compile process stops.

By default, each subdesign in a hierarchical design is compiled separately. During compilation of subdesigns, Design Compiler optimization takes into account the unique boundary conditions of each subdesign.

This section includes these tasks and exercises:

- Using dc_shell to Compile the Design to Optimize It
- Setting Tighter Time Constraints Using dc_shell
- Recompiling the Design Using dc_shell
- Generating New Reports Using dc_shell

These tasks and exercises are presented in both dcsh and dctcl modes.

Using dc_shell to Compile the Design to Optimize It

Before compilation, resolve any multiple instances of the same design by using the uniquify, set_dont_touch or ungroup command.

After you set constraints and attributes on the Alarm Clock design and check_design, the design is ready to be optimized with the compile command.

To compile the design,

- Enter one of the following commands at the command prompt:

  ```
  dc_shell> compile -map_effort medium -verify
  -verify_effort low -boundary_optimization
  ```

  or

  ```
  dc_shell-t> compile -map_effort medium -verify
  -verify_effort low -boundary_optimization
  ```

The options used for compilation of the design example direct Design Compiler to

- Dedicate the standard CPU effort to the mapping phase of the compilation (-map_effort)
- Perform a functional comparison between the initial design and the synthesized result (-verify)
- Dedicate a relatively low amount of CPU time for the verification phase of the compilation (-verify_effort)

- Optimize the design across all hierarchical boundaries. This can change the function of the design so that it can operate only in its current environment (-boundary_optimization)

  Note:

  > Though used for the tutorial, the -verify and -boundary_optimization parameters are optional.

Example Output

The compilation process produces extensive output, segments of which are shown and explained here.

At the outset of compilation, dc_shell displays initialization and loading information such as the following in the shell window:

```
Loading target library ’class’
Loading design ’TOP’
.
.
.
Using non-hierarchical verification because of boundary
optimization.
Copying Design (for verification)
Allocating blocks in ’U1/U1’
Allocating blocks in ’U1/U2’
Allocating blocks in ’U6’
Allocating blocks in ’U2/U1’
Allocating blocks in ’U2/U2’
Allocating blocks in ’U5’
Allocating blocks in ’U3/U9’
Allocating blocks in ’U4’
```

After reading in the default variables and design for the compile, Design Compiler begins CMOS optimization. During this phase, Design Compiler tries to meet the specified constraints.

```
Beginning CMOS optimization
----------------------------
```

```
Beginning Resource Allocation (constraint driven)
-----------------------------
Structuring ’COMPARATOR’
Mapping ’COMPARATOR’
Structuring ’HOURS_FILTER’
Mapping ’HOURS_FILTER’
       .
       .
       .
Structuring ’TIME_STATE_MACHINE’
Mapping ’TIME_STATE_MACHINE’
Allocating blocks in ’U1/U1’
Allocating blocks in ’U1/U2’
Allocating blocks in ’U6’
       .
       .
       .
```

The process proceeds to the mapping optimizations phase. During the mapping optimizations phase, Design Compiler structures and remaps the design. Two new designs, related to resource sharing, are created. The subdesigns ALARM_COUNTER and TIME_COUNTER use a synthetic resource to implement a multiple addition operation. Only one resource is needed by both designs to implement all additional operations. The resource is the synthetic module DW01_inc_no (incrementor).

```
Beginning Mapping Optimizations (Medium effort)
-----------------------------------------------
Structuring ’TIME_COUNTER_DW01_inc_6_1’
Mapping ’TIME_COUNTER_DW01_inc_6_1’
Structuring ’TIME_COUNTER_DW01_inc_6_0’
Mapping ’TIME_COUNTER_DW01_inc_6_0’
Structuring ’ALARM_COUNTER_DW01_inc_6_0’
Mapping ’ALARM_COUNTER_DW01_inc_6_0’
Structuring ’COMPARATOR’
Mapping ’COMPARATOR’
Structuring ’HOURS_FILTER’
Mapping ’HOURS_FILTER’
```

```
                 .
                 .
                 .
```

During the next phase, Design Compiler tries different strategies to meet the constraints and further improve the circuit.

```
Beginning Incremental Mapping Optimizations
-------------------------------------------
Selecting implementations in ’U1/U2’
Selecting implementations in ’U2/U2’


                                          TOTAL NEG     DESIGN RULE
   ACTION   TRAILS   AREA    DELTA DELAY   SLACK          COST
  --------  ------   ----    -----------   -----        ----------
                 0
            ------
                 0

  Beginning Area-Recovery Phase (cleanup)

                                             TOTAL NEG     DESIGN RULE
   ACTION       TRAILS   AREA    DELTA DELAY   SLACK          COST
  --------      ------   ----    -----------   -----        ----------
  sdn_clean_inv      2   1045.0  0.00    0.00    0.0             0.0
  sdn_clean_buf      2   1041.0  0.00    0.00    0.0             0.0
  sdn_unphse         2   1041.0  0.00    0.00    0.0             0.0

                         .
                         .
                         .
  sdn_single_size   179  1024.0  0.00    0.00    0.0             0.0
  sdn_single_size    25  1023.0  0.00    0.00    0.0             0.0
                         .
                         .
                         .
```

When optimization is complete, Design Compiler displays the following output:

```
Optimization Complete
---------------------
Transferring Design ’COMPARATOR’ to database ’compile.db’
Transferring Design ’HOURS_FILTER’ to database ’compile.db’
Transferring Design ’CONVERTOR_0’ to database ’compile.db’

                         .
                         .
                         .
Transferring Design ’TIME_COUNTER_DW01_inc_6_1’ to database ’compile.db’
Transferring Design ’TIME_COUNTER_DW01_inc_6_0’ to database ’compile.db’
Transferring Design ’ALARM_COUNTER_DW01_inc_6_0’ to database ’compile.db’
Transferring Design ’ALARM_COUNTER’ to database ’compile.db’
Transferring Design ’ALARM_BLOCK’ to database ’compile.db’
Transferring Design ’MUX’ to database ’compile.db’
                         .
                         .
                         .

Verifying Designs TOP (Low effort)
Verification Succeeded
Current design is ’TOP’
1
```

Setting Tighter Time Constraints Using dc_shell

Set tighter time constraints on the optimized design. You can set a new clock period of 23 ns to define delay constraints of 18 ns on the output ports.

To set new timing constraints on the design,

- Enter one of the following dc_shell commands:

  ```
  dc_shell> create_clock -name CLK -period 23 -waveform
  { 0 11.5 }{ CLK }
  ```

  or

  ```
  dc_shell-t> create_clock -name CLK -period 23 -waveform
  [ list 0 11.5 ] CLK
  ```

  In response, dc_shell displays the following confirmation message:

```
Performing create_clock on port ’CLK’.
1
```

Recompiling the Design Using dc_shell

When you recompile the design, use the settings from the last optimization session. Only the clock constraint is changed.

To recompile the design using a different clock constraint,

- Enter one of the following pairs of commands at the command prompt:

  ```
  dc_shell> compile -map_effort
  dc_shell> current_design = “compile.db:TOP”
  ```

  or

  ```
  dc_shell-t> compile -map_effort
  dc_shell-t> set current_design {compile.db:TOP}
  ```

Generating New Reports Using dc_shell

After recompiling the design, generate new area and constraints reports.

To generate new reports,

- Enter one of the following pairs of commands from the command prompt:

  ```
  dc_shell> report_area
  dc_shell> report_constraints
  ```

  or

  ```
  dc_shell-t> report_area
  ```

```
dc_shell-t> report_constraints
```

Example Output

Use the area report to find out the circuit area. The area report lists the number of ports, nets, cells, and references in the design. The area of the design is divided into combinational, noncombinational, and net interconnect.

Some ASIC vendors provide a number that is used with the fanout value to estimate the net interconnect area. For this library, the information is not available. The net interconnect area is zero because the wire load model wire area is zero.

```
*************************************************
Report  : area
Design  : TOP
Version : 1998.08
Date    : Thu Oct 15 1998
*************************************************

Library(s) used:

class (File: /usr/synopsys/libraries/syn/class.db)

Number of ports:     36
Number of nets:      69
Number of cells:      6
Number of references: 6

Combinational area:     653.000000
Noncombinational area:  231.000000
Net Interconnect area:   undefined (Wire load has zero net area)

Total cell area:        884.000000
Total area:
1
```

Use the constraint report to find out whether constraints are met. The constraint report provides information about design rules and optimization constraints. The report includes all violations.

In this constraint report, all the design constraints are met, including the design path delay of 20 ns and the clock period of 25 ns. You define the design path delay of 20 ns by setting the external output delay to 5 ns. Calculate the design path delay by subtracting the output delay from the clock period.

```
*************************************
Report: constraint
Design: TOP
Version: 1998.08
Date:    Thu Oct 15 17:52:27 1998
**************************************


                                            Weighted
Group (max_delay/setup)   Cost    Weight     Cost
----------------------------------------------------
CLK                       0.00    1.00       0.00
default                   0.00    1.00       0.00
-----------------------------------------------------
max_delay/setup                              0.00

                         Total Neg  Critical
Group(critical_range)      Slack    Endpoints Cost
--------------------------------------------------
critical_range                               0.00

                                            Weighted
Group (min_delay/hold)    Cost     Weight    Cost
----------------------------------------------------
CLK (no fix_hold)         0.00      1.00     0.00
default                   0.00      1.00     0.00
-----------------------------------------------------
min_delay/hold                               0.00

Constraint                                    Cost
-----------------------------------------------------
max_delay/setup                                0.00 (MET)
critical_range                                 0.00 (MET)
```

Saving the Newly Compiled Version Using dc_shell

Having tightened timing constraints and increased performance, save this faster version as TOP_compiled.db. Use this design in the next chapter to further demonstrate the reporting and analyzing of design results.

To save the newly compiled version,

- Enter one of the following commands from the command prompt:

  ```
  dc_shell> write -format db -hierarchy -output
   “./db/TOP_compiled.db” {“compile.db:TOP”}
  ```

  or

  ```
  dc_shell-t> write -format db -hierarchy -output
  {./db/TOP_compiled.db} [list {compile.db:TOP}]
  ```

  In response, dc_shell displays the following commentary:

  ```
  Writing to file bohm/tutorial/db/TOP_compiled.db
  1
  ```

Optional Exercises

You can perform optional exercises, using dc_shell to set attributes on specific cells and experiment with alternative ways to resolve multiple instances of a subdesign. The following section provides exercises that set attributes on specific cells.

Note:

> Except for the read command, all commands in these exercises are the same in dcsh and dctcl modes.

Setting Attributes on Specific Cells

An instance is a cell embedded in the hierarchy of a design. In this exercise, you set attributes on a specific cell in the design hierarchy. You use the dc_shell current_instance command to

- Define a current instance of a cell
- Report and alter attributes on that instance
- Move up the hierarchy from that instance

To read in the design,

1. Change to your tutorial directory. At the operating system prompt, enter

   ```
   % cd ~/tutorial
   ```

2. Start the command-line interface. At the operating system prompt, enter

   ```
   % dc_shell
   ```

   or

   ```
   % dc_shell-t
   ```

3. Read in the db file. At the dc_shell prompt, enter

   ```
   dc_shell> read -f db ./db/TOP_before_compile.db
   ```

   or

   ```
   dc_shell-t> read_file -format db [list {./db/
   TOP_before_compile.db}]
   ```

To move down the design hierarchy to instance U7,

1. Set the current design to the top of the hierarchy. At the dc_shell prompt, enter

   ```
   dc_shell> current_design TOP
   ```

   The command is the same in dctcl mode.

2. Change to instance U3, which is CONVERTOR_CKT TOP/U3. At the dc_shell prompt, enter

   ```
   dc_shell> current_instance U3
   ```

   The command is the same in dctcl mode.

3. Change to instance U7, which is CONVERTOR_1 TOP/U3/U7. At the dc_shell prompt, enter

   ```
   dc_shell> current_instance U7
   ```

   The command is the same in dctcl mode.

To change pin capacitance,

1. Before changing the pin capacitance of U7, generate a net report. At the dc_shell prompt, enter

   ```
   dc_shell> report_net
   ```

   The command is the same in dctcl mode.

   The load value for net A0 is 3.53. This value is calculated as the sum of the pin load (3) plus the wire load (0.53).

2. Annotate the capacitance on pin A0 of instance U7. At the dc_shell prompt, enter

```
dc_shell> set_load 2.5 A0
```

The command is the same in dctcl mode.

Generate another net report. At the dc_shell prompt, enter

```
dc_shell> report_net
```

The command is the same in dctcl mode.

The load for net A0 now is 5.50. This value is calculated as the sum of the pin load (3) and the value of 2.5 assigned in the set_load command. The wire load is not included in the load value because the load has changed since the last compile.

To move up the design hierarchy,

1. Display the current instance. At the dc_shell prompt, enter

   ```
   dc_shell> current_instance .
   ```

   The command is the same in dctcl mode.

2. Move up one level in the design hierarchy. At the dc_shell prompt, enter

   ```
   dc_shell> current_instance ..
   ```

   The command is the same in dctcl mode.

3. Move up to the top level of the hierarchy. At the dc_shell prompt, enter

   ```
   dc_shell> current_instance
   ```

   The command is the same in dctcl mode.

4. After reaching the top level, exit dc_shell. At the dc_shell prompt, enter

   ```
   dc_shell> quit
   ```

   The command is the same in dctcl mode.

10

Analyzing Design Results

Throughout the tutorial exercises in the previous chapters, you generate and analyze area, constraints, and timing reports. This chapter describes how to generate and analyze other reports. It explains how to generate bus, cell, net, compile options, and hierarchy reports, analyze circuits by using schematics; report point-to-point timing; and generate a sized schematic for printing.

Allow less than two hours to complete the exercises in this chapter.

This chapter includes the following sections:

- Before You Begin

  Explains how to generate the schematic view, using Design Analyzer.

- Analyzing Design Results Using Design Analyzer

Contains the complete set of tasks and exercises you perform to generate reports and analyze design results, using Design Analyzer. For a list of the exercises, see the introduction to the section.

- Analyzing Design Results Using dc_shell

  Contains the complete set of tasks and exercises you perform—focusing on the commands you use—to generate reports and analyze design results.

Before You Begin

Before undertaking the exercises in this chapter, complete the exercises in Chapter 9, “Compiling a Hierarchical Design.”

If you quit the tutorial after performing the previous exercise, you must first read in the design and generate the schematic view before you can perform the exercises in this chapter.

Generating the Schematic View Using Design Analyzer

To generate the Schematic view for TOP,

1. Start Design Analyzer from your tutorial directory.

   If Design Analyzer is already running, delete any designs in the Designs view.

2. Read in TOP_compiled.db by using File > Read. You optimized this design previously with a clock period of 23 and a delay to the output ports of 18.

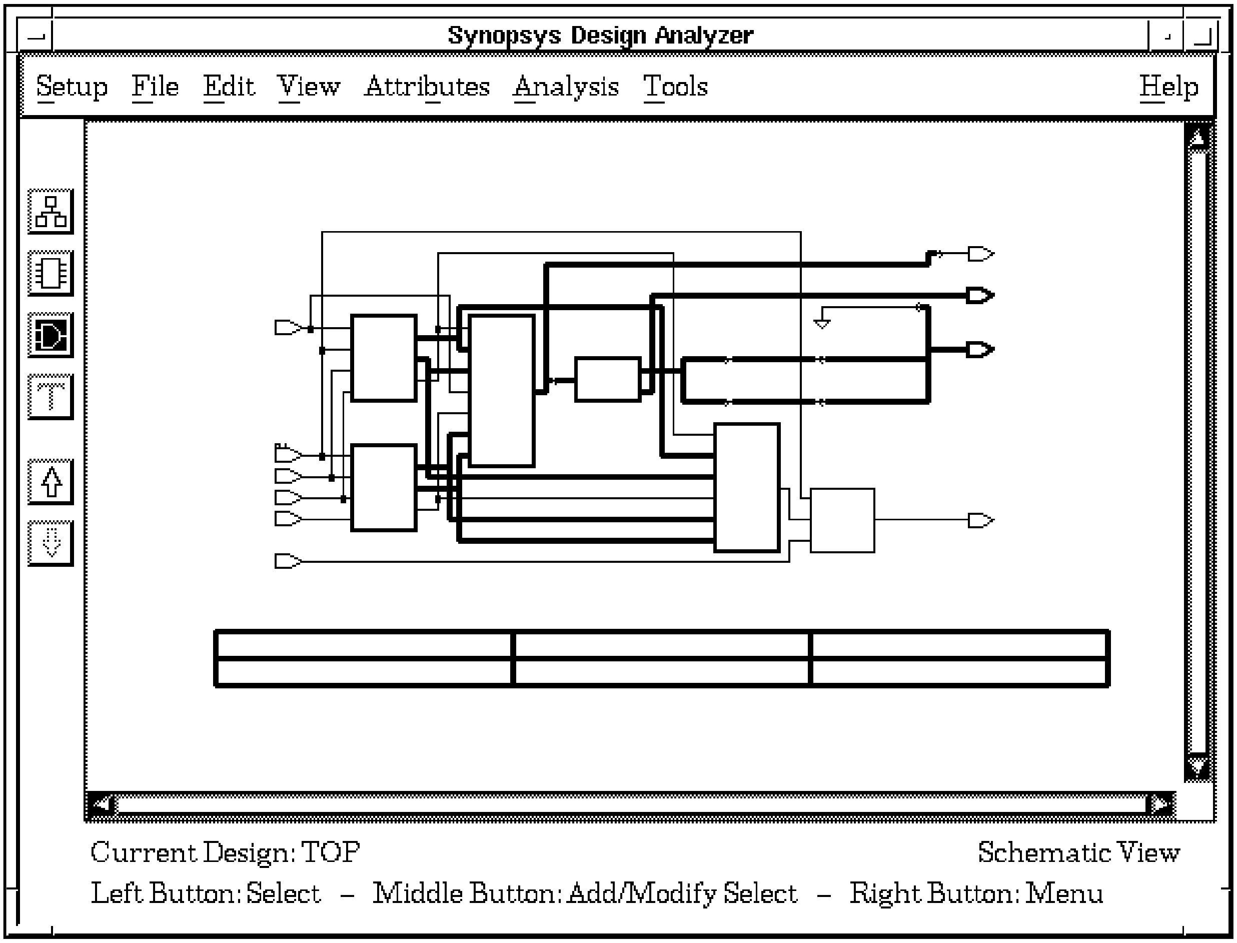
3. Generate the schematic for TOP.

Note:

> You can skip these steps if you did not quit the tutorial after performing the previous exercise.

Figure 10-1 shows the schematic for TOP in schematic view.

Figure 10-1 Schematic View


Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools Help
Current Design: TOP
Schematic View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Analyzing Design Results Using Design Analyzer

This section contains a description of tasks and exercises in the following sections:

- Generating Attribute Reports Using Design Analyzer

  Explains how to generate attribute reports for various types of design objects.

- Generating Analysis Reports Using Design Analyzer

  Explains how to generate hierarchy, timing, and port timing reports.

- Using the Schematic View for Analysis in Design Analyzer

  Explains how to display timing or load values for points in the schematic to check these values instead of generating reports.

- Displaying Multiple Design Analyzer Windows

  Explains how to generate two schematic views and display them at one time.

- Printing Schematics Using Design Analyzer

  Explains how to resize and print a schematic.

Generating Attribute Reports Using Design Analyzer

The reports available under Attribute Reports provide attribute information for various types of design objects. This section describes how to generate attribute reports for the Alarm Clock design in the following sections:

- Bus Report

  Identifies the width of buses. Use for buses and nets.

- Cell Report

  Displays information about cells and subdesigns in the current design. Includes cell (instance) and reference names, their libraries, and the area for each cell.

- Net Report

Provides information about fanout, fanin, loading, attributes, and the number of pins connected to each net.

- Compile Options Report

  Summarizes the compile options applied to the submodules.

The results in your reports and schematics might vary from those displayed.

Bus Report

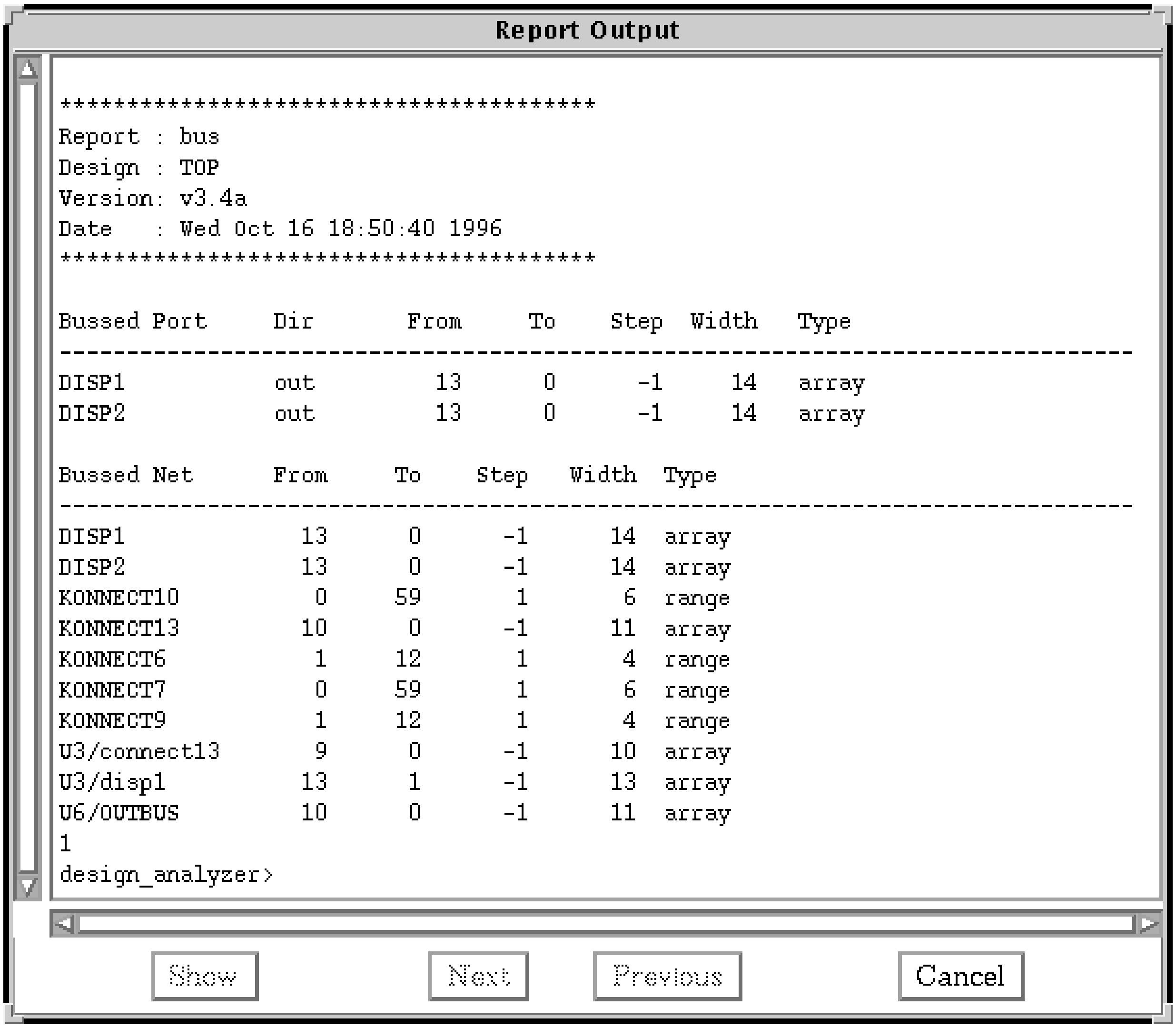
The bus report lists the bused ports and nets in the current design. The information in the report is linked to the schematic.

To generate the bus report,

1. Select Busing in the open Report window.
2. Click Apply.

   The Report Output window, shown in Figure 10-2, appears and displays the bus report.

Figure 10-2   Report Output Window



```
****************************************
Report : bus
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 18:50:40 1996
****************************************

Bussed Port     Dir       From     To    Step  Width   Type
--------------------------------------------------------------------------------
DISP1           out         13      0      -1     14   array
DISP2           out         13      0      -1     14   array

Bussed Net      From     To    Step   Width  Type
--------------------------------------------------------------------------------
DISP1             13      0      -1      14  array
DISP2             13      0      -1      14  array
KONNECT10          0     59       1       6  range
KONNECT13         10      0      -1      11  array
KONNECT6           1     12       1       4  range
KONNECT7           0     59       1       6  range
KONNECT9           1     12       1       4  range
U3/connect13       9      0      -1      10  array
U3/disp1          13      1      -1      13  array
U6/OUTBUS         10      0      -1      11  array
1
design_analyzer>
```

Buses are usually indexed in ascending order. If the bus is in descending order, the numbers in the Step column are negative.

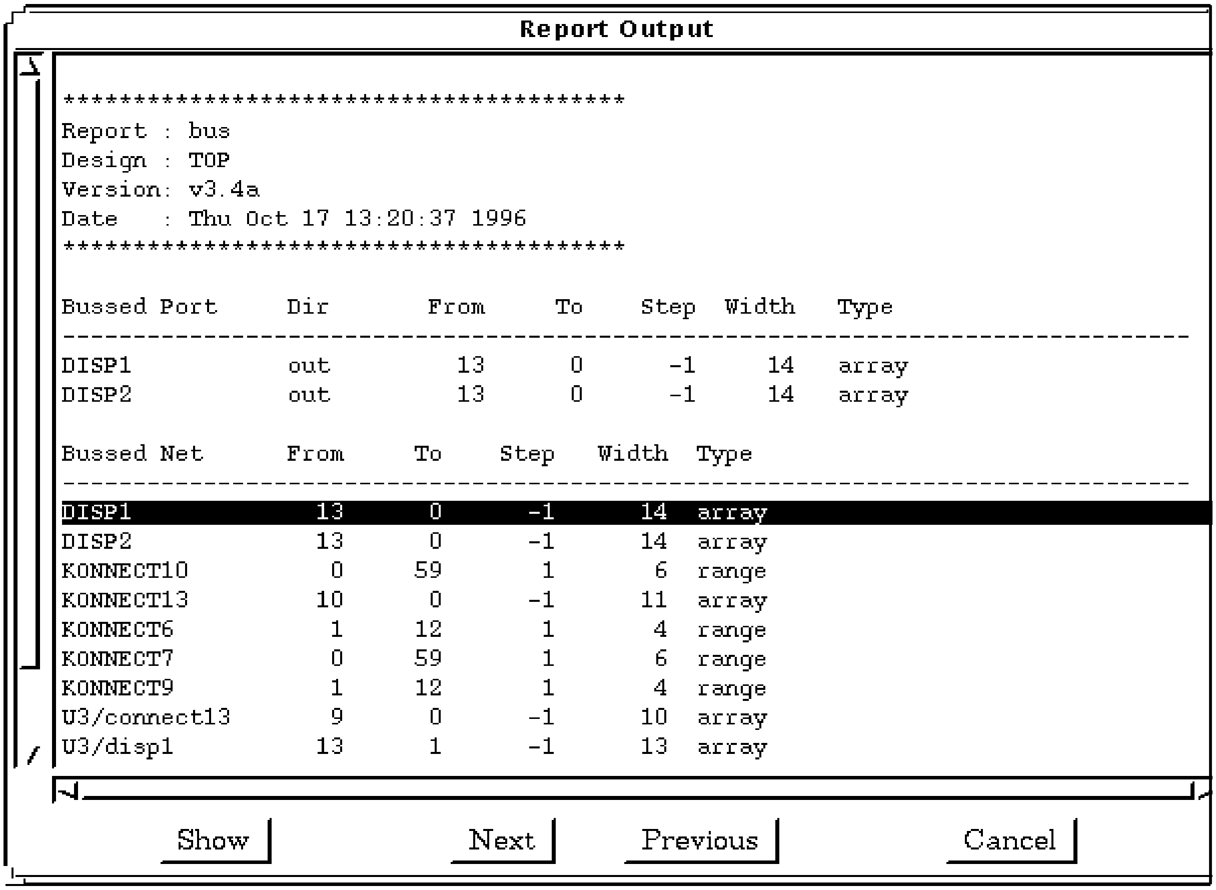
To determine the bused ports locations,

1. Move the Report and Report Output windows next to the Design Analyzer window.

2. Select DISP1 in the Bused Port section.

   Design Analyzer highlights the line, as shown in Figure 10-3.

Figure 10-3 DISP1 in the Bused Port Section

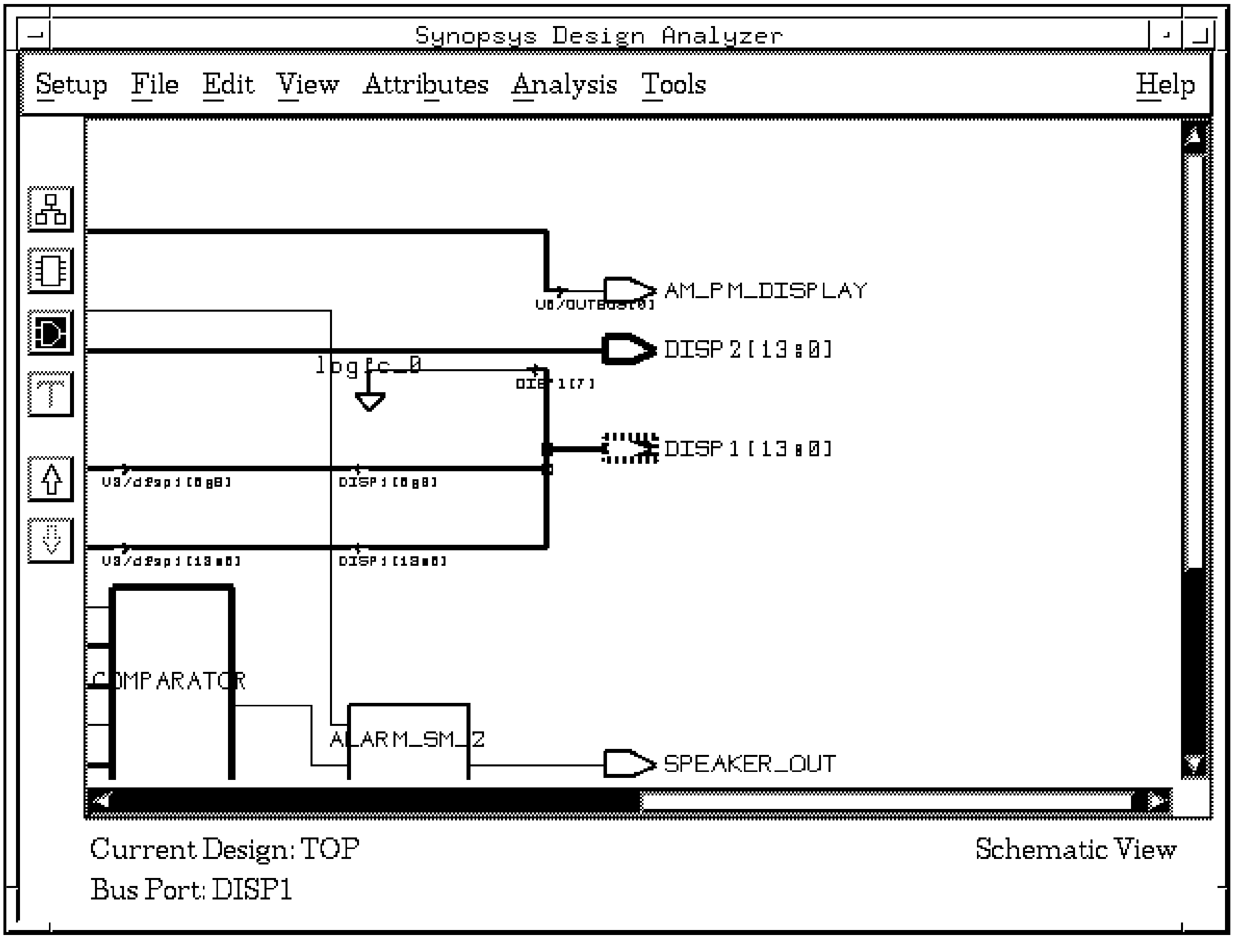


```
****************************************
Report : bus
Design : TOP
Version: v3.4a
Date   : Thu Oct 17 13:20:37 1996
****************************************

Bussed Port     Dir       From     To    Step  Width   Type
--------------------------------------------------------------------------------
DISP1           out         13      0      -1     14   array
DISP2           out         13      0      -1     14   array

Bussed Net      From     To    Step   Width  Type
--------------------------------------------------------------------------------
DISP1             13      0      -1      14  array
DISP2             13      0      -1      14  array
KONNECT10          0     59       1       6  range
KONNECT13         10      0      -1      11  array
KONNECT6           1     12       1       4  range
KONNECT7           0     59       1       6  range
KONNECT9           1     12       1       4  range
U3/connect13       9      0      -1      10  array
U3/disp1          13      1      -1      13  array
```

3. Click Show.

   This action selects and zooms in on the port in the Schematic view, shown in Figure 10-4.

Figure 10-4   Close-Up of the Bused Port in Schematic View



Current Design: TOP

Bus Port: DISP1

Schematic View

4. Click Next to display the schematics of the next bused port and highlight DISP2 in the Schematic View.

Cell Report

The cell report lists cells and subdesigns in the design with

- Reference names
- Source library, if applicable
- Area

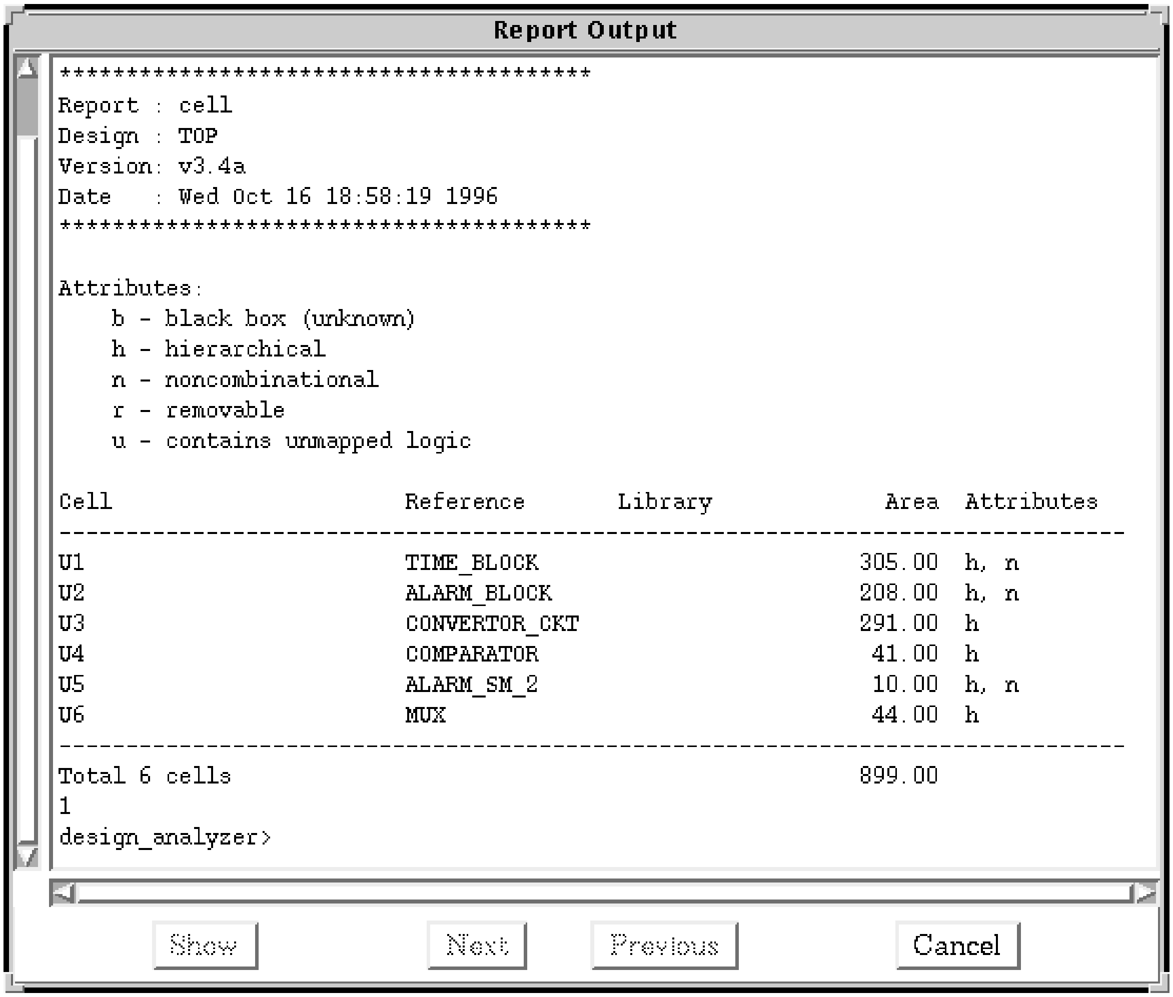
- Attributes

To generate the cell report,

1. Make sure that Busing is not selected in the Report window.
2. Select Cell.
3. Click Apply.

   The cell report appears, as shown in Figure 10-5.

Figure 10-5   Cell Report



```
****************************************
Report : cell
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 18:58:19 1996
****************************************

Attributes:
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                      Reference       Library             Area  Attributes
--------------------------------------------------------------------------------
U1                        TIME_BLOCK                        305.00  h, n
U2                        ALARM_BLOCK                       208.00  h, n
U3                        CONVERTOR_CKT                     291.00  h
U4                        COMPARATOR                         41.00  h
U5                        ALARM_SM_2                         10.00  h, n
U6                        MUX                                44.00  h
--------------------------------------------------------------------------------
Total 6 cells                                               899.00
1
design_analyzer>
```

The TOP design contains only subdesigns, which are listed in the report with cell (instance) and reference names.

Library

Lists the library that contains the reference. This column is empty because these designs are not library cells.

Area

Lists the area for each cell.

Attributes

Lists the design attributes. The Attributes key defines the letters listed in the Attributes column. Subdesigns have an h because they are always hierarchical. Subdesigns that contain sequential elements also have an n, for noncombinational.

Net Report

The net report lists nets in the current design with

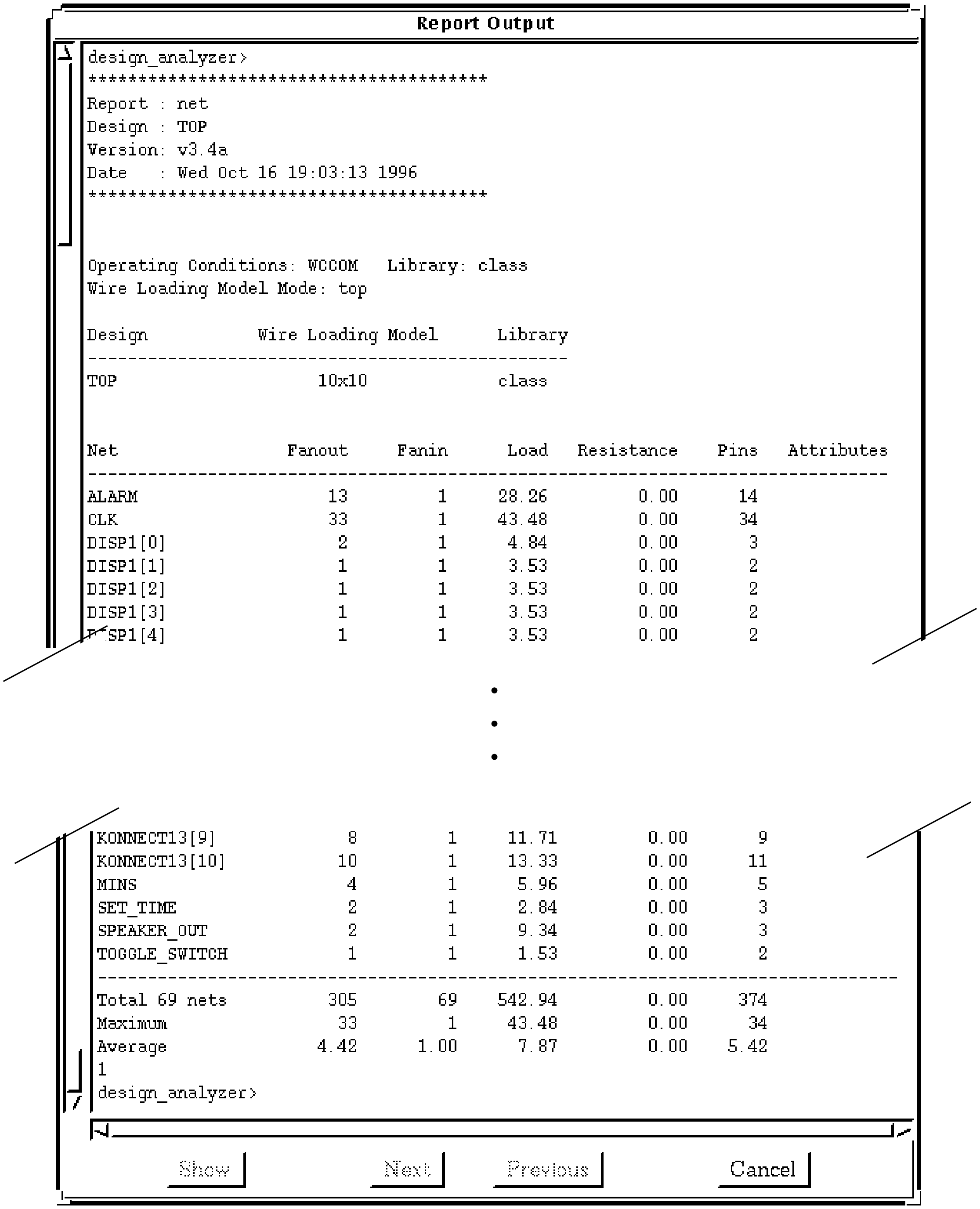
- Fanout
- Fanin
- Load values
- Number of pins
- Attributes for the nets, if any

To generate the net report,

1. Make sure Cell is not selected.
2. Select Net.
3. Click Apply.

   The net report appears, as shown in Figure 10-6.

Figure 10-6   Net Report



```
design_analyzer>
****************************************
Report : net
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 19:03:13 1996
****************************************


Operating Conditions: WCCOM   Library: class
Wire Loading Model Mode: top

Design           Wire Loading Model      Library
------------------------------------------------
TOP                    10x10             class


Net                  Fanout     Fanin      Load   Resistance    Pins   Attributes
--------------------------------------------------------------------------------
ALARM                    13         1     28.26         0.00      14
CLK                      33         1     43.48         0.00      34
DISP1[0]                  2         1      4.84         0.00       3
DISP1[1]                  1         1      3.53         0.00       2
DISP1[2]                  1         1      3.53         0.00       2
DISP1[3]                  1         1      3.53         0.00       2
DISP1[4]                  1         1      3.53         0.00       2

KONNECT13[9]              8         1     11.71         0.00       9
KONNECT13[10]            10         1     13.33         0.00      11
MINS                      4         1      5.96         0.00       5
SET_TIME                  2         1      2.84         0.00       3
SPEAKER_OUT               2         1      9.34         0.00       3
TOGGLE_SWITCH             1         1      1.53         0.00       2
--------------------------------------------------------------------------------
Total 69 nets           305        69    542.94         0.00     374
Maximum                  33         1     43.48         0.00      34
Average                4.42      1.00      7.87         0.00    5.42
1
design_analyzer>
```

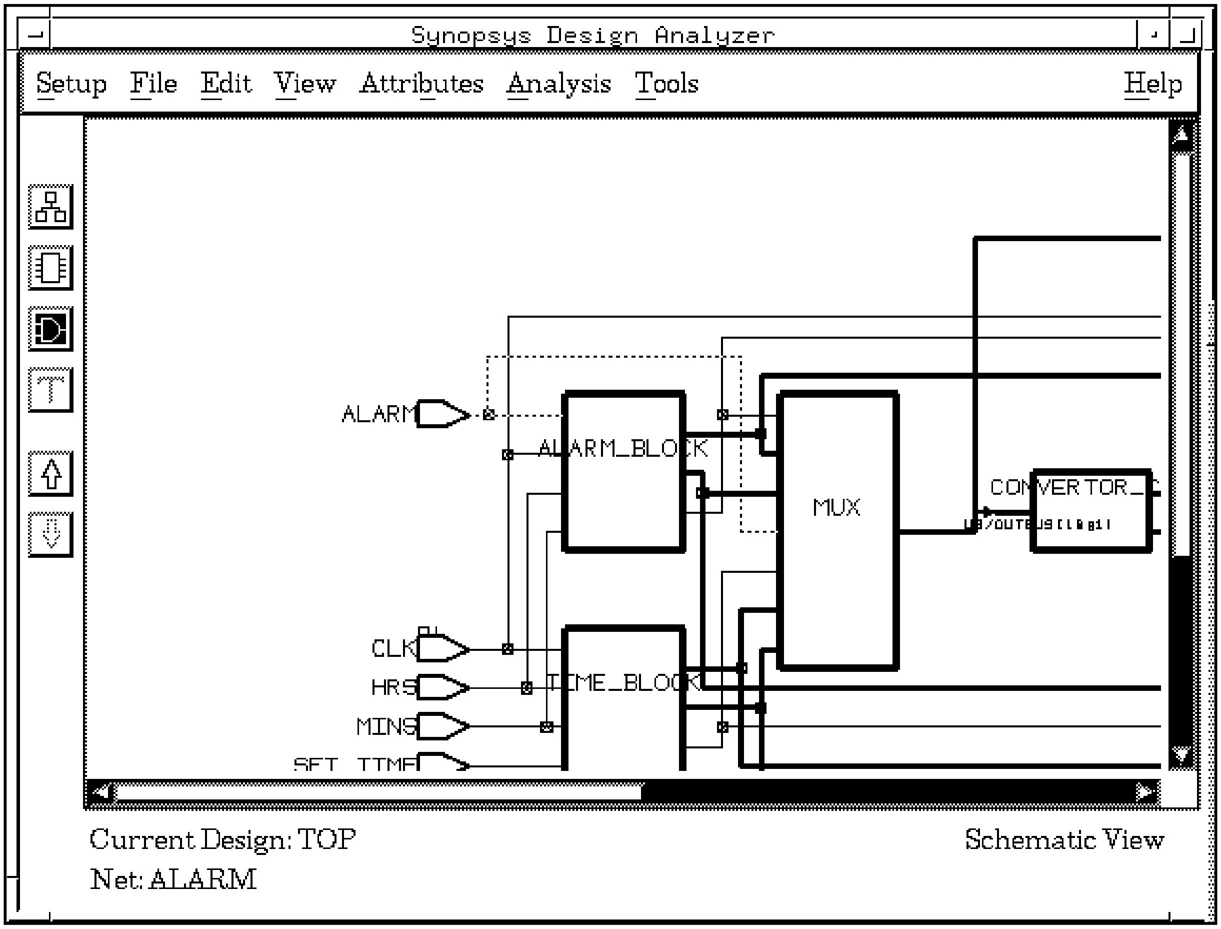
After place and route tools create the circuit layout, the load values of interconnect nets can be derived. You can input load values to Design Compiler to model the synthesized circuit more accurately. This process is called back-annotation. If a net is assigned an annotated capacitance or resistance value, a c or an r appears in the Attributes column.

To view a net in the schematic,

1. Select ALARM in the Report Output window.
2. Click Show.

   This action selects and zooms in on the net in the Schematic View, as shown in Figure 10-7.

Figure 10-7   Schematic View Zoomed In on a Net



Current Design: TOP

Net: ALARM

3. Click Next to display the schematic of the next net in the report (CLK).

Note:

Even though Show and Next are highlighted when you select DISP1[0], you cannot use them to view single bits of a bus.

Compile Options Report

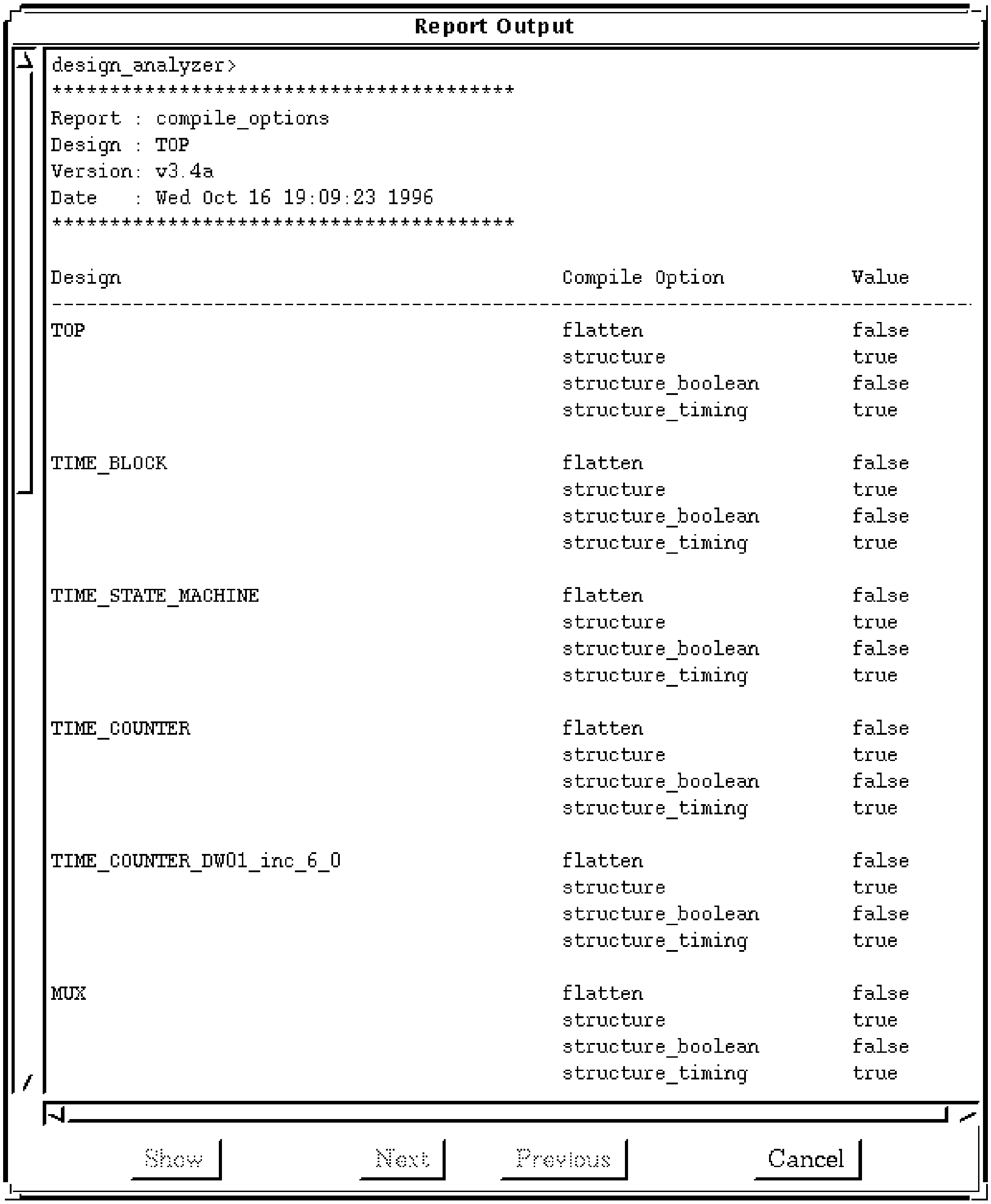
The compile options report lists the options for compile for the current design.

To generate the compile options report,

1. Make sure Net is not selected.
2. Select Compile Options.
3. Click Apply.

   The compile options report appears, as shown in Figure 10-8.

Figure 10-8   Compile Options Report



```
design_analyzer>
****************************************
Report : compile_options
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 19:09:23 1996
****************************************

Design                                       Compile Option          Value
--------------------------------------------------------------------------------
TOP                                          flatten                 false
                                             structure               true
                                             structure_boolean       false
                                             structure_timing        true

TIME_BLOCK                                   flatten                 false
                                             structure               true
                                             structure_boolean       false
                                             structure_timing        true

TIME_STATE_MACHINE                           flatten                 false
                                             structure               true
                                             structure_boolean       false
                                             structure_timing        true

TIME_COUNTER                                 flatten                 false
                                             structure               true
                                             structure_boolean       false
                                             structure_timing        true

TIME_COUNTER_DW01_inc_6_0                    flatten                 false
                                             structure               true
                                             structure_boolean       false
                                             structure_timing        true

MUX                                          flatten                 false
                                             structure               true
                                             structure_boolean       false
                                             structure_timing        true
```

Default compile options were used previously for flatten and structure.

The flatten option is off by default. The structure option is on and is timing-driven. These options are reflected in the report for the subdesigns.

Generating Analysis Reports Using Design Analyzer

The reports available under the Analysis Reports section of the Report window provide information you need in order to analyze the design.

Generate the reports described in these sections:

- Hierarchy Report

  Lists the subdesigns of the hierarchy, including subdesign cells and libraries.

- Timing Report

  Analyzes the critical path, path delays, and incremental delays through cells.

- Point Timing Report

  Analyzes the timing between the points you select.

The results in your reports and schematics might vary from those displayed.

Hierarchy Report

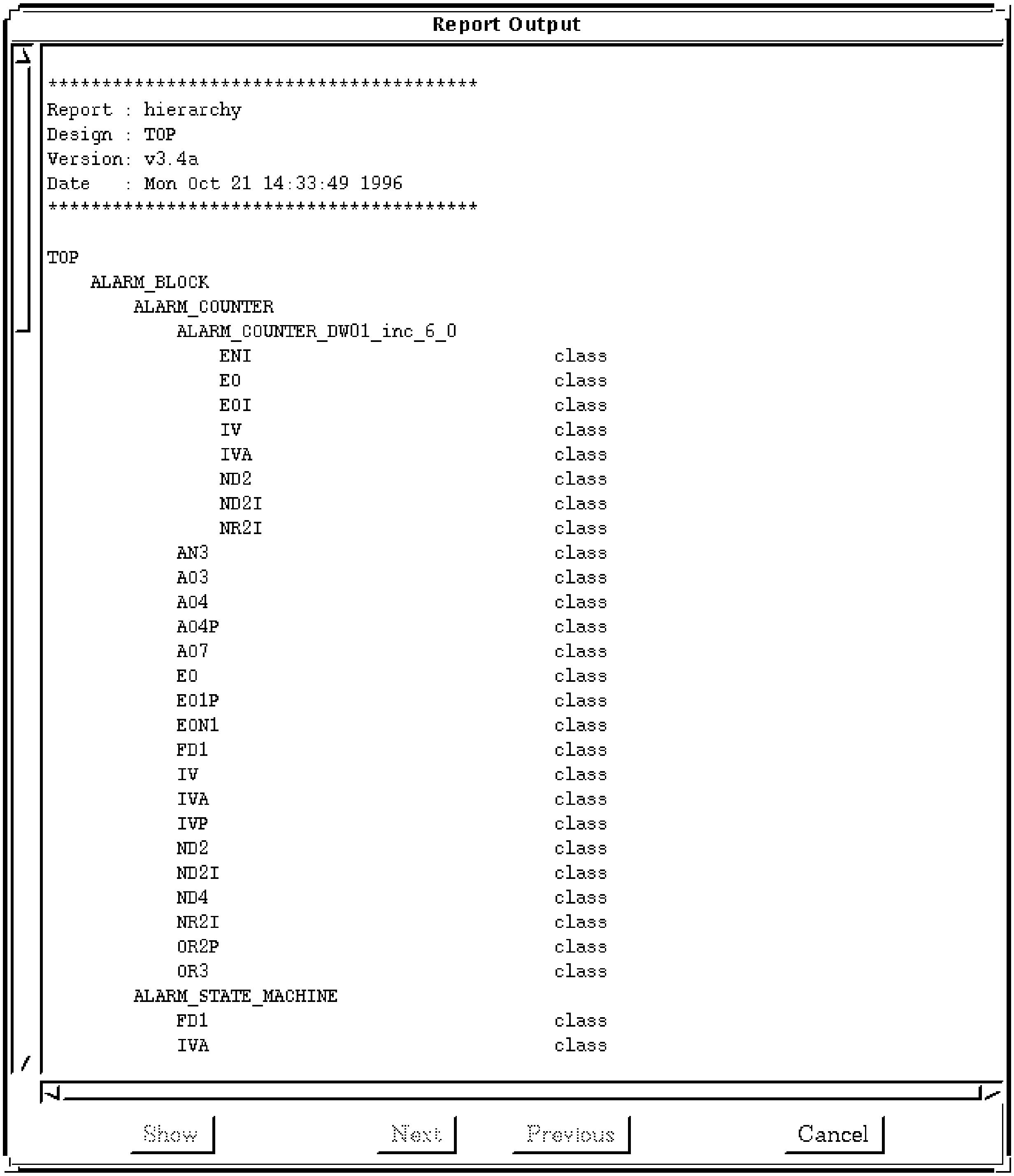
The hierarchy report lists all the cells in the current design or current instance. The hierarchy is shown by using indentations in the report.

To generate a hierarchy report,

1. Click Clear Choices.
2. Select Hierarchy.
3. Click Apply.

   The hierarchy report appears, as shown in Figure 10-9.

Figure 10-9   Hierarchy Report



```
****************************************
Report : hierarchy
Design : TOP
Version: v3.4a
Date   : Mon Oct 21 14:33:49 1996
****************************************

TOP
    ALARM_BLOCK
        ALARM_COUNTER
            ALARM_COUNTER_DW01_inc_6_0
                ENI                            class
                EO                             class
                EOI                            class
                IV                             class
                IVA                            class
                ND2                            class
                ND2I                           class
                NR2I                           class
            AN3                                class
            AO3                                class
            AO4                                class
            AO4P                               class
            AO7                                class
            EO                                 class
            EO1P                               class
            EON1                               class
            FD1                                class
            IV                                 class
            IVA                                class
            IVP                                class
            ND2                                class
            ND2I                               class
            ND4                                class
            NR2I                               class
            OR2P                               class
            OR3                                class
        ALARM_STATE_MACHINE
            FD1                                class
            IVA                                class
```

Timing Report

A timing report provides timing information for defined endpoints and constrained pins. If no endpoints or constrained pins are explicitly specified, the report includes timing information only for the most critical path in the design.

To select the timing report option,

1. Click Clear Choices.
2. Select Timing.

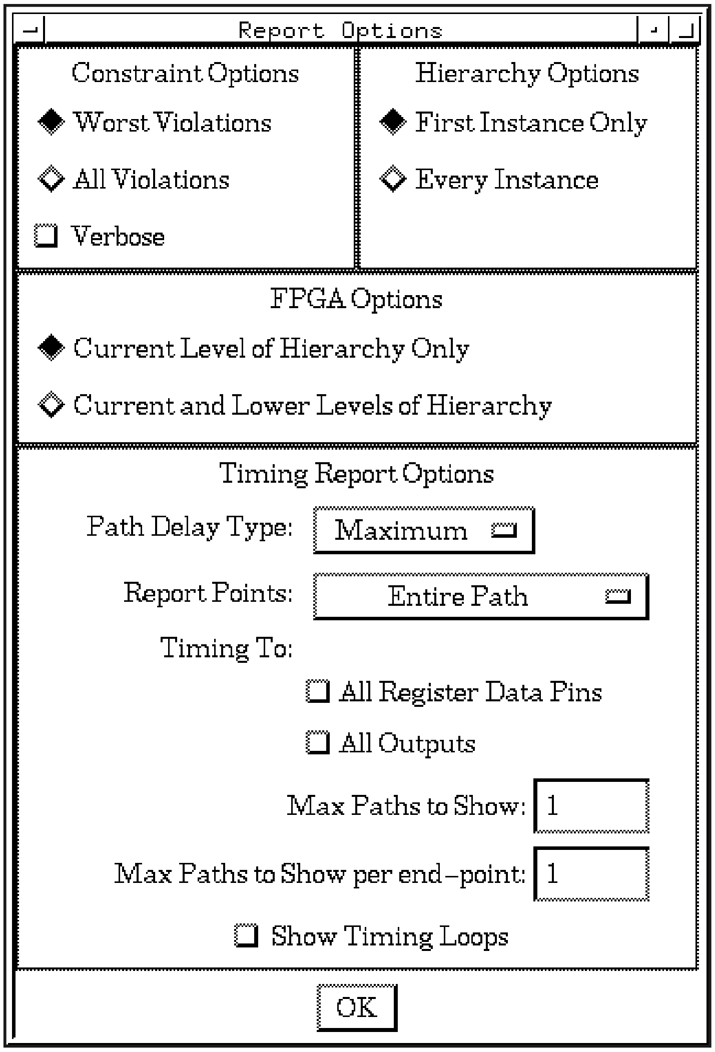
   When you select Timing to be on, Set Options is enabled.

To open the Report Options window,

1. Click Set Options in the Report window.

   The Report Options window appears, as shown in Figure 10-10.

Figure 10-10 Report Options Window



Report Options

Constraint Options

◆ Worst Violations

◇ All Violations

□ Verbose

Hierarchy Options

◆ First Instance Only

◇ Every Instance

FPGA Options

◆ Current Level of Hierarchy Only

◇ Current and Lower Levels of Hierarchy

Timing Report Options

Path Delay Type: Maximum

Report Points: Entire Path

Timing To:

□ All Register Data Pins

□ All Outputs

Max Paths to Show: 1

Max Paths to Show per end-point: 1

□ Show Timing Loops

2. In the Hierarchy Options section of the window, make sure First Instance Only is selected.

   Only the first occurrence of a submodule that is used multiple times in the hierarchy is shown.

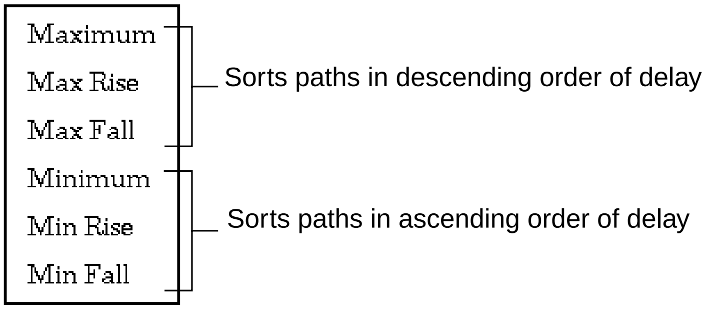
3. In the Constraint Options section of the window, select Verbose.

In the Constraint Options section, Worst Violations and All Violations apply only to the constraint report. Do not use them for timing reports.

The FPGA Options section applies only when the design's target library is an FPGA library.

To select the path delay type,

- In the Timing Report Options section of the window, open the Path Delay Type menu and choose Maximum.

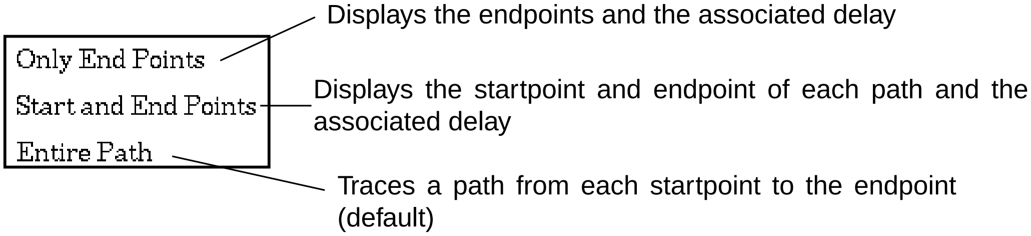


Sorts paths in descending order of delay

Sorts paths in ascending order of delay

To select endpoints,

1. Open the Report Points and choose Entire Path.



Displays the endpoints and the associated delay

Displays the startpoint and endpoint of each path and the associated delay

Traces a path from each startpoint to the endpoint (default)

The rise or fall delay, including the incremental delay from the driver, is displayed for each point in the path.

2. Type 3 in the "Max Paths to Show" field to list the three paths that have the longest delay.

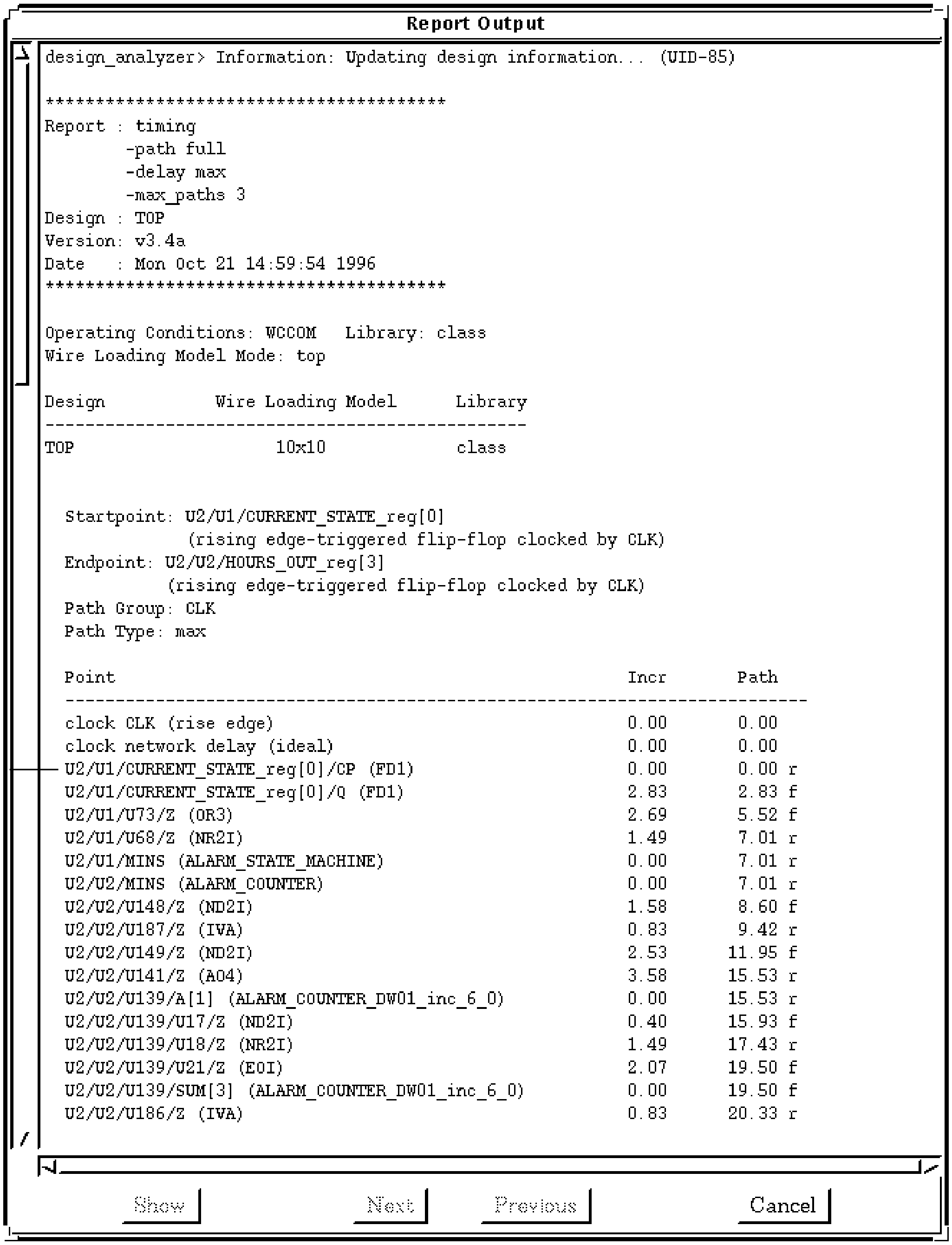
3. Click OK.

   The Report Options window closes.

4. Click Apply in the Report window.

   The timing report, shown in Figure 10-11, appears,. It lists the three most critical paths in the design.

Figure 10-11   Timing Report



```
design_analyzer> Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 3
Design : TOP
Version: v3.4a
Date   : Mon Oct 21 14:59:54 1996
****************************************

Operating Conditions: WCCOM   Library: class
Wire Loading Model Mode: top

Design           Wire Loading Model      Library
------------------------------------------------
TOP                    10x10             class


  Startpoint: U2/U1/CURRENT_STATE_reg[0]
              (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: U2/U2/HOURS_OUT_reg[3]
            (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock CLK (rise edge)                                   0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  U2/U1/CURRENT_STATE_reg[0]/CP (FD1)                     0.00       0.00 r
  U2/U1/CURRENT_STATE_reg[0]/Q (FD1)                      2.83       2.83 f
  U2/U1/U73/Z (OR3)                                       2.69       5.52 f
  U2/U1/U68/Z (NR2I)                                      1.49       7.01 r
  U2/U1/MINS (ALARM_STATE_MACHINE)                        0.00       7.01 r
  U2/U2/MINS (ALARM_COUNTER)                              0.00       7.01 r
  U2/U2/U148/Z (ND2I)                                     1.58       8.60 f
  U2/U2/U187/Z (IVA)                                      0.83       9.42 r
  U2/U2/U149/Z (ND2I)                                     2.53      11.95 f
  U2/U2/U141/Z (AO4)                                      3.58      15.53 r
  U2/U2/U139/A[1] (ALARM_COUNTER_DW01_inc_6_0)            0.00      15.53 r
  U2/U2/U139/U17/Z (ND2I)                                 0.40      15.93 f
  U2/U2/U139/U18/Z (NR2I)                                 1.49      17.43 r
  U2/U2/U139/U21/Z (EOI)                                  2.07      19.50 f
  U2/U2/U139/SUM[3] (ALARM_COUNTER_DW01_inc_6_0)          0.00      19.50 f
  U2/U2/U186/Z (IVA)                                      0.83      20.33 r
```

First Cell in Path

The first cell in the first path is the register

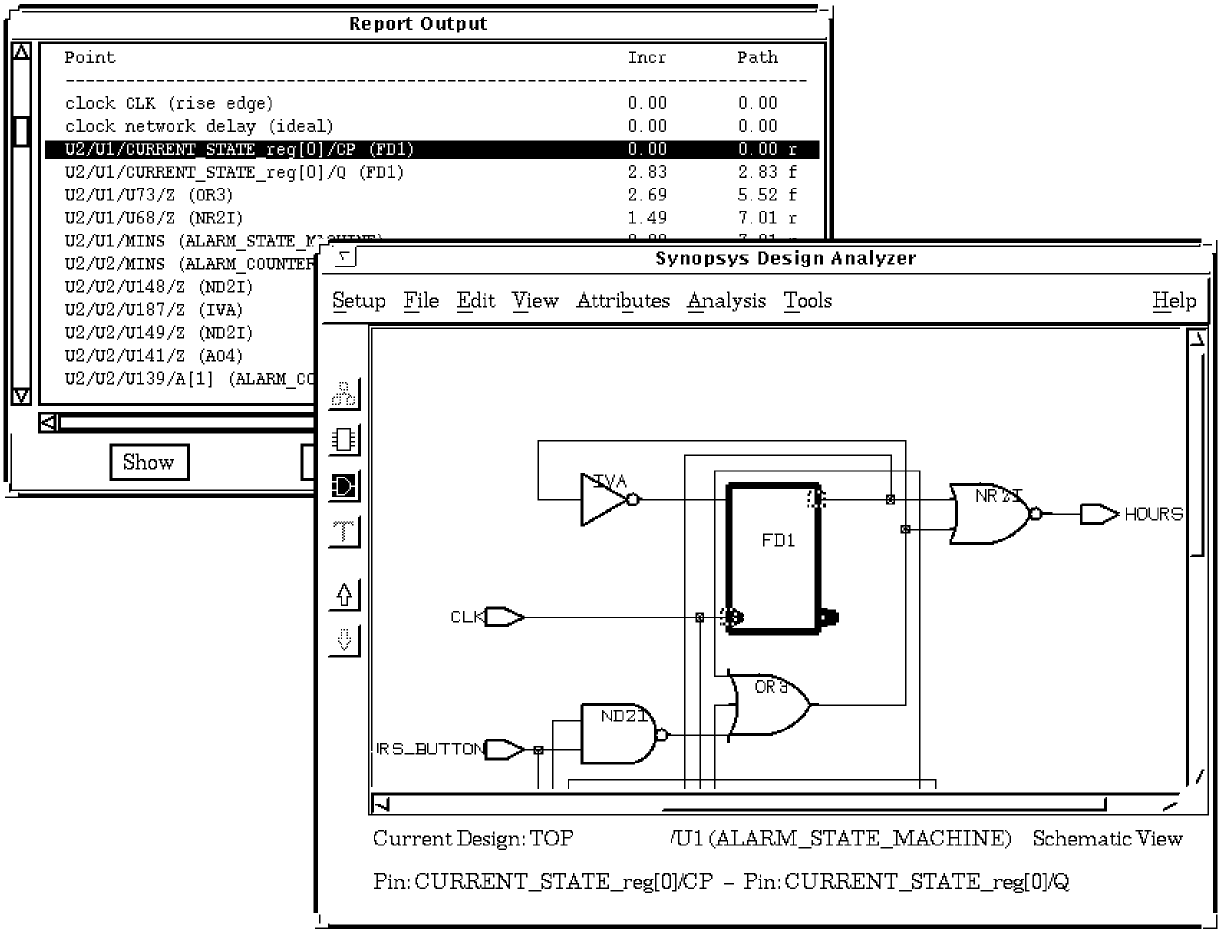
```
U2/U1/CURRENT_STATE_reg[0]/CP
```

You might have identified another path as the most critical in your design.

To trace the critical path,

1. Click the line in the Report Output window that displays the first cell (U2/U1/CURRENT_STATE_reg[0]/CP) in the path.
2. Click Show to select and zoom in on the register (and pin) in the Schematic View, as shown in Figure 10-12. Click Show again to enhance the highlighting.

Figure 10-12   Close Up of a Register and Pin in the Critical Path



Report Output

```
Point                                                    Incr      Path
-------------------------------------------------------------------------
clock CLK (rise edge)                                    0.00      0.00
clock network delay (ideal)                              0.00      0.00
U2/U1/CURRENT_STATE_reg[0]/CP (FD1)                      0.00      0.00 r
U2/U1/CURRENT_STATE_reg[0]/Q (FD1)                       2.83      2.83 f
U2/U1/U73/Z (OR3)                                        2.69      5.52 f
U2/U1/U68/Z (NR2I)                                       1.49      7.01 r
U2/U1/MINS (ALARM_STATE_M
U2/U2/MINS (ALARM_COUNTER
U2/U2/U148/Z (ND2I)
U2/U2/U187/Z (IVA)
U2/U2/U149/Z (ND2I)
U2/U2/U141/Z (A04)
U2/U2/U139/A[1] (ALARM_CO
```

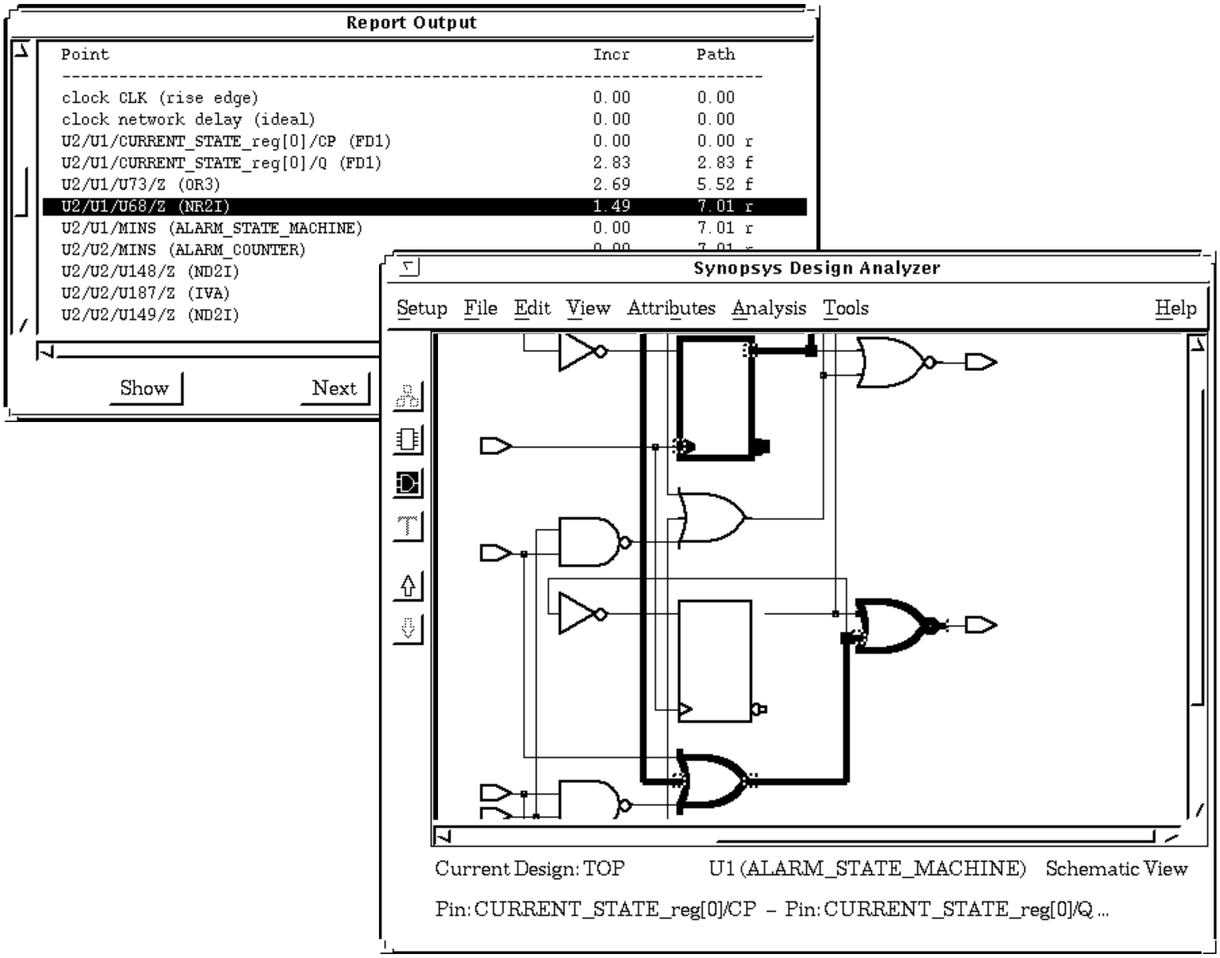
Synopsys Design Analyzer

Current Design: TOP   /U1 (ALARM_STATE_MACHINE)   Schematic View

Pin: CURRENT_STATE_reg[0]/CP – Pin: CURRENT_STATE_reg[0]/Q

3. Click Next three times to highlight the next three points in the path, as shown in Figure 10-13.

Figure 10-13   Critical Path Trace



Report Output

```
Point                                                 Incr       Path
-------------------------------------------------------------------------
clock CLK (rise edge)                                 0.00       0.00
clock network delay (ideal)                           0.00       0.00
U2/U1/CURRENT_STATE_reg[0]/CP (FD1)                   0.00       0.00 r
U2/U1/CURRENT_STATE_reg[0]/Q (FD1)                    2.83       2.83 f
U2/U1/U73/Z (OR3)                                     2.69       5.52 f
U2/U1/U68/Z (NR2I)                                    1.49       7.01 r
U2/U1/MINS (ALARM_STATE_MACHINE)                      0.00       7.01 r
U2/U2/MINS (ALARM_COUNTER)                            0.00       7.01 r
U2/U2/U148/Z (ND2I)
U2/U2/U187/Z (IVA)
U2/U2/U149/Z (ND2I)
```

Synopsys Design Analyzer

Current Design: TOP   U1 (ALARM_STATE_MACHINE)   Schematic View

Pin: CURRENT_STATE_reg[0]/CP – Pin: CURRENT_STATE_reg[0]/Q ...

To continue tracing the critical path,

1. Continue examining the critical path, using Next and Show.
2. When you finish following the critical path, click Cancel.
3. Choose Analysis > Highlight > Clear to remove the highlighting on the path in the Schematic view.

   You can also remove the highlighting on the critical path by placing the pointer in the Design Analyzer window and pressing Ctrl-h.

For more information about types of reports and report options, see the *Design Analyzer Reference Manual* and the report commands in the man pages.

Point Timing Report

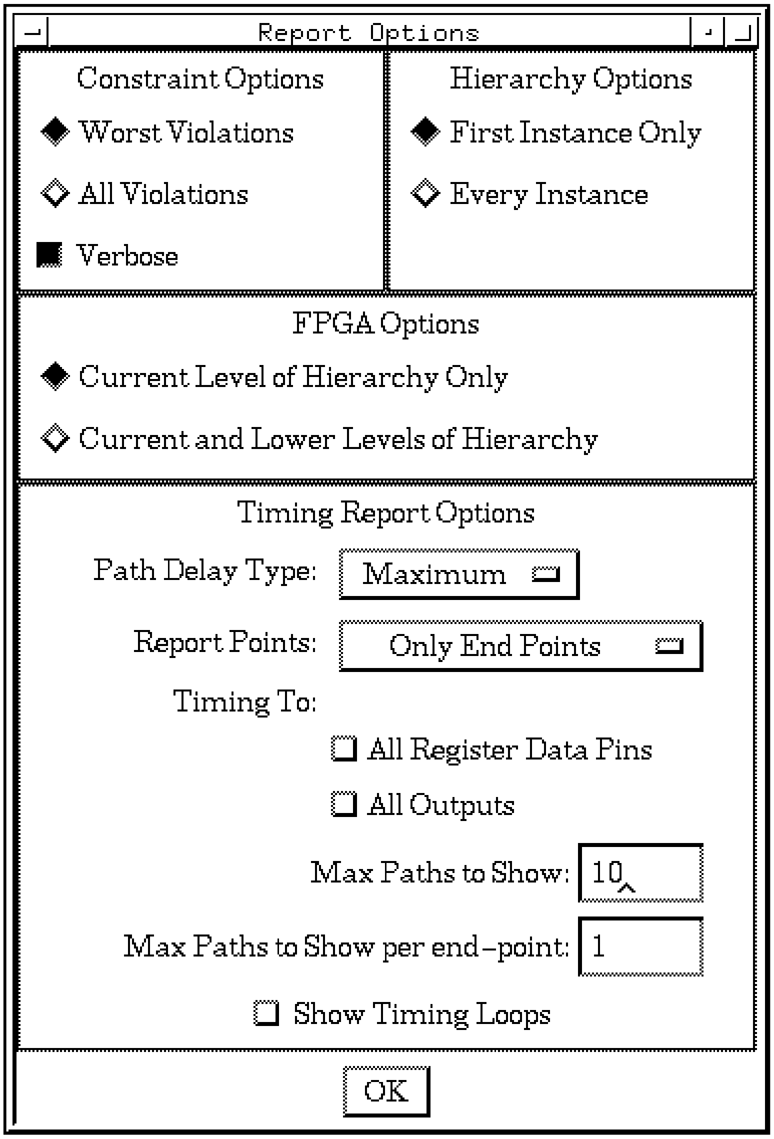
The timing report provides information for defined startpoints, endpoints, and constrained pins and ports. If no endpoints are defined and there are no constrained pins or ports, the report includes all output ports. In this exercise, you generate a report on timing between selected points as ports, pins, or nets in the design.

To open the Report and Report Options windows,

1. Return to the full view of the TOP schematic.
2. Choose Analysis > Report.

   The Report window opens.
3. Select Timing in the Report window.
4. Click Set Options to open the Report Options window, shown in Figure 10-14.

Figure 10-14 Report Options Window



Report Options

Constraint Options

◆ Worst Violations

◇ All Violations

■ Verbose

Hierarchy Options

◆ First Instance Only

◇ Every Instance

FPGA Options

◆ Current Level of Hierarchy Only

◇ Current and Lower Levels of Hierarchy

Timing Report Options

Path Delay Type: Maximum

Report Points: Only End Points

Timing To:

□ All Register Data Pins

□ All Outputs

Max Paths to Show: 10

Max Paths to Show per end-point: 1

□ Show Timing Loops

To select timing report options,

1. Open the Path Delay Type menu and choose Maximum.
2. Open the Report Points menu and choose Only End Points.
3. Set "Max Paths to Show" to 10.

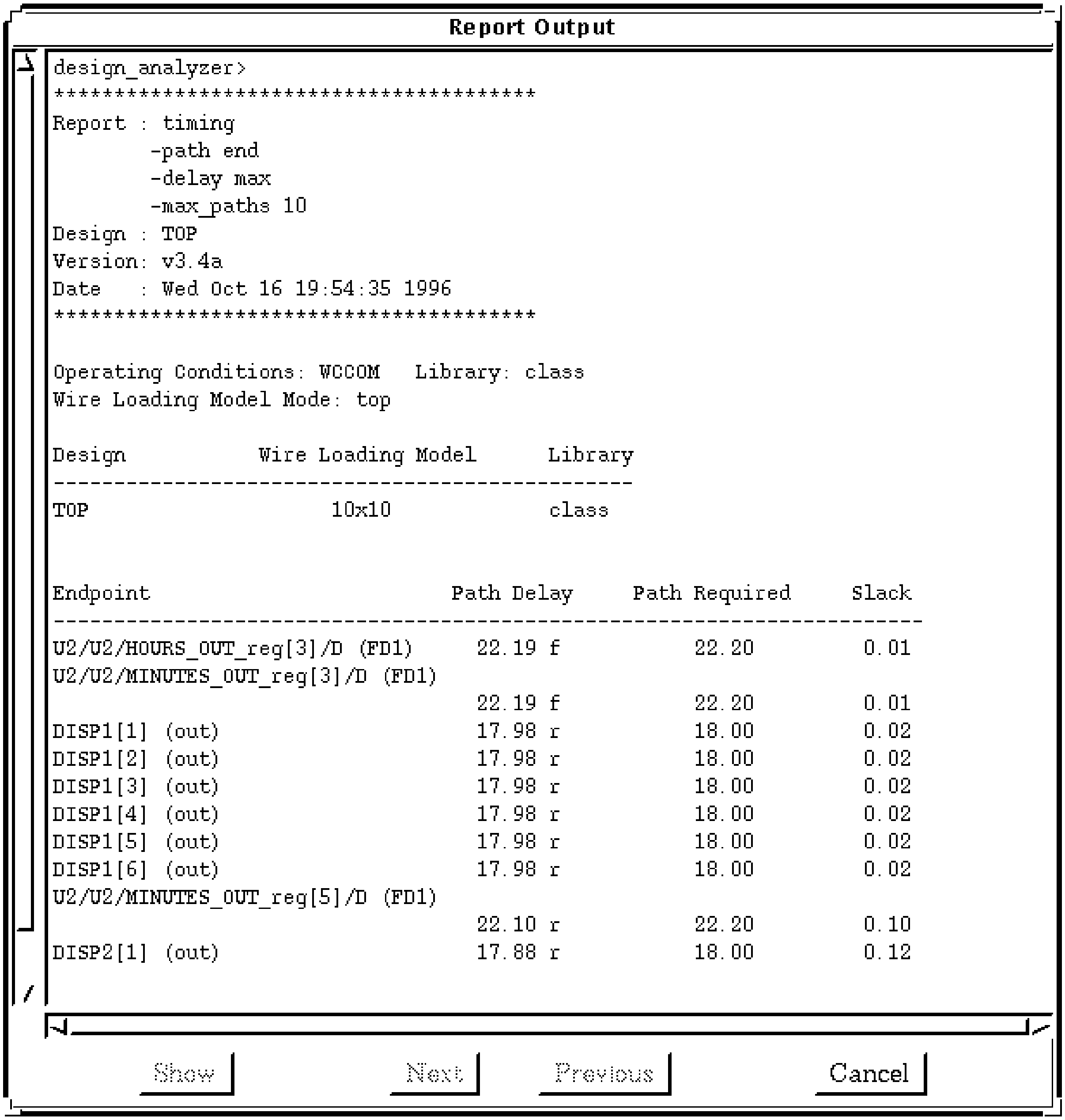
4. Set "Max Paths to Show per end-point" to 1.
5. Click OK.

To generate the timing report,

1. In the Report window, select "Send Output To:" to Window.
2. Click Apply.

   The timing report, shown in Figure 10-15, appears.

Figure 10-15   Timing Report



```
design_analyzer>
****************************************
Report : timing
        -path end
        -delay max
        -max_paths 10
Design : TOP
Version: v3.4a
Date   : Wed Oct 16 19:54:35 1996
****************************************

Operating Conditions: WCCOM   Library: class
Wire Loading Model Mode: top

Design           Wire Loading Model      Library
------------------------------------------------
TOP                    10x10             class


Endpoint                         Path Delay     Path Required     Slack
------------------------------------------------------------------------
U2/U2/HOURS_OUT_reg[3]/D (FD1)     22.19 f          22.20          0.01
U2/U2/MINUTES_OUT_reg[3]/D (FD1)
                                   22.19 f          22.20          0.01
DISP1[1] (out)                     17.98 r          18.00          0.02
DISP1[2] (out)                     17.98 r          18.00          0.02
DISP1[3] (out)                     17.98 r          18.00          0.02
DISP1[4] (out)                     17.98 r          18.00          0.02
DISP1[5] (out)                     17.98 r          18.00          0.02
DISP1[6] (out)                     17.98 r          18.00          0.02
U2/U2/MINUTES_OUT_reg[5]/D (FD1)
                                   22.10 r          22.20          0.10
DISP2[1] (out)                     17.88 r          18.00          0.12
```

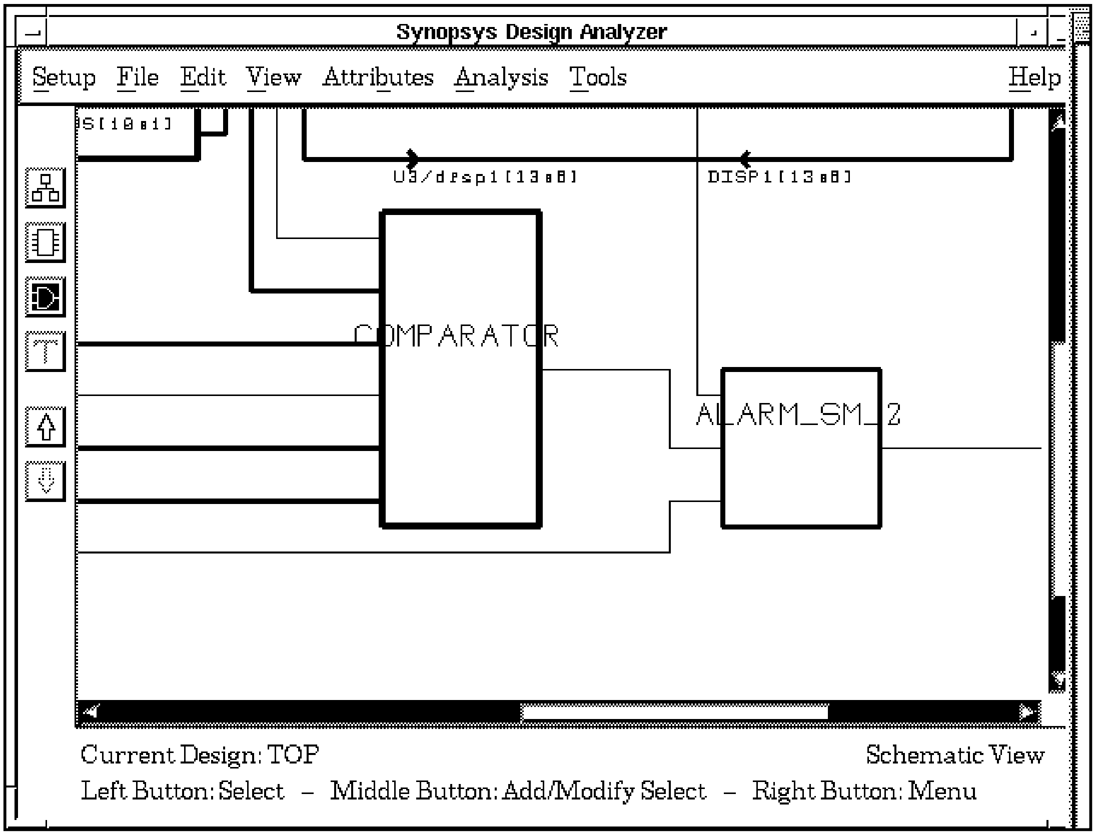
By default, the timing endpoints that Design Compiler reports are the flip-flop D pins and output ports. The D pins have setup constraints derived because of the clock period constraint. The output ports have the max_delay constraint.

You can determine the time when a signal arrives at pins other than those reported by the default option by completing the following tasks:

To zoom in on an area,

1. Display the Schematic view of TOP.
2. Zoom in on the part of the schematic for TOP that contains COMPARATOR and ALARM_SM_2, as shown in Figure 10-16.

Figure 10-16   Schematic View of TOP



Current Design: TOP   Schematic View

Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

To display the pin names layer,

1. Choose View > Style to display the View Style window.
2. Select pin_name_layer from the list.
3. Click Visible option to On.
4. Click Apply to turn on the pin_name_layer.
5. Click Cancel to close the View Style window.

To select two input pins,

1. Select input pin COMPARE_IN of subdesign ALARM_SM_2 (U5/COMPARE_IN).

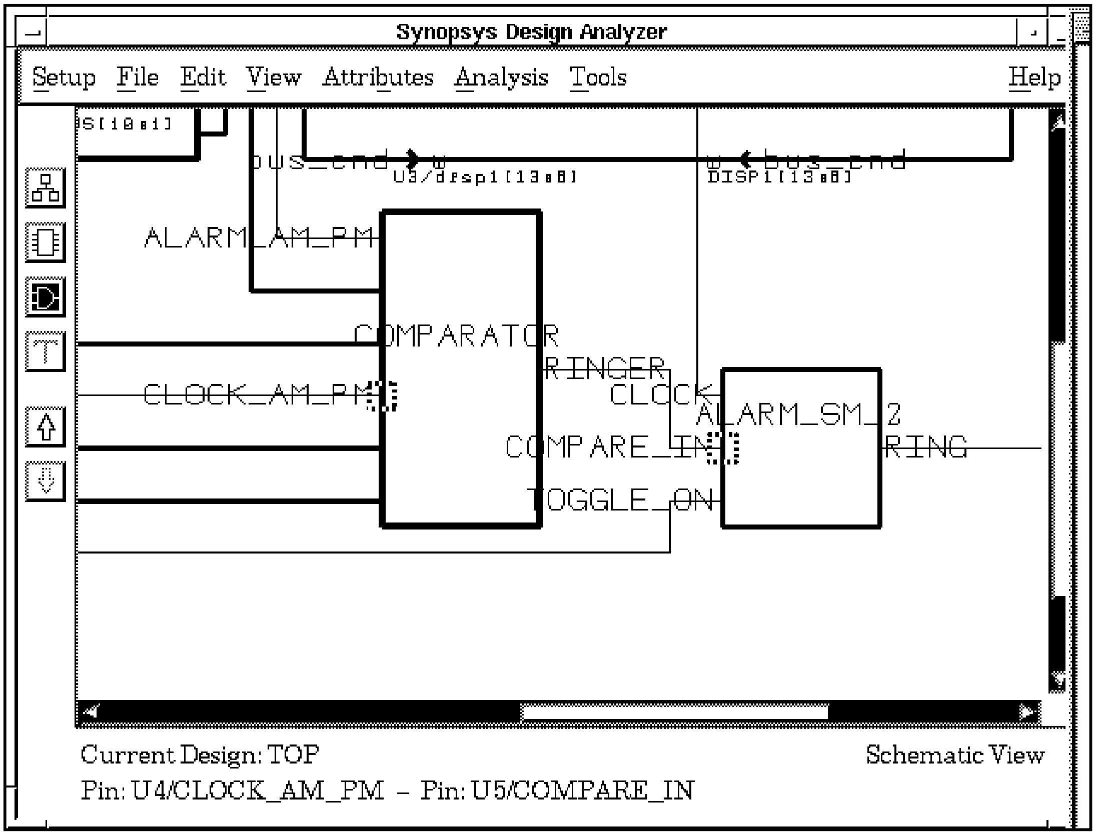
   Pin COMPARE_IN receives input from the COMPARATOR block.

2. Make sure the pin is selected instead of the net. Check the lower left corner of the Design Analyzer window.

3. Select input pin CLOCK_AM_PM of subdesign COMPARATOR (U4/CLOCK_AM_PM) with the middle mouse button.

   A startpoint and an endpoint are selected for a path, as shown in Figure 10-17.

Figure 10-17   Startpoint and Endpoint of the Path



Current Design: TOP

Schematic View

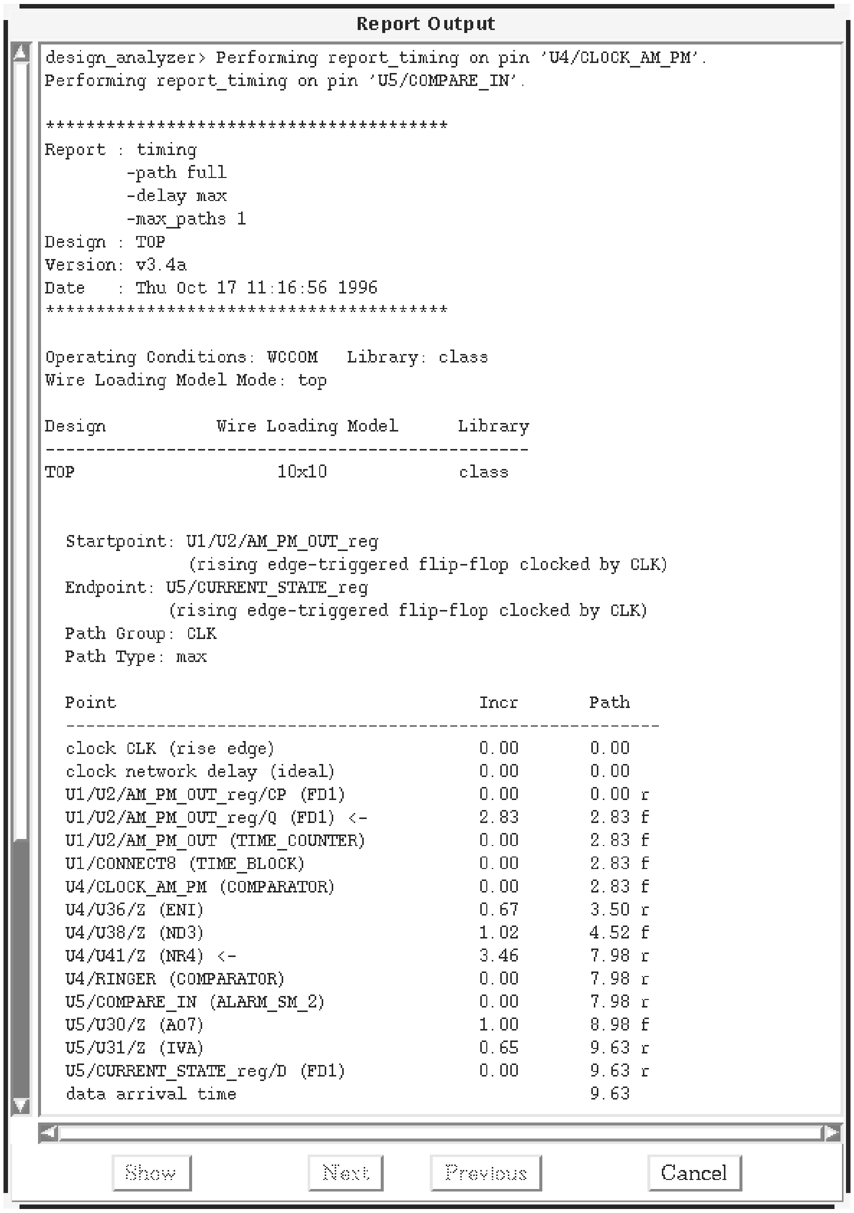
Pin: U4/CLOCK_AM_PM – Pin: U5/COMPARE_IN

To generate a point timing report,

1. Click Clear Choices in the Report window.
2. Select Point Timing.
3. Click Set Options to open the Report Options window.
4. Open the Report Points menu and choose Entire Path.
5. Click OK in the Report Options window.
6. Click Apply in the Report window.

Design Analyzer displays the timing report for the path between the selected points, as shown in Figure 10-18.

Figure 10-18 Timing Report for the Identified Path



```
design_analyzer> Performing report_timing on pin 'U4/CLOCK_AM_PM'.
Performing report_timing on pin 'U5/COMPARE_IN'.

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : TOP
Version: v3.4a
Date   : Thu Oct 17 11:16:56 1996
****************************************

Operating Conditions: WCCOM   Library: class
Wire Loading Model Mode: top

Design           Wire Loading Model      Library
------------------------------------------------
TOP                   10x10             class


  Startpoint: U1/U2/AM_PM_OUT_reg
              (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: U5/CURRENT_STATE_reg
            (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock CLK (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  U1/U2/AM_PM_OUT_reg/CP (FD1)             0.00       0.00 r
  U1/U2/AM_PM_OUT_reg/Q (FD1) <-           2.83       2.83 f
  U1/U2/AM_PM_OUT (TIME_COUNTER)           0.00       2.83 f
  U1/CONNECT8 (TIME_BLOCK)                 0.00       2.83 f
  U4/CLOCK_AM_PM (COMPARATOR)              0.00       2.83 f
  U4/U36/Z (ENI)                           0.67       3.50 r
  U4/U38/Z (ND3)                           1.02       4.52 f
  U4/U41/Z (NR4) <-                        3.46       7.98 r
  U4/RINGER (COMPARATOR)                   0.00       7.98 r
  U5/COMPARE_IN (ALARM_SM_2)               0.00       7.98 r
  U5/U30/Z (AO7)                           1.00       8.98 f
  U5/U31/Z (IVA)                           0.65       9.63 r
  U5/CURRENT_STATE_reg/D (FD1)             0.00       9.63 r
  data arrival time                                   9.63
```

The startpoint and endpoint of the path are denoted by the <-. The requested subpath starts with cell U1/U2/AM_PM_OUT_reg/Q (driving pin U4/CLOCK_AM_PM) and ends at cell U4/U41/Z (driving pin U5/COMPARE_IN). The full path through the pins is listed.

When you are finished examining the report, click Cancel in the Report Output window and Report window.

Using the Schematic View for Analysis in Design Analyzer

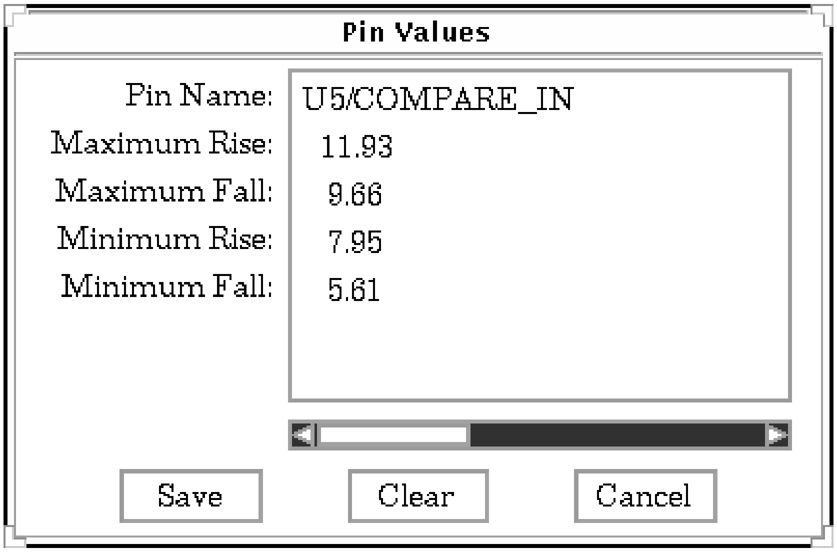
It is not always necessary to generate reports to obtain timing or load values for points in the schematic. You can display these values by selecting a pin on the schematic and requesting its timing.

To determine the time a signal arrives,

1. Select pin COMPARE_IN in the Schematic view.
2. Click the right mouse button in the Schematic view and choose Show Timing in the menu.

   The Pin Values window, shown in Figure 10-19, appears and displays timing information.

Figure 10-19 Pin Values Window



Pin Values

Pin Name: U5/COMPARE_IN
Maximum Rise: 11.93
Maximum Fall: 9.66
Minimum Rise: 7.95
Minimum Fall: 5.61

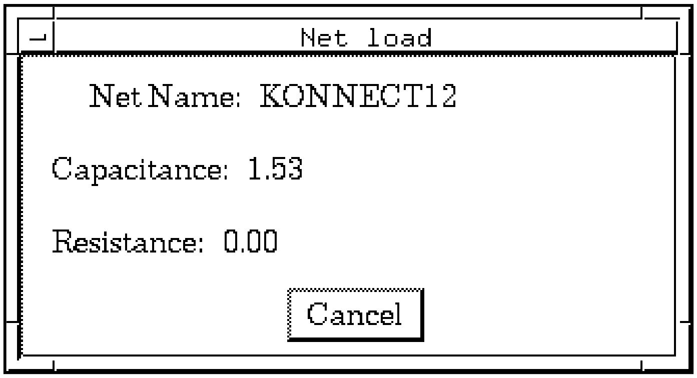
3. Click Cancel in the Pin Values window.

To determine the load on a net,

1. Select the net connecting COMPARE_IN to the COMPARATOR block.
2. Choose Analysis > Show Net Load.

   The Net Load window shown in Figure 10-20 appears. The load is 1.53.

Figure 10-20 Net Load Window


Net load

Net Name: KONNECT12

Capacitance: 1.53

Resistance: 0.00

3. Click Cancel to close the Net Load window.

Displaying Multiple Design Analyzer Windows

You can display multiple views of a schematic at one time.

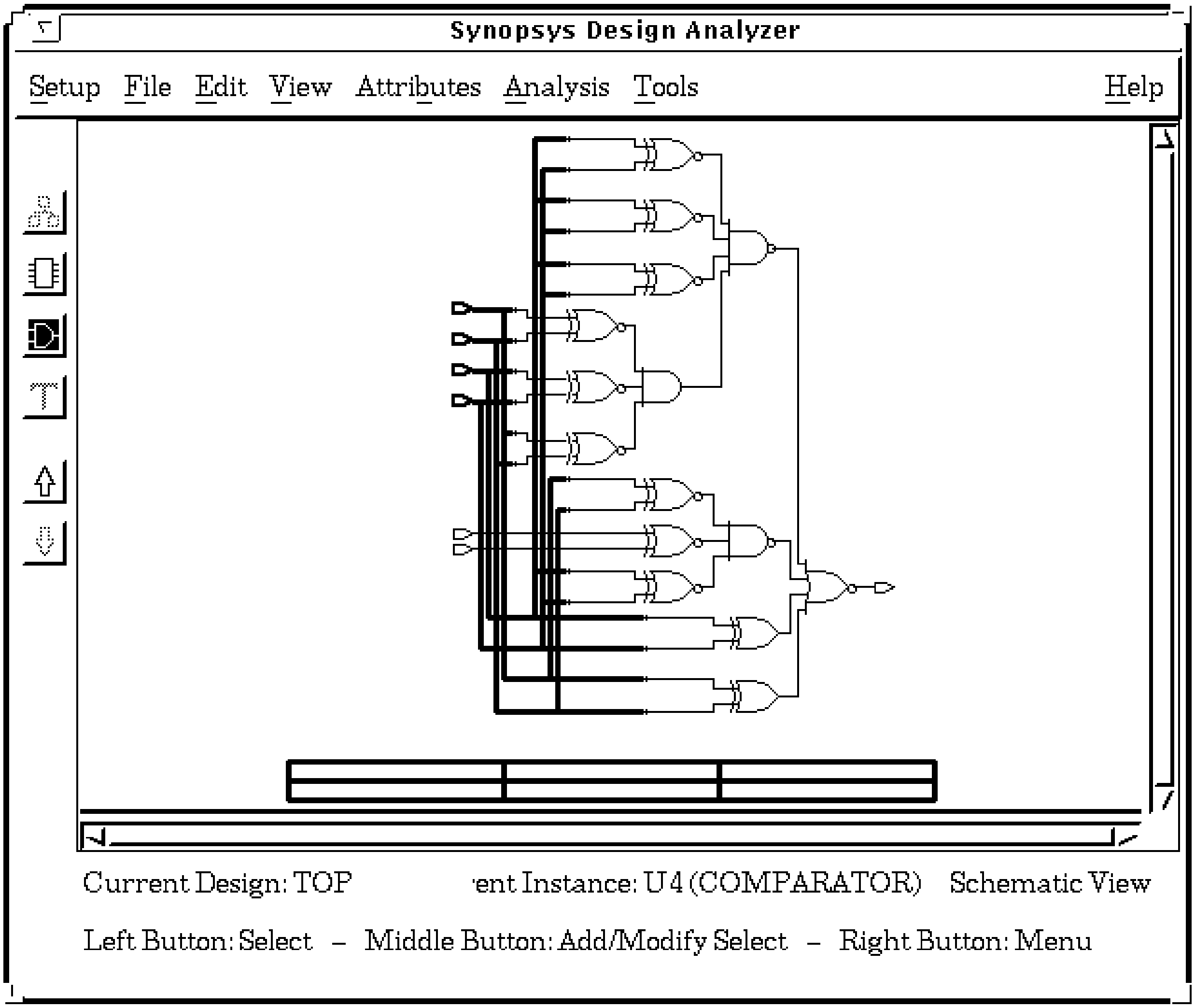
To generate two schematic views,

1. Zoom in on COMPARATOR and ALARM_SM_2 in the Schematic view of TOP design.
2. Choose View > New View.

   Design Analyzer opens a copy of the current Design Analyzer window.

3. Generate the Schematic view for COMPARATOR in the new window, shown in Figure 10-21.

Figure 10-21   Schematic View for COMPARATOR



Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools Help
Current Design: TOP
ent Instance: U4 (COMPARATOR)
Schematic View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

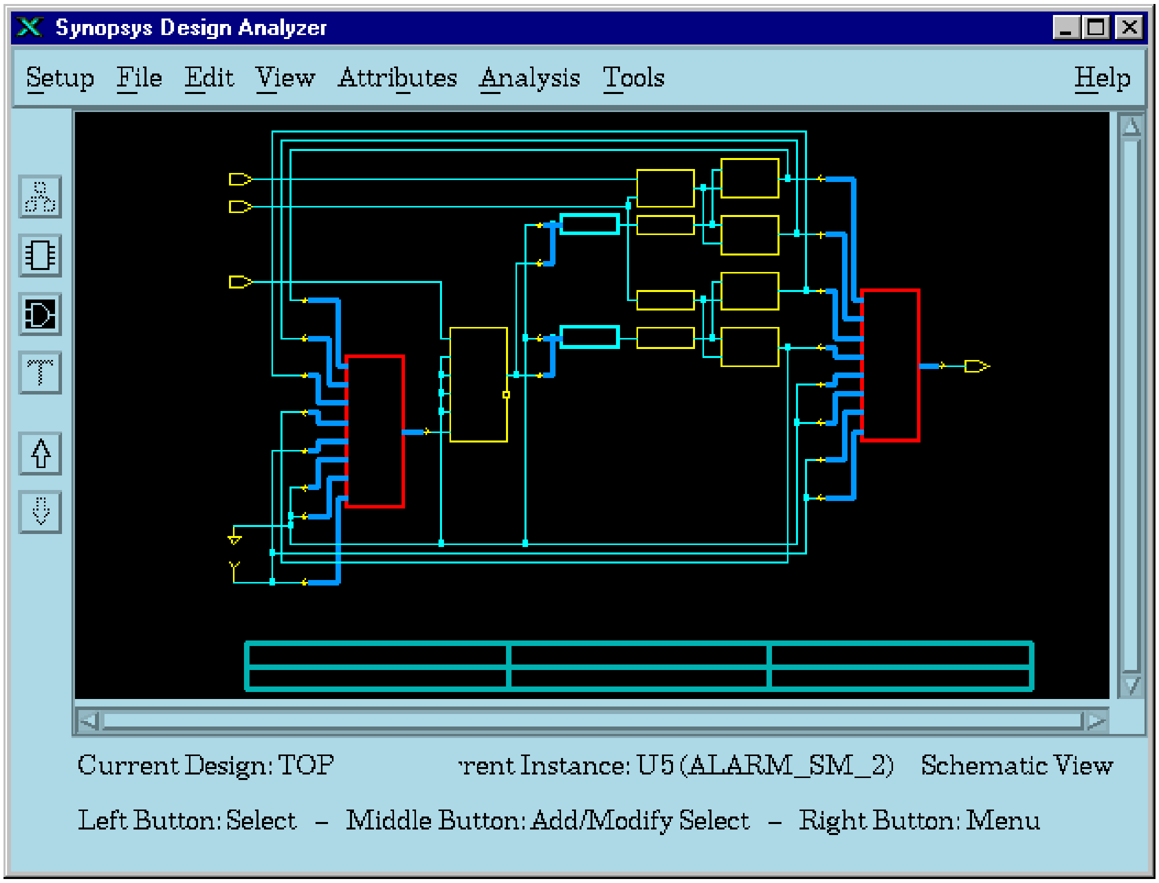
Both windows view the same database. Changes to the database are reflected in all current Design Analyzer windows.

4. Generate the schematic view for ALARM_SM_2 in the other window.

   Using the two windows, you can display different views for the same design at one time.

Figure 10-22 shows the schematic view for ALARM_SM_2 for UNIX.

Figure 10-22 Schematic View for ALARM_SM_2 in UNIX

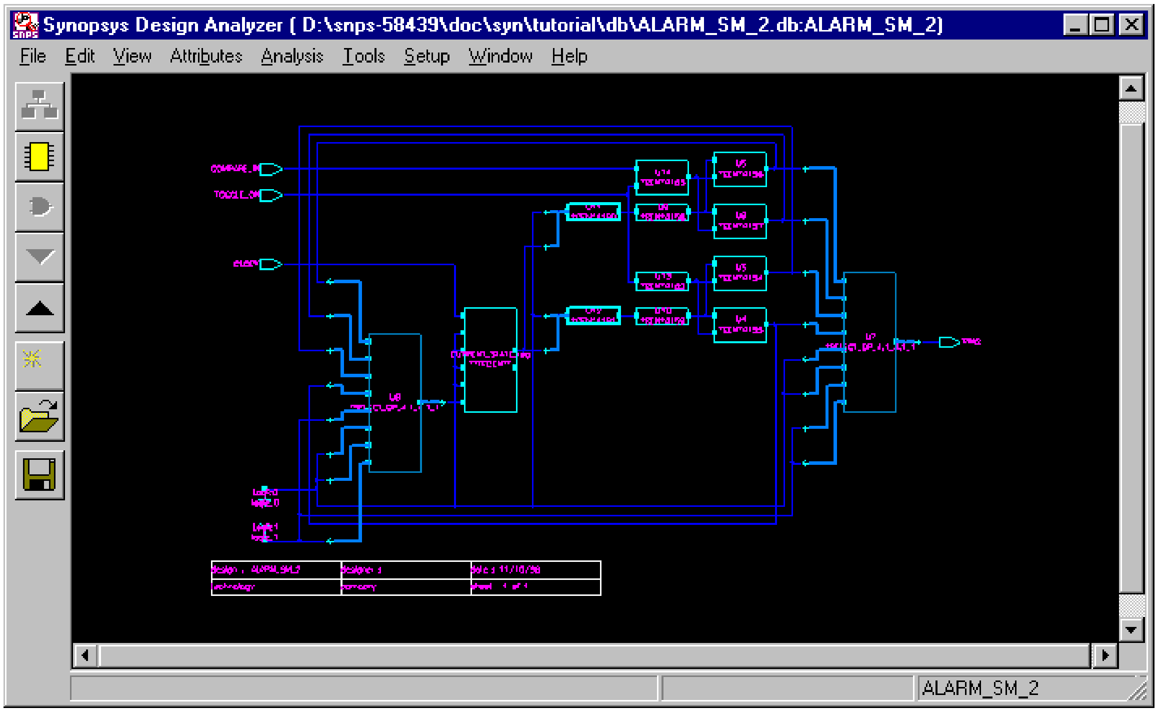


Current Design: TOP ·rent Instance: U5 (ALARM_SM_2) Schematic View

Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Figure 10-23 shows the schematic view for ALARM_SM_2 for the Windows NT OS.

Figure 10-23   Schematic View for ALARM_SM_2


Synopsys Design Analyzer (D:\snps-58439\doc\syn\tutorial\db\ALARM_SM_2.db:ALARM_SM_2)
File
Edit
View
Attributes
Analysis
Tools
Setup
Window
Help
ALARM_SM_2

5. When you finish, close the new window.

Printing Schematics Using Design Analyzer

A schematic can print on different sheet sizes. The default sheet size is infinite (the schematic is generated on one sheet). The common sheet sizes provided in the Synopsys generic symbol library are

- A through E
- A0 through A4
- mentor_maximum

For more information about sheet sizes, see the create_schematic command in the man pages and the *Design Compiler User Guide*.

Resize and print a schematic by completing the following tasks:

- Reset the default schematic size.
- Regenerate the schematic.
- Print the schematic.

To reset the default schematic size,

1. Select CONVERTOR_0 in the Designs view.
2. Generate the Schematic view.
3. Choose Setup > Defaults to display the Defaults window.

   The current value for Schematic Options is -size infinite.
4. Change infinite to **A** in the Schematic Options field.
5. Click OK.

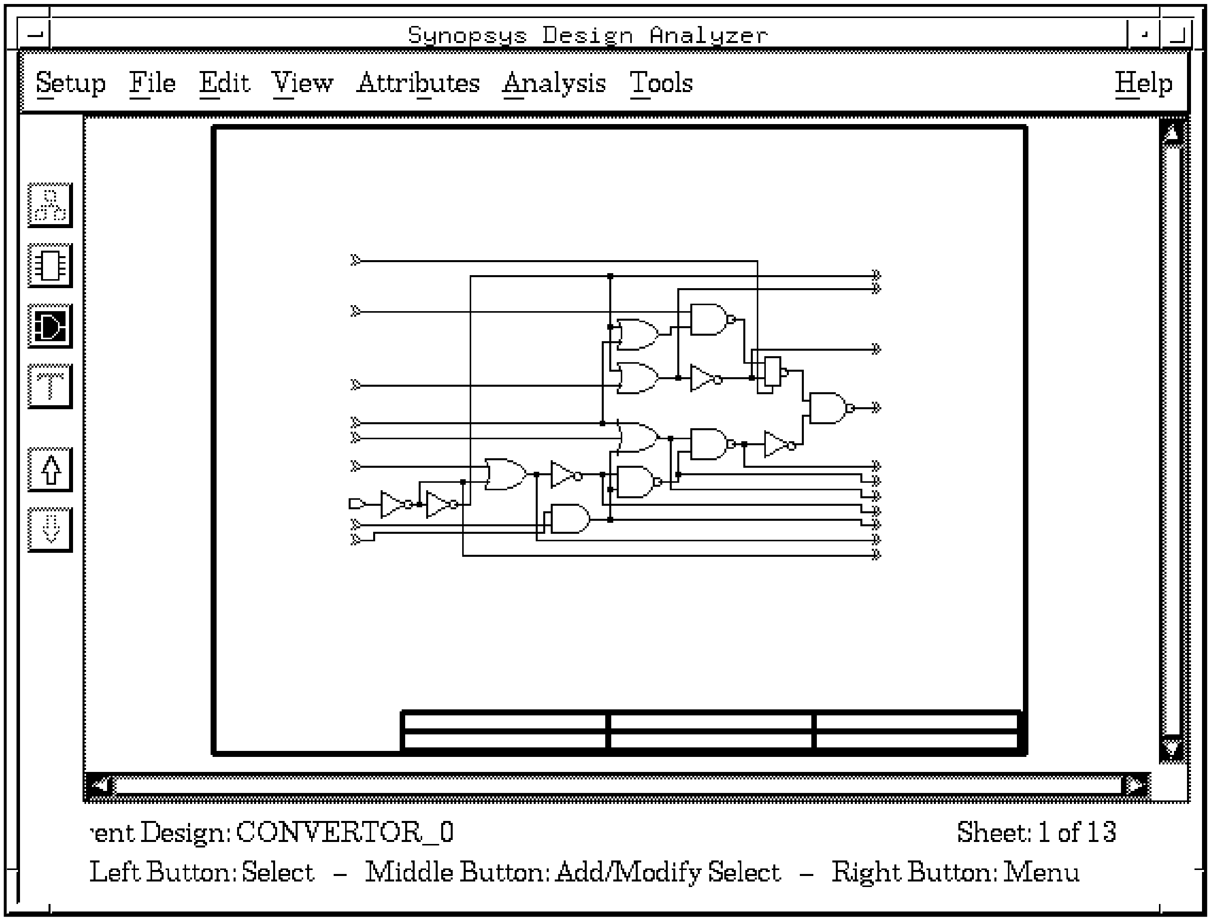
   The default schematic size is set to size A.

To recreate the schematic,

- To view the Schematic view for size A, choose View > Recreate.

  Design Analyzer regenerates the view, as shown in Figure 10-24.

Figure 10-24 Schematic Using Schematic Size-A



ent Design: CONVERTOR_0 Sheet: 1 of 13

Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

This schematic view formats the schematic of CONVERTOR_0 to fit onto 13 size-A pages.

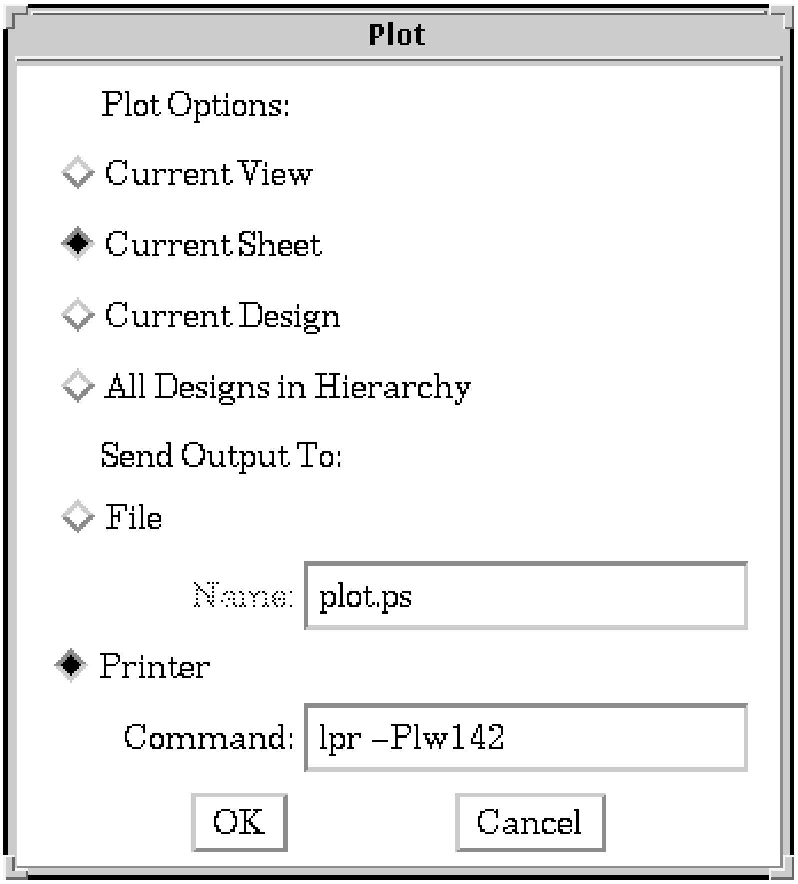
You can view other pages by choosing View > Change Sheet.

To print the schematic,

1. Choose File > Plot.

   The Plot window opens, as shown in Figure 10-25.

Figure 10-25 Plot Window



Plot Options:

◇ Current View

◆ Current Sheet

◇ Current Design

◇ All Designs in Hierarchy

Send Output To:

◇ File

Name: plot.ps

◆ Printer

Command: lpr –Plw142

2. Select Current Sheet.

   Current Sheet prints only the sheet shown in the Schematic view.

3. In the Command field, enter the print command for your local printer.

4. Click OK.

   The current schematic sheet prints.

To restore the size,

1. Choose Setup > Defaults.
2. Change A to infinite in the Schematic Options field.
3. Click OK.
4. Choose View > Recreate to re-create the size infinite Schematic View.

   The size is restored.

Analyzing Design Results Using dc_shell

This section contains a description of the tasks and exercises in the following sections:

- Generating Attribute Reports Using dc_shell
- Generating Analysis Reports Using dc_shell

Note:

In the following sections, all commands are presented in dcsh mode. The commands are identical in Tcl mode except in three instances. The Tcl form of the command is shown for these exceptions.

Generating Attribute Reports Using dc_shell

Attribute reports provide attribute information for various types of design objects. You can generate the attribute reports (using dc_shell) that are described in the following sections:

- Bus Report

Identifies the width of buses. Use for buses and nets.

- Cell Report

  Displays information about cells and subdesigns in the current design. Includes cell (instance) and reference names, their libraries, and the area for each cell.

- Net Report

  Provides information about fanout, fanin, loading, attributes, and the number of pins connected to each net.

- Compile Options Report

  Summarizes the compile options applied to the submodules.

Bus Report

The bus report lists bused ports and nets in the current design. Information in the report is linked to the schematic.

To generate the bus report,

- Enter the following command at the prompt:

  ```
  dc_shell> report_bus
  ```

Example Output

Buses are usually indexed in ascending order. If the bus is in descending order, the numbers in the Step column are negative.

```
Loading db file ’/usr/synopsys/libraries/syn/class.db’
Loading db file ’/usr/synopsys/libraries/syn/gtech.db’
Loading db file ’/usr/synopsys/libraries/syn/standard.sldb’
```

```
**************************************
Report : bus
Design : TOP
Version: 1998.08
Date   : Mon Oct 19 19:21:28 1998
*************************************

Bussed Port     Dir    From   To   Step   Width   Type
------------------------------------------------------
DISP1          out     13    0    -1       14    array
DISP2          out     13    0    -1       14    array


Bussed Net      From   To   Step    Width   Type
-------------------------------------------------------
DISP1           13      0    -1        14    array
DISP2           13      0    -1        14    array
KONNECT10        0     59     1         6    range
KONNECT6         1     12     1         4    range
KONNECT7         0     59     1         6    range
KONNECT9         1     12     1         4    range
KONNECT13      10     0   -1        11   array
U3/connect13     9      0    -1         10   array
U3/disp1         13     1    -1         13   array
U6/OUTBUS        10     0    -1         11   array
1
```

Cell Report

The cell report lists cells and subdesigns in the current design with

- Reference names
- Source library, if applicable
- Area
- Attributes

To generate the cell report,

- Enter the following command at the prompt:

```
dc_shell> report_cell
```

The TOP design contains only subdesigns, which are listed in the report with cell (instance) and reference names.

Library

Lists the library that contains the reference. This column is empty in this example because these designs are not library cells.

Area

Lists the area for each cell.

Attributes

Lists the design attributes. The Attributes key defines the letters listed in the Attributes column. Subdesigns have an h because they are always hierarchical. Subdesigns that contain sequential elements have an n, for noncombinational.

Example Output

```
Information: Updating design information... (UID-85)

*****************************************
Report : cell
Design : TOP
Version: 1998.08
Date   : Mon Oct 19 19:57:40 1998
******************************************


Attributes:
 b - black box (unknown)
 h - hierarchical
 n - noncombinational
 r - removable
 u - contains unmapped logic

Cell           Reference      Library  Area   Attributes
--------------------------------------------------------
U1             TIME_BLOCK              331.00 h, n
U2             ALARM_BLOCK             204.00 h, n
U3             CONVERTOR_CKT           325.00 h
U4             COMPARATOR               44.00 h
U5             ALARM_SM_2               11.00 h, n
U6             MUX                      55.00 h
-----------------------------------------------------
Total 6 cells                           970.00
1
```

Net Report

The net report lists nets in the current design with

- Fanout
- Fanin
- Load values
- Number of pins
- Attributes for the nets, if any

To generate the net report,

- Enter

  ```
  dc_shell> report_net
  ```

Example Output

After place and route tools create the circuit layout, load values of interconnect nets can be derived. You can input load values to Design Compiler to model the synthesized circuit more accurately. This process is called back-annotation. If a net is assigned an annotated capacitance or resistance value, a c or an r appears in the Attributes column. (This is not the case for the example design.)

```
****************************************************
Report : net
Design : TOP
Version: 1998.08
Date   : Mon Oct 19 20:06:20 1998
*********************************

Operating Conditions: WCCOM   Library: class
Wire Loading Model Mode: top

Design    Wire Loading Model     Library
--------------------------------------------------
TOP          10X10               class

Net               Fanout   Fanin    Load   Resistance  Pins Attributes
----------------------------------------------------------------------
ALARM                 13       1    28.26       0.00    14
AM_PM_DISPLAY          1       1     2.53       0.00     2
CLK                   33       1    43.48       0.00    34
DISP1[0]               2       1     4.84       0.00     3
DISP1[1]               1       1     3.53.      0.00     2
DISP1[2]               1       1     3.53       0.00     2
DISP1[3]               1       1     3.53.      0.00     2
DISP1[4]               1       1     3.53.      0.00     2
DISP1[5]               1       1     3.53.      0.00     2
DISP1[6]               1       1     3.53.      0.00     2
DISP1[7]               1       1     3.00       0.00     2
DISP1[8]               4       1    13.46       0.00     5
DISP1[9]               1       1    13.46       0.00     5
                               .
                               .
                               .

DISP2[7]               1       1     0.53       0.00     2
DISP2[8]               1       1     0.53       0.00     2
DISP2[9]               1       1     0.53       0.00     2
```

```
DISP2[10]              2       1     0.84       0.00     3
DISP2[11]              1       1     0.53       0.00     2
DISP[12]               1       1     0.53       0.00     2
DISP[13]               2       1     0.84       0.00     3
HRS                    4       1     5.46       0.00     5
KONNECT6[0]            5       1     7.77       0.00     6
KONNECT6[1]            6       1     10.09      0.00     7
KONNECT6[2]            5       1     8.77       0.00     6
KONNECT6[3]            6       1     9.09       0.00     7
KONNECT6[0]            6       1     10.09      0.00     7
                               .
                               .
                               .

KONNECT13[8]           4       1      6.46      0.00     5
KONNECT13[9]           4       1     15.02      0.00     10
KONNECT13[10]          8       1     11.71      0.00     9
MINS                   4       1      5.46      0.00     5
SET_TIME               2       1      2.84      0.00     3
SPEAKER_OUT            1       1      8.03      0.00     2
TOGGLE_SWITCH          1       1      2.53      0.00     2
------------------------------------------------------------------
Total 69 nets        260      69      440.94    0.00    329
Maximum               33       1       43.48    0.00     34
Average              3.77   1.00        6.39    0.00    4.77
1
```

Compile Options Report

The compile options report lists the options for compile for the current design.

To generate the compile options report, enter

- To generate the compile options report, enter

```
dc_shell> report_compile_options
```

Example Output

Default compile options were used previously for flatten and structure. The flatten option is off by default. The structure option is on and is timing-driven. These options are reflected in the report for the subdesigns.

```
************************************************
Report : compile_options
Design : TOP
Version: 1998.08
Date   : Tue Oct 20 07:04:18 1998
**************************************************

Design                    Compile Option          Value
--------------------------------------------------------
TOP                       flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

TIME_BLOCK                flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

TIME_STATE_MACHINE        flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

TIME_COUNTER              flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

TIME_COUNTER_DW01_inc_6_0 flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

MUX                       flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true
```

```
ALARM_BLOCK              flatten                   false
                         structure                 true
                         structure_boolean         false
                         structure_timing          true
                              .
                              .
                              .
CONVERTOR_0               flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

HOURS_FILTER              flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true

COMPARATOR                flatten                  false
                          structure                true
                          structure_boolean        false
                          structure_timing         true
-------------------------------------------------------------
1
```

Generating Analysis Reports Using dc_shell

The reports you generate in these exercises provide information you need in order to analyze the design. You can generate the analysis reports (using the dc_shell interface) that are described in the following sections:

- Hierarchy Report

  Lists the subdesigns of the hierarchy, including subdesigns cells and libraries.

- Timing Report

  Analyzes the critical path, path delays, and incremental delays through cells.

- Point Timing Report

  Analyzes the timing between the points you select.

The results in your reports and schematics might vary from those displayed.

Hierarchy Report

The hierarchy report lists all the cells in the current design or current instance.

To generate a hierarchy report,

- Enter

  ```
  dc_shell> report_hierarchy
  ```

Example Output

As is the case with example output for other reports with lengthy output, an abbreviated version of it is shown here, mainly to give you a sense of the report content. When you enter the command and it completes execution successfully, you will see the full report.

```
************************************************
Report : hierarchy
Design : TOP
Version: 1998.08
Date   : Tue Oct 20 07:28:38 1998
***********************************


TOP
   ALARM_BLOCK
      ALARM_COUNTER
        ALARM_COUNTER_DW01_inc_6_0
            ENI                            class
            EOI                            class
            IVI                            class
            ND2I                           class
```

```
            AN2I                        class
            ENI                         class
            FD1                         class
           FD1S                        class


           IVI                          class
           MUX21L                      class
            ND2I                        class
            NR2I                        class
        ALARM_STATE_MACHINE
            AN2I                        class
            FD1S                        class
            IVI                         class
            ND2I                        class
            NR2I                        class
      ALARM_SM_2
                         .
                         .
                         .
      TIME_STATE_MACHINE
            AN2I                        class
            FD1                         class
            FD1S                        class
            IVI                         class
            ND2I                        class
            NR2I                        class
1
```

Timing Report

A timing report provides timing information for defined endpoints and constrained pins. If no endpoints or constrained pins are explicitly specified, the report includes timing information only for the most critical path in the design.

To generate the timing report,

- Enter

```
dc_shell> report_timing -path full -delay max -max_paths 3 -nworst 1
```

The FPGA Options section applies only when the design’s target library is an FPGA library.

Example Output

The first cell in the first path is the register

`U2/U1/CURRENT_STATE_reg[0]/CP`

You might have identified another path as the most critical in your design.

```
************************************************
Report : timing
         -path full
         -delay max
         -max_paths 3
Design : TOP
Version: 1998.08
Date   : Tue Oct 20 07:40:15 1998
***********************************

Operating Conditions: WCCOM  Library: class
Wire Loading Model Mode: top

Design   Wire Loading Model       Library
-------------------------------------------
TOP              10X10             class

     Startpoint: U1/U1/CURRENT_STATE_reg[0]
                  (rising edge-trimmed flip-flop clocked by CLK)
     Endpoint: U1/U2/HOURS_OUT_reg[3]
                  (rising edge-trimmed flip-flop clocked by CLK)
     Path Group: CLK
     Path Type:  max


     Point                                  Incr   Path
     ---------------------------------------------------
     clock CLK (rise edge)                  0.00    0.00
     clock network delay (ideal)            0.00    0.00
     U1/U1/CURRENT_STATE_reg[0]/CP (FD1S)   0.00    0.00 r
     U1/U1/CURRENT_STATE_reg[0]/QN (FD1S)   3.73    3.73 r
                       .
                       .
                       .
     data arrival time                                 16.22
                       .
                       .
                       .
     clock CLK (rise edge)                     23.00  23.00
     clock network delay (ideal)                0.00  23.00
     U1/U2/HOURS_OUT reg[1]/CP (FD1)            0.00  23.00
     library setup time                        -0.80  22.20
     data required time                               22.20
-------------------------------------------------------------
     data required time                               22.20
     data arrival time                               -14.65
--------------------------------------------------------------
     slack (MET)                                       7.55
1
```

To trace the critical path,

- Enter

```
dc_shell> current_instance “U1/U1”
dc_shell> remove_highlighting -all -hier
dc_shell> report_timing -path full -delay max -max_paths 3 -nworst 1
```

In Tcl mode, the quotation marks are replaced by braces in the first command above:

```
dc_shell-t> current_instance {U1/U1}
```

Example Output

This section shows each discrete command, displays the output, and shows an abbreviated version of the report. When you run the command and it completes execution successfully, you see the following report:

```
dc_shell> current_instance “U1/U1”
Current instance is ’TOP/U1/U1’.
"TOP/U1/U1"
Generating schematic for design: TIME_STATE_MACHINE
The schematic for design ’TIME_STATE_MACHINE’ has 1 page(s).
1
```

```
dc_shell> remove_highlighting -all -hier
1

dc_shell> report_timing -path full -delay max -max_paths 3 -nworst 1

************************************************
Report : timing
         -path full
         -delay max
         -max_paths 3
Design : TOP
Version: 1998.08
Date   : Tue Oct 20 08:18:24 1998
***********************************

Operating Conditions: WCCOM  Library: class
Wire Loading Model Mode: top

Design     Wire Loading Model   Library
-----------------------------------------
TOP           10X10             class

   Startpoint: U1/U1/CURRENT_STATE_reg[0]
                (rising edge-trimmed flip-flop  clocked by CLK)
   Endpoint: U1/U2/HOURS_OUT_reg[3]
                (rising edge-triggered flip-flop clocked bk CLK)
   Path Group: CLK
   Path Type: max

Point                                      Incr    Path
----------------------------------------------------------
clock CLK (rise edge)                      0.00     0.00
clock network delay (ideal)                0.00     0.00
U1/U1/CURRENT_STATE_reg[0]/CP (FD1S)       0.00     0.00 r
U1/U1/CURRENT_STATE_reg[0]/QN (FD1S)       3.73     3.73 r
                   .
                   .
                   .
U1/U2/U361/Z  (ND2I)                        0.40    13.21 f
U1/U2/U341/Z  (MUX21H)                      2.31    15.52 f
U1/U2/HOURS_OUT_reg[2]/D (FD1)              0.00    15.52
data arrival time

clock CLK (rise edge)                       23.00   23.00
clock network delay (ideal)                  0.00   23.00
U1/U2/HOURS_OUT_reg[2]/CP (FD1)              0.00   23.00 r
library setup time                          -0.80   22.20
data required time                                  22.20
```

```
-------------------------------------------------------------
data required time                                  22.20
data arrival time                                  -15.52
--------------------------------------------------------------
slack (MET)                                          6.68

Startpoint: U1/U1/CURRENT_STATE_Reg[0]
              (rising edge-triggered flip-flop clocked by CLK)
Endpoint: U1/U2/HOURS_OUT_reg[1]
              (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point                                      Incr         Path
-------------------------------------------------------------
clock CLK (rise edge)                      0.00         0.00
clock network delay (ideal)                0.00         0.00
U1/U1/CURRENT_STATE_reg[0]/CP (FD1S)       0.00         0.00 r
U1/U1/CURRENT_STATE_reg[0]/QN (FD1S)       3.73         3.37 r
                              .
                              .
                              .
data required time                                      22.20
data arrival time                                      -14.65
--------------------------------------------------------------
slack (MET)                                              7.55
1
```

Point Timing Report

The point timing report provides information for defined startpoints, endpoints, and constrained pins or ports. If no endpoints are defined and there are no constrained pins or ports, the report includes all output ports. In this exercise, you generate a report on timing between selected points as ports, pins, or nets in the design.

By default, the timing endpoints reported by Design Compiler are the flip-flop D pins and output ports. The D pins have setup constraints derived because of the clock period constraint. The output ports have the max_delay constraint.

To generate the point timing report,

- Enter

  ```
  dc_shell> current_instance “../..”
  dc_shell> report_timing -path end -delay max -max_paths
  10 -nworst 1
  ```

  In Tcl mode, the quotation marks are replaced by braces in the first command above:

  ```
  dc_shell-t> current_instance {../..}
  ```

Example Output

An abbreviated version of the point timing report is shown here. You see the complete report when you run the report_timing command and it successfully completes execution.

```
dc_shell> current_instance “../..”
Current instance is the top-level of design
1

dc_shell> report_timing -path end -delay max -max_paths 10 -nworst 1
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 10
Design : TOP
Version: 1998.08
Date   : Tue Oct 20 10:03:49 1998
*****************************************

Operating Conditions: WCCOM  Library: class
Wire Loading Model Mode: top

Design    Wire Loading Model     Library
-----------------------------------------
TOP          10X10               class

Endpoint                        Path Delay      Path Required    Slack
-------------------------------------------------------------------------
U1/U2/HOURS_OUT_reg[3]/D (FD1)   16.22 f           22.20          5.98
```

```
U1/U2/HOURS_OUT_reg[2]/D (FDI)   15.52 f          22.20          6.68
U1/U2/HOURS_OUT_reg[1]/D (FDI)   14.65 f          22.20          7.55
U1/U2/AM_PM_reg/D        (FD1)   14.49 f          22.20          7.71
                           .
                           .
                           .
U1/U2/MINUTES_OUT_reg[3]/D (FD1)
                                 12.10 r          22.20         10.10
1
```

To display the pin layer, select two input pins, and generate a point timing report,

- Enter

```
dc_shell> set_layer pin_name_layer visible TRUE
dc_shell> set_layer pin_name_layer line_width 1
dc_shell> set_layer pin_name_layer plot_line_width 0
dc_shell> set_layer pin_name_layer red 65535
dc_shell> set_layer pin_name_layer green 65535
dc_shell> set_layer pin_name_layer blue 65535
dc_shell> report_timing -path full -delay max -max_paths 10 -nworst 1 -from
find( pin, { “U4/CLOCK_AM_PM” } ) -to find( pin, { “U5/COMPARE_IN” } )
```

In Tcl mode, the report_timing command is

```
dc_shell-t> report_timing -path full -delay max -max_paths 10 -nworst 1
-from [find pin [list {U4/CLOCK_AM_PM}]]  -to [find pin [list {U5/COMPARE_IN}]]
```

Example Output

You can use the point timing report to determine when a signal arrives at pins other than those reported by the default option. Here is an abbreviated version of the point timing report, the complete version of which is displayed when you issue the report_timing command.

```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 10
Design : TOP
Version: 1998.08
Date   : Tue Oct 20 10:27:28 1998
****************************************
```

```
Operating Conditions: WCCOM Library: class
Wire Loading Model Mode: top

Design    Wire Loading Model       Library
------------------------------------------
TOP         10x10                   class

  Startpoint: U1/U2/AM_PM_OUT_reg
             (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: U5/CURRENT_STATE_reg
             (rising edge-triggered  flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                                Incr   Path
  ------------------------------------------------
  clock CLK (rise edge)                0.00    0.00
  clock network delay (ideal)          0.00    0.00
  U1/U2/AM_PM_OUT_reg/CP (FD1)         0.00    0.00 r
  U1/U2/AM_PM_OUT_reg/Q (FD1)          3.70    3.70 r
                       .
                       .
                       .
data arrival time                              7.79

clock CLK (rise edge)                 23.00    23.00
clock network delay (ideal)            0.00    23.00
U5/CURRENT_STATE_reg/CP (FD15)         0.00    23.00 r
library setup time                    -1.30    21.70
data required time                             21.70
-----------------------------------------------------
data required time                             21.70
data arrival time                              -7.79
-----------------------------------------------------
slack (MET)                                    13.91
1
```

A

Tutorial Script Files

This appendix contains script files for the tutorial exercises. These script files contain Design Compiler commands and produce the same design results as the exercises in Chapters 7 to 10. The scripts are available in both dcsh mode and dctcl mode in your tutorial/ appendix_A directory.

You can run a script from the dc_shell command-line interface or in Design Analyzer by using either the Command Window or the Execute File window. You cannot run a dctcl mode script in Design Analyzer.

To run a script in dcsh mode, enter the command

```
    include script_filename
```

In dctcl mode, enter the command

```
    source script_filename
```

Note:

> Before you can run a dcsh-mode script or a dctcl-mode script, you must ensure that you are using the correct setup file. Your tutorial directory contains setup files for both modes. The dcsh-mode file name is .synopsys_dcsh.setup, and the dctcl-mode file name is .synopsys_dctcl.setup. Copy the appropriate file to .synopsys_dc.setup, depending on which mode you intend to use when running the scripts.

The script file names with their corresponding chapter numbers are listed below (dcsh mode and dctcl mode script files are identified by their .script and .tcl file extensions).

- setenv.script and setenv.tcl

  Use either script to set environment variables. The scripts correspond to the tasks of Chapter 7.

- optgoals.script and optgoals.tcl

  Use either script to define optimization goals and set constraints. The scripts correspond to the tasks of Chapter 8.

- cmpldes1.script and cmpldes1.tcl

  Use either script to compile the design. The scripts correspond to the tasks of Chapter 9.

- cmpldes2.script and cmpldes2.tcl

  Use either script to recompile the design with different parameter values. The scripts correspond to the tasks of Chapter 9.

- analyzres.script and analyzres.tcl

Use either script to generate reports on the results of design compilation. The scripts correspond to the tasks of Chapter 10.

In the scripts, the file names following the read commands are in .db format. If you are reading in the design in either Verilog or VHDL format, substitute the appropriate format and file names in the scripts.

Note:

If you are using VHDL format, read in the synopsys.vhd package, located in the vhdl directory, before you read in the first design file.

Set Environment Script

Use this script to set environment variables. The tasks associated with this script are explained in Chapter 7.

This script is presented in both dcsh mode and dctcl mode.

dcsh-mode script file setenv.script

```
/* design_analyzer script file for Chapter 7 */

/* Read in the design (all designs in the hierarchy) */

/* Analyze and elaborate 5 designs in the lowest hierarchy level */

        read -format vhdl {“./vhdl/synopsys.vhd”}
        analyze -format vhdl -lib WORK {./vhdl/ALARM_COUNTER.vhd, \
        ./vhdl/ALARM_STATE_MACHINE.vhd, ./vhdl/HOURS_FILTER.vhd, \
        ./vhdl/TIME_COUNTER.vhd, ./vhdl/TIME_STATE_MACHINE.vhd}
        elaborate ALARM_COUNTER -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate ALARM_STATE_MACHINE -arch “BEHAVIOR” -lib WORK \
        -update
        create_schematic -size infinite -gen_database
        elaborate HOURS_FILTER -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate TIME_COUNTER -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate TIME_STATE_MACHINE -arch “BEHAVIOR” -lib WORK \
        -update
        create_schematic -size infinite -gen_database

/*      Read CONVERTOR.pla in the lowest hierarchy level */

        read -format pla {“./vhdl/CONVERTOR.pla”}
        create_schematic -size infinite -gen_database

/* Analyze and elaborate designs in the second hierarchy level */

        analyze -format vhdl -lib WORK {./vhdl/ALARM_BLOCK.vhd, \
        ./vhdl/ALARM_SM_2.vhd, ./vhdl/COMPARATOR.vhd, \
        ./vhdl/CONVERTOR_CKT.vhd, ./vhdl/MUX.vhd, ./vhdl/\
        TIME_BLOCK.vhd}
        elaborate ALARM_BLOCK -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate COMPARATOR -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate CONVERTOR_CKT -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate MUX -arch “behavior” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate TIME_BLOCK -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database
        elaborate ALARM_SM_2 -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database

/*      Analyze and elaborate the top-level design */

        analyze -format vhdl -lib WORK {“./vhdl/TOP.vhd”}
        elaborate TOP -arch “BEHAVIOR” -lib WORK -update
        create_schematic -size infinite -gen_database

/* current_design “~tutorial/vhdl/TOP.db:TOP” */

        create_schematic -size infinite -symbol_view
        create_schematic -size infinite -hier_view
        create_schematic -size infinite -schematic_view
        current_instance “U2”
        create_schematic -size infinite -symbol_view -reference
        create_schematic -size infinite -hier_view -reference
        create_schematic -size infinite -schematic_view \
        -reference current_instance “..”

/* Set design attributes*/

        set_drive -rise .08 “ALARM”
        set_drive -fall .08 “ALARM”
        set_drive -rise .08 “HRS”
        set_drive -fall .08 “HRS”
        set_drive -rise .08 “MINS”
        set_drive -fall .08 “MINS”
        set_drive -rise .08 “SET_TIME”
        set_drive -fall .08 “SET_TIME”
        set_drive -rise .08 “TOGGLE_SWITCH”
        set_drive -fall .08 “TOGGLE_SWITCH”
        set_drive -rise drive_of (class/B4I/Z) “CLK”
        set_drive -fall drive_of (class/B4I/Z) “CLK”
        set_drive -rise 0.06 “SET_TIME”
        set_drive -fall 0.06 “SET_TIME”
        set_load load_of (class/IVA/A) * 5 “SPEAKER_OUT”
        set_load 3.0 “DISP1[13]”
        set_load 3.0 “DISP1[12]”
        set_load 3.0 “DISP1[11]”
        set_load 3.0 “DISP1[10]”
        set_load 3.0 “DISP1[9]”
        set_load 3.0 “DISP1[8]”
        set_load 3.0 “DISP1[7]”
        set_load 3.0 “DISP1[6]”
        set_load 3.0 “DISP1[5]”
        set_load 3.0 “DISP1[4]”
        set_load 3.0 “DISP1[3]”
        set_load 3.0 “DISP1[2]”
        set_load 3.0 “DISP1[1]”
        set_load 3.0 “DISP1[0]”
        set_load 3.0 “DISP2[13]”
        set_load 3.0 “DISP2[12]”
        set_load 3.0 “DISP2[11]”
        set_load 3.0 “DISP2[10]”
        set_load 3.0 “DISP2[9]”
        set_load 3.0 “DISP2[8]”
        set_load 3.0 “DISP2[7]”
        set_load 3.0 “DISP2[6]”
        set_load 3.0 “DISP2[5]”
        set_load 3.0 “DISP2[4]”
        set_load 3.0 “DISP2[3]”
        set_load 3.0 “DISP2[2]”
        set_load 3.0 “DISP2[1]”
        set_load 3.0 “DISP2[0]”
        set_load 2.0 “AM_PM_DISPLAY”

/* Set wire load model and operating conditions */

        set_wire_load “10x10” -library “class”
        set_operating_conditions -library “class” “WCCOM”
```

```
/* Save the design */

        write -format db -hierarchy -output “./db/ \
        TOP_attributes.db” {“TOP.db:TOP“}

quit
```

dctcl-mode script file setenv.tcl

```
#

# Tcl script file for Chapter 7

# Read in the design (all designs in the hierarchy)

# Analyze and elaborate 5 designs in the lowest hierarchy level

        read_file -format vhdl [list {./vhdl/synopsys.vhd}]
        analyze -format vhdl -lib WORK [list ./vhdl/ALARM_COUNTER.vhd
         ./vhdl/ALARM_STATE_MACHINE.vhd ./vhdl/HOURS_FILTER.vhd
        ./vhdl/TIME_COUNTER.vhd ./vhdl/TIME_STATE_MACHINE.vhd]
        elaborate ALARM_COUNTER -arch {BEHAVIOR} -lib WORK -update
        elaborate ALARM_STATE_MACHINE -arch {BEHAVIOR} -lib WORK -update
        elaborate HOURS_FILTER -arch {BEHAVIOR} -lib WORK -update
        elaborate TIME_COUNTER -arch {BEHAVIOR} -lib WORK -update
        elaborate TIME_STATE_MACHINE -arch {BEHAVIOR} -lib WORK -update

# Read CONVERTOR.pla in the lowest hierarchy level

        read_file -format pla [list {./vhdl/CONVERTOR.pla}]

# Analyze and elaborate designs in the second hierarchy level

        analyze -format vhdl -lib WORK [list ./vhdl/ALARM_BLOCK.vhd
        ./vhdl/ALARM_SM_2.vhd ./vhdl/COMPARATOR.vhd
        ./vhdl/CONVERTOR_CKT.vhd ./vhdl/MUX.vhd ./vhdl/TIME_BLOCK.vhd]
        elaborate ALARM_BLOCK -arch {BEHAVIOR} -lib WORK -update
        elaborate COMPARATOR -arch {BEHAVIOR} -lib WORK -update
        elaborate CONVERTOR_CKT -arch {BEHAVIOR} -lib WORK -update
        elaborate MUX -arch {behavior} -lib WORK -update
        elaborate TIME_BLOCK -arch {BEHAVIOR} -lib WORK -update
        elaborate ALARM_SM_2 -arch {BEHAVIOR} -lib WORK -update

# Analyze and elaborate the top-level design

        analyze -format vhdl -lib WORK [list {./vhdl/TOP.vhd}]
        elaborate TOP -arch {BEHAVIOR} -lib WORK -update

# current_design "~tutorial/vhdl/TOP.db:TOP"
```

```
        current_instance {U2}


# Set design attributes

        set_drive -rise .08 {ALARM}
        set_drive -fall .08 {ALARM}
        set_drive -rise .08 {HRS}
        set_drive -fall .08 {HRS}
        set_drive -rise .08 {MINS}
        set_drive -fall .08 {MINS}
        set_drive -rise .08 {SET_TIME}
        set_drive -fall .08 {SET_TIME}
        set_drive -rise .08 {TOGGLE_SWITCH}
        set_drive -fall .08 {TOGGLE_SWITCH}
        set_drive -rise [drive_of class/B4I/Z] {CLK}
        set_drive -fall [drive_of class/B4I/Z] {CLK}
        set_drive -rise 0.06 {SET_TIME}
        set_drive -fall 0.06 {SET_TIME}
        set_load [expr [load_of class/IVA/A] * 5] {SPEAKER_OUT}
        set_load 3.0 {DISP1[13]}
        set_load 3.0 {DISP1[12]}
        set_load 3.0 {DISP1[11]}
        set_load 3.0 {DISP1[10]}
        set_load 3.0 {DISP1[9]}
        set_load 3.0 {DISP1[8]}
        set_load 3.0 {DISP1[7]}
        set_load 3.0 {DISP1[6]}
        set_load 3.0 {DISP1[5]}
        set_load 3.0 {DISP1[4]}
        set_load 3.0 {DISP1[3]}
        set_load 3.0 {DISP1[2]}
        set_load 3.0 {DISP1[1]}
        set_load 3.0 {DISP1[0]}
        set_load 3.0 {DISP2[13]}
        set_load 3.0 {DISP2[12]}
        set_load 3.0 {DISP2[11]}
        set_load 3.0 {DISP2[10]}
        set_load 3.0 {DISP2[9]}
        set_load 3.0 {DISP2[8]}
        set_load 3.0 {DISP2[7]}
        set_load 3.0 {DISP2[6]}
        set_load 3.0 {DISP2[5]}
        set_load 3.0 {DISP2[4]}
        set_load 3.0 {DISP2[3]}
        set_load 3.0 {DISP2[2]}
        set_load 3.0 {DISP2[1]}
        set_load 3.0 {DISP2[0]}
        set_load 2.0 {AM_PM_DISPLAY}

# Set wire load model and operating conditions

        set_wire_load {10x10} -library {class}
        set_operating_conditions -library {class} {WCCOM}

# Save the design

        write -format db -hierarchy -output {./db/TOP_attributes.db}
        [list {TOP.db:TOP}]

        quit
```

Optimization and Constraints Script

Use this script to define optimization goals and set constraints. The tasks associated with this script are explained in Chapter 8.

This script is presented in both dcsh mode and dctcl mode.

dcsh-mode script file optgoals.script

```
/* design_analyzer script file for Chapter 8 */

/* Remove any designs in memory */

        remove_design -all

/* Read in the design with attributes set from Chapter 7 */

        read -format db {“./db/TOP_attributes.db”}
        create_schematic -size infinite -gen_database

/* Create a clock object, then set a clock constraint. */

        create_clock -name CLK -period 25 -waveform { 0 12.5 } { CLK }

/* Set a delay constraint on the output ports. */

        set_output_delay -clock CLK -max -rise 5 “SPEAKER_OUT”
        set_output_delay -clock CLK -max -fall 5 “SPEAKER_OUT”
        set_output_delay -clock CLK -max -rise 5 “AM_PM_DISPLAY”
        set_output_delay -clock CLK -max -fall 5 “AM_PM_DISPLAY”
        set_output_delay -clock CLK -max -rise 5 “DISP2[13]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[13]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[12]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[12]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[11]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[11]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[10]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[10]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[9]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[9]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[8]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[8]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[7]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[7]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[6]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[6]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[5]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[5]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[4]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[4]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[3]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[3]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[2]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[2]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[1]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[1]”
        set_output_delay -clock CLK -max -rise 5 “DISP2[0]”
        set_output_delay -clock CLK -max -fall 5 “DISP2[0]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[13]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[13]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[12]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[12]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[11]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[11]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[10]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[10]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[9]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[9]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[8]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[8]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[7]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[7]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[6]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[6]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[5]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[5]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[4]”
```

```
        set_output_delay -clock CLK -max -fall 5 “DISP1[4]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[3]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[3]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[2]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[2]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[1]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[1]”
        set_output_delay -clock CLK -max -rise 5 “DISP1[0]”
        set_output_delay -clock CLK -max -fall 5 “DISP1[0]”

/* Run check design */

        check_design
        check_timing
        current_design “TOP_attributes.db:CONVERTOR_CKT”
        create_schematic -size infinite -symbol_view
        create_schematic -size infinite -hier_view
        create_schematic -size infinite -schematic_view

/* Show the pin name layer */

        set_layer pin_name_layer visible TRUE
        set_layer pin_name_layer line_width 1
        set_layer pin_name_layer plot_line_width 0
        set_layer pin_name_layer red 65535
        set_layer pin_name_layer green 65535
        set_layer pin_name_layer blue 65535
        current_design “TOP_attributes.db:TOP”

/* Use uniquify to resolve multiple design instances */

        uniquify
        create_schematic -size infinite -gen_database

/* Save the design */

        write -format db -hierarchy -output “./db/\
        TOP_before_compile.db” \
        {“TOP_attributes.db:TOP”}

/* Delete designs for the optional exercises */

        remove_design find(design,”*”)

/* Read in TOP_attributes (from Chapter 7) */

        read -format db {“./db/TOP_attributes.db”}
        create_schematic -size infinite -gen_database
```

```
/* Run check design to recreate the warning message about multiple */
/* design instances */

        check_design
        current_design “TOP_attributes.db:CONVERTOR”
        create_schematic -size infinite -symbol_view
        create_schematic -size infinite -hier_view

/* Map design CONVERTOR to gates */

        compile -map_effort medium
        current_design = “TOP_attributes.db:CONVERTOR”
        create_schematic -size infinite -gen_database
        create_schematic -size infinite -schematic_view \
        -symbol_view -hier_view

/* Place the dont_touch attribute on design CONVERTOR */

        set_dont_touch “TOP_attributes.db:CONVERTOR”
        current_design “TOP_attributes.db:TOP”

/* Rerun check design */

        check_design

/* Delete designs to run optional exercise on ungroup */

        remove_design find(design,”*”)

/* Read in TOP_attributes (from Chapter 7) */

        read -format db {“./db/TOP_attributes.db”}
        create_schematic -size infinite -gen_database

/* Rerun check design to recreate the warning message on multiple */
/* design instances */

        check_design
        current_design “TOP_attributes.db:CONVERTOR”

/* Set the ungroup attribute on design CONVERTOR */

        set_ungroup “TOP_attributes.db:CONVERTOR”

/* Rerun check design */

        current_design = “TOP_attributes.db:TOP”
        check_design
        quit
```

dctcl-mode script file optgoals.tcl

```
#

# Tcl script file for Chapter 8

# Remove any designs in memory

        remove_design -all

# Read in the design with attributes set from Chapter 7

        read_file -format db [list {./db/TOP_attributes.db}]

# Create a clock object, then set a clock constraint

        create_clock -name CLK -period 25 -waveform [list 0 12.5] [list CLK]

# Set a delay constraint on the output ports

        set_output_delay -clock CLK -max -rise 5 {SPEAKER_OUT}
        set_output_delay -clock CLK -max -fall 5 {SPEAKER_OUT}
        set_output_delay -clock CLK -max -rise 5 {AM_PM_DISPLAY}
        set_output_delay -clock CLK -max -fall 5 {AM_PM_DISPLAY}
        set_output_delay -clock CLK -max -rise 5 {DISP2[13]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[13]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[12]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[12]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[11]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[11]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[10]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[10]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[9]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[9]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[8]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[8]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[7]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[7]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[6]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[6]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[5]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[5]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[4]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[4]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[3]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[3]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[2]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[2]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[1]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[1]}
        set_output_delay -clock CLK -max -rise 5 {DISP2[0]}
        set_output_delay -clock CLK -max -fall 5 {DISP2[0]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[13]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[13]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[12]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[12]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[11]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[11]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[10]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[10]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[9]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[9]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[8]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[8]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[7]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[7]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[6]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[6]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[5]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[5]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[4]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[4]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[3]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[3]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[2]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[2]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[1]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[1]}
        set_output_delay -clock CLK -max -rise 5 {DISP1[0]}
        set_output_delay -clock CLK -max -fall 5 {DISP1[0]}

# Run check design

        check_design
        check_timing

# Show the pin name layer

        set_layer pin_name_layer visible TRUE
        set_layer pin_name_layer line_width 1
        set_layer pin_name_layer plot_line_width 0
        set_layer pin_name_layer red 65535
        set_layer pin_name_layer green 65535
        set_layer pin_name_layer blue 65535
        current_design {TOP_attributes.db:TOP}

# Use uniquify to resolve multiple design instances

        uniquify

# Save the design

        write -format db -hierarchy -output {./db/TOP_before_compile.db}
        [list {TOP_attributes.db:TOP}]

# Delete designs for the optional exercises

        remove_design [find design {*}]

# Read in TOP_attributes (from Chapter 7)

        read_file -format db [list {./db/TOP_attributes.db}]

# Run check design to recreate the warning message about multiple
# design instances

        check_design
        current_design {TOP_attributes.db:CONVERTOR}

# Map design CONVERTOR to gates

        compile -map_effort medium
        set current_design {TOP_attributes.db:CONVERTOR}

# Place the dont_touch attribute on design CONVERTOR

        set_dont_touch {TOP_attributes.db:CONVERTOR}
        current_design {TOP_attributes.db:TOP}

# Rerun check design

        check_design

# Delete designs to run optional exercise on ungroup

        remove_design [find design {*}]

# Read in TOP_attributes (from Chapter 7)

        read_file -format db [list {./db/TOP_attributes.db}]

# Rerun check design to recreate the warning message on multiple
# design instances

        check_design
        current_design {TOP_attributes.db:CONVERTOR}

# Set the ungroup attribute on design CONVERTOR

        set_ungroup {TOP_attributes.db:CONVERTOR}

# Rerun check design

        set current_design {TOP_attributes.db:TOP}
        check_design

        quit
```

First Compilation Script

Use this script to compile the hierarchical design example. The tasks associated with this script are explained in Chapter 9.

This script is presented in both dcsh mode and dctcl mode.

dcsh-mode script file cmpldes1.script

```
/* design_analyzer script file for first part of Chapter 9 */

/* Run this compile script first. */

/* Remove any designs in memory */

        remove_design -all

/* Read in the design */

        read -format db {“./db/TOP_before_compile.db”}
        create_schematic -size infinite -gen_database

/* Optimize the design to meet the constraints */

        compile -map_effort medium -verify -verify_effort low \
        -boundary_optimization

        quit
```

dctcl-mode script file cmpldes1.tcl

```
#

# Tcl script file for first part of Chapter 9

# Run this compile script first.

# Remove any designs in memory

        remove_design -all

# Read in the design

        read_file -format db [list {./db/TOP_before_compile.db}]

# Optimize the design to meet the constraints

        compile -map_effort medium -verify -verify_effort low
        -boundary_optimization

# Write current design to compile.db

        write $current_design -format db -hier -output ./db/compile.db

        quit
```

Second Compilation Script

Use this script to recompile the hierarchical design example with a new clock constraint. The tasks associated with this script are explained in Chapter 9.

This script is presented in both dcsh mode and dctcl mode.

dcsh-mode script file cmpldes2.script

```
/* design_analyzer script file for second part of Chapter 9 */

/* Note: Run this script after the first compile completes. */

/* Remove any designs in memory */
```

```
        remove_design -all

/* Read in the optimized design */

        read -format db {./db/compile.db}
        create_schematic -size infinite -gen_database
        current_design = “compile.db:TOP”
        create_schematic -size infinite -symbol_view
        create_schematic -size infinite -hier_view

/* Generate the area and constraints reports */

        report_area
        report_constraints

/* Set new constraints */

        create_clock -name CLK -period 23 -waveform { 0 11.5 } { CLK }

/* Reoptimize the design and explore the design space */

        compile -map_effort medium -verify -verify_effort low \
        -boundary_optimization
        current_design = “compile.db:TOP”
        create_schematic -size infinite -schematic_view \
        -symbol_view -hier_view

/* Generate area and constraints reports on the optimized design */

        report_area
        report_constraints

/* Save the optimized design */

        write -format db -hierarchy -output “./db/TOP_compiled.db” \
        {“compile.db:TOP”}

/* Delete designs */

        remove_design find(design,”*”)

/****** Optional Exercise - Instance-Specific Hierarchy ******/

        read -format db {“./db/TOP_before_compile.db”}

/* set current design to TOP */

        current_design = TOP
```

```
/* set current instance to CONVERTOR_CKT TOP/U3 */

        current_instance U3

/* set current instance to CONVERTOR_1 TOP/U3/U7 */

        current_instance U7
        report_net
        set_load 2.5 A0
        report_net
        current_instance .
        current_instance ..
        current_instance

        quit
```

dctcl-mode script file cmpldes2.tcl

```
#

# Tcl script file for second part of Chapter 9

# Note: Run this script after the first compile completes.

# Remove any designs in memory

        remove_design -all

# Read in the optimized design

        read_file -format db [list ./db/compile.db]
        set current_design {compile.db:TOP}

# Generate the area and constraints reports

        report_area
        report_constraint

# Set new constraints

        create_clock -name CLK -period 23 -waveform [list 0 11.5] [list CLK]

# Reoptimize the design and explore the design space

        compile -map_effort medium -verify -verify_effort low
        -boundary_optimization
        set current_design {compile.db:TOP}

# Generate area and constraints reports on the optimized design
```

```
        report_area
        report_constraint

# Save the optimized design

        write -format db -hierarchy -output {./db/TOP_compiled.db}
        [list {compile.db:TOP}]

# Delete designs

        remove_design [find design {*}]

# ****** Optional Exercise - Instance-Specific Hierarchy ******

        read_file -format db [list {./db/TOP_before_compile.db}]

# set current design to TOP

        set current_design TOP

# set current instance to CONVERTOR_CKT TOP/U3

        current_instance U3

# set current instance to CONVERTOR_1 TOP/U3/U7

        current_instance U7
        report_net
        set_load 2.5 A0
        report_net
        current_instance .
        current_instance ..
        current_instance

        quit
```

Design Analysis Script

Use this script to generate various reports on the compilation results of the previous scripts. The tasks associated with this script are explained in Chapter 10.

This script is presented in both dcsh mode and dctcl mode.

dcsh-mode script file anlyzres.script

```
/* design_analyzer script file for Chapter 10 */

/* Remove any designs in memory */

        remove_design -all

/* Read in the compiled design */

        read -format db {“./db/TOP_compiled.db”}
        create_schematic -size infinite -gen_database

/* Generate reports */

        report_bus
        report_cell
        report_net
        report_compile_options
        report_hierarchy
        report_timing -path full -delay max -max_paths 3 -nworst 1
        current_instance “U1/U1”
        create_schematic -size infinite -symbol_view -reference
        create_schematic -size infinite -hier_view -reference
        create_schematic -size infinite -schematic_view -reference
        remove_highlighting -all -hier
        report_timing -path full -delay max -max_paths 3 -nworst 1
        current_instance “../..”
        report_timing -path end -delay max -max_paths 10 -nworst 1

/* Turn on the pin_name_layer */

        set_layer pin_name_layer visible TRUE
        set_layer pin_name_layer line_width 1
        set_layer pin_name_layer plot_line_width 0
        set_layer pin_name_layer red 65535
        set_layer pin_name_layer green 65535
        set_layer pin_name_layer blue 65535

/* Show timing to a specific pin */

        report_timing -path full -delay max -max_paths 10 -nworst 1 \
        -from find( pin, { “U4/CLOCK_AM_PM” } ) -to find( pin, \
        { “U5/COMPARE_IN” } )

/* Set current_design to CONVERTOR_0 */

        current_instance “U4”
        create_schematic -size infinite -symbol_view -reference
        create_schematic -size infinite -hier_view -reference
        create_schematic -size infinite -schematic_view -reference
        current_instance “../U5”
        create_schematic -size infinite -symbol_view -reference
        create_schematic -size infinite -hier_view -reference
        create_schematic -size infinite -schematic_view -reference
        current_instance “..”
        current_design “TOP_compiled.db:CONVERTOR_0”
        create_schematic -size infinite -symbol_view
        create_schematic -size infinite -hier_view
        create_schematic -size infinite -schematic_view

/* Set default schematic option to size A */

        designer = “Mike”
        company = “Synopsys”
        link_library = “class.db “
        target_library = “class.db “
        symbol_library = “class.sdb “
        default_schematic_options = “-size A”
        hdlin_source_to_gates = “off”

/* Recreate schematic on size A sheets */

        create_schematic -size a -schematic_view -symbol_view \
        -hier_view

/* Set default schematic option to size infinite */

        designer = “Mike”
        company = “Synopsys”
        link_library = “class.db “
        target_library = “class.db “
        symbol_library = “class.sdb “
        default_schematic_options = “-size infinite”
        hdlin_source_to_gates = “off”
        create_schematic -size infinite -schematic_view \
        -symbol_view -hier_view

/* Perform a check_design and check_timing on design TOP */

        current_design “TOP_compiled.db:TOP”
        check_design
        check_timing

        quit
```

dctcl-mode script file anlyzres.tcl

```
#

# Tcl script file for Chapter 10

# Remove any designs in memory

        remove_design -all

# Read in the compiled design

        read_file -format db [list {./db/TOP_compiled.db}]

# Generate reports

        report_bus
        report_cell
        report_net
        report_compile_options
        report_hierarchy
        report_timing -path full -delay max -max_paths 3 -nworst 1
        current_instance {U1/U1}
        report_timing -path full -delay max -max_paths 3 -nworst 1
        current_instance {../..}
        report_timing -path end -delay max -max_paths 10 -nworst 1

# Turn on the pin_name_layer

        set_layer pin_name_layer visible TRUE
        set_layer pin_name_layer line_width 1
        set_layer pin_name_layer plot_line_width 0
        set_layer pin_name_layer red 65535
        set_layer pin_name_layer green 65535
        set_layer pin_name_layer blue 65535

# Show timing to a specific pin

        report_timing -path full -delay max -max_paths 10 -nworst 1 \
        -from [find pin [list {U4/CLOCK_AM_PM}]] -to [find pin \
        [list {U5/COMPARE_IN}]]

# Set current_design to CONVERTOR_0

        current_instance {U4}
        current_instance {../U5}
        current_instance {..}
        current_design {TOP_compiled.db:CONVERTOR_0}
```

```
# Perform a check_design and check_timing on design TOP

        current_design {TOP_compiled.db:TOP}
        check_design
        check_timing

        quit
```

B

UNIX and the Windows NT OS for Synthesis Products

The Synopsys synthesis products are designed to operate similarly on UNIX systems and Windows NT systems. With care and third-party tools, the same designs and script files can be made to run in all environments; however, the underlying operating systems do impose some differences. This appendix summarizes those differences and discusses their impact on the operation of the Synopsys synthesis tools.

This appendix includes the following sections:

- Specifying Files
- Using Environment Variables
- Location of Files and Directories
- Using Operating System Supplied Commands

Because a command can modify your input before the command accesses the operating system, the differences between, for example, path specifications under the Windows NT operating system and UNIX might not apply when you are specifying arguments to that command. Specifically, dc_shell is designed to accept path names in the UNIX style whether it is running on UNIX or the Windows NT operating system. However, the dc_shell command sh passes arguments to an operating system supplied command. That operating system supplied command might have requirements that differ from the dc_shell requirement. Test your specific environment to determine the correct combinations.

Specifying Files

How you specify a file depends on whether you are using a UNIX system or a Windows NT system and whether you are operating within the dc_shell or using third-party utilities.

Comparison of UNIX and Windows NT OS Paths

Table B-1 compares UNIX and Windows NT operating system path specifications:

Table B-1 UNIX and Windows NT OS Path Specifications

Type of Path	UNIX	WIndows NT OS
Absolute	/user/designers/my_design.db	c:\users\%username%\designers\my_design.db
Relative to current directory	./my_design.db or my_design.db	.\my_design.db or my_design.db
Relative to parent directory	../designers/my_design.db	..\designers\my_design.db

Table B-1 UNIX and Windows NT OS Path Specifications

Type of Path	UNIX	WIndows NT OS
Relative to home directory	~designers/my_design.db	not supported

The UNIX paths are relative to the local file system root, and the Windows NT operating system paths are relative to the root of a drive or partition. Under both systems, files located on a file server can be mounted to appear as part of the local file system. Under UNIX, the remote files appear as a portion of the local file system, whereas under the Windows NT operating system, the remote files appear under a drive letter that is defined when the remote file system is imported.

Universal Naming Convention (UNC) Path Names

In addition to the native Windows NT operating system path name specification, the Windows NT operating system also supports Universal Naming Convention (UNC) path names. UNC path names start with two backslashes (\\) and consist of a server portion and path name portion. For example: \\chip_server\designers\my_design.db. In this example, the file my_design.db exists on the system chip_server in the folder designers. Note that no drive letter appears in the UNC name. When resolving a UNC name, the Windows NT operating system searches for the top-level folder among all the top-level folders on the specified system. If the name of the top level folder is not unique on that system, file access fails, and you receive an error message. For example, if on the system chip_server, both C:\designers and D:\designers exist, using the UNC file name \\chip_server\designers\my_design.db produces an error.

Backslash (\) Versus Forward Slash (/)

Both the UNIX and the Windows NT operating systems use a hierarchical file specification with directories (or folders), which can contain other directories (folders) and files. UNIX uses the forward slash (/) to separate the different levels of the hierarchical specification; by default, the Windows NT operating system uses the backslash (\). However, the user interface of a program can use a different file naming system than the operating system uses. In particular, dc_shell running on Windows NT systems accepts path names using either the forward slash or backslash, whether the format is basic Windows NT operating system or UNC. Various third-party tools that dc_shell might interact with can impose different requirements. For example, some NFS clients accept UNC-style names with the forward slash as the separator, whereas others require backslashes.

Because the individual programs impose the requirements, you must determine which forms are correct for your own environment.

Using Environment Variables

Both the UNIX and the Windows NT operating systems have environment variables, but the syntax for using these variables differs. Shell scripts or individual commands executed through dc_shell use the operating system's environment variable syntax. For example:

UNIX

```
set file = "${design}.vhd"
```

Windows NT

```
set file = "%{design}.vhd%"
```

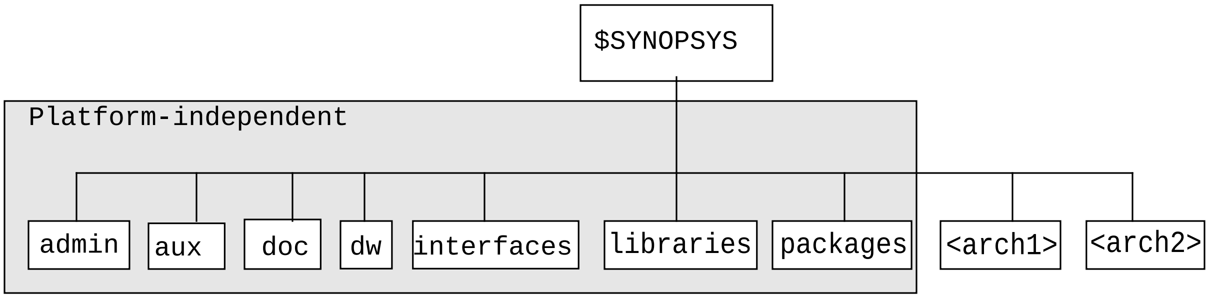
In addition to the command-line syntax illustrated here, with the Windows NT operating system, several environment variables are set by using an installation wizard when a product is installed, or they use the Control Panel.

Location of Files and Directories

For the products that are run from the terminal command line, such as dc_shell, the most common difference between operation under the Windows NT operating system and the UNIX operating system is the difference in path names. Typically, the actual file name is the same. Under the UNIX and the Windows NT operating systems, the Synopsys synthesis products are installed relative to a single root directory. This root directory is defined by the environment variable SYNOPSYS. In most cases, the path starting from SYNOPSYS is the same on both platforms, but the value of SYNOPSYS is something similar to S:\ (the root of a specific drive) under the Windows NT operating system, whereas the value is something similar to / (root) for a UNIX file system. The actual values are site-specific and determined when the tools are installed.

The basic directory structure is the same in the UNIX operating system versions and the Windows NT operating system versions, with one exception. Figure B-1 illustrates the UNIX version.

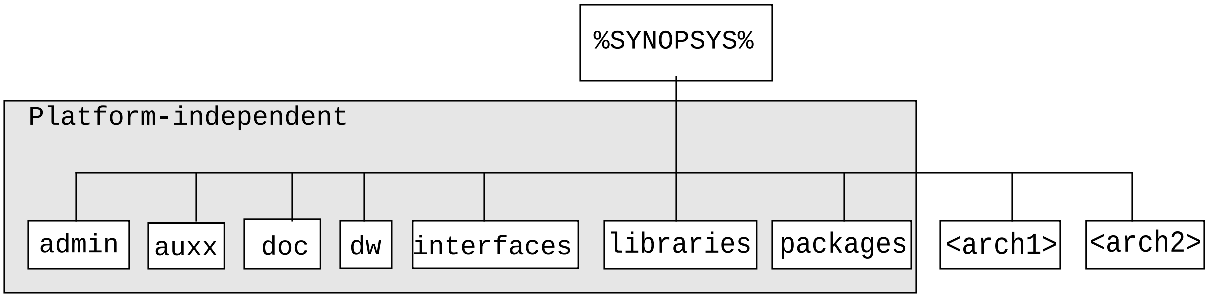
Figure B-1 UNIX Directory Structure



Note that the UNIX operating system versions contain a directory aux.

Figure B-2 illustrates the Windows NT OS version.

Figure B-2 Windows NT OS Directory Structure



Under the Windows NT operating system, the directory equivalent to aux is auxx.

Using Operating System Supplied Commands

You can use commands made available by the operating system whether you are using the tools under Windows NT operating system or the UNIX operating system. The mechanism is the same. However, because the commands are made available by the operating system, the availability, syntax, and semantics of the actual commands might differ.

Here are some examples:

- The UNIX command set is unavailable on most Windows NT systems
- The command lpr is available on both UNIX and Windows NT operating systems and performs the same function; however, the syntax is different.

  Under UNIX, the syntax is as follows:

  ```
  dc_shell> sh "lpr filename"
  ```

  or

  ```
  dc_shell-t> eval sh {lpr filename}
  ```

  The equivalent under Windows NT is as follows:

  ```
  dc_shell> sh "lpr -S server -P printer filename"
  ```

  or

  ```
  dc_shell-t> eval sh {lpr -S server -P printer filename}
  ```

- The command date is available on both UNIX and Windows NT operating systems, but the semantics differ. On UNIX systems the date command is most commonly used to report the current date and time, whereas on Windows NT systems, the command is used to change the date and time maintained by the system clock.

For some applications, you might want to supplement the commands supplied as part of the Windows NT system with the third-party product MKS Toolkit.

C

Creating a Home Directory in the Windows NT Operating System

To run the tutorial under the Windows NT operating system, you set up the tutorial directory structure in your home directory. In order to do so, you must define a home directory or a folder. This appendix explains how to create a home directory. To create a home directory, you must have administrator capability.

To create a home directory,

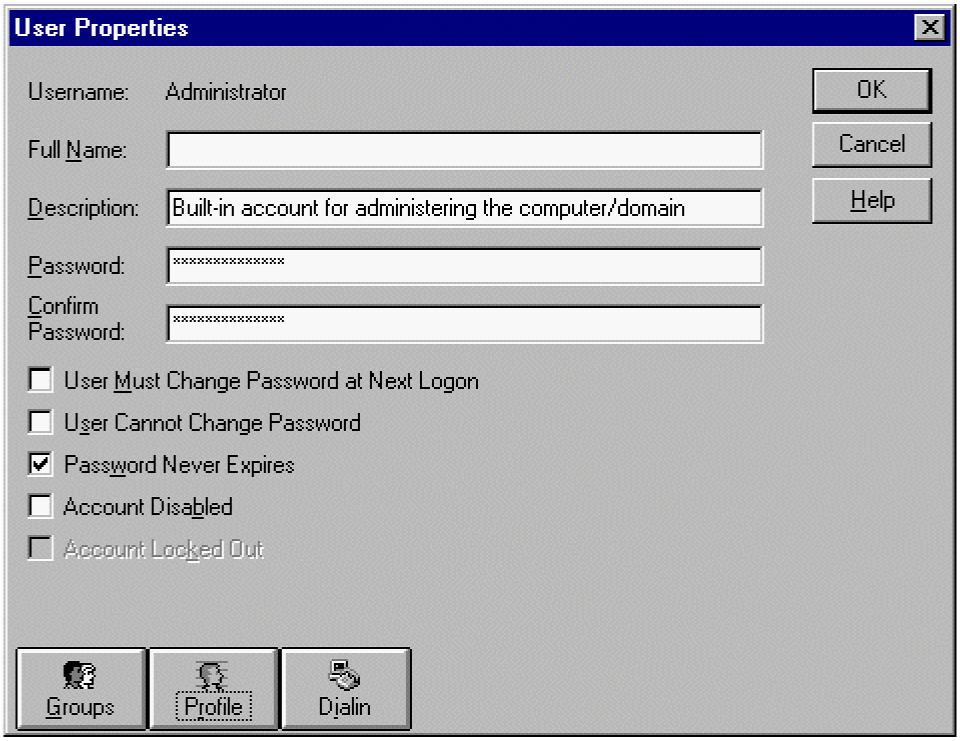
1. Log on to the Windows NT OS as administrator.
2. Double-click the “My Computer” icon.
3. Double-click the C: drive to display its contents.
4. Open the File menu: choose New, then choose Folder.
5. Name the new folder “users”.

6. Open the Start menu, choose Programs, then choose Administrative Tools, and finally choose User Manager.

7. Locate your name in the User Accounts List, and double-click it.

   If you do not find your name in the User Accounts List, open the User menu and choose New User. To create a new user for yourself, fill in the fields of the User Properties dialog box shown in Figure C-1, then click Profile at the bottom of the screen.

Figure C-1 User Properties Dialog Box



User Properties

Username: Administrator

Full Name:

Description: Built-in account for administering the computer/domain

Password: ***************

Confirm Password: ***************

□ User Must Change Password at Next Logon

□ User Cannot Change Password

☑ Password Never Expires

□ Account Disabled

□ Account Locked Out

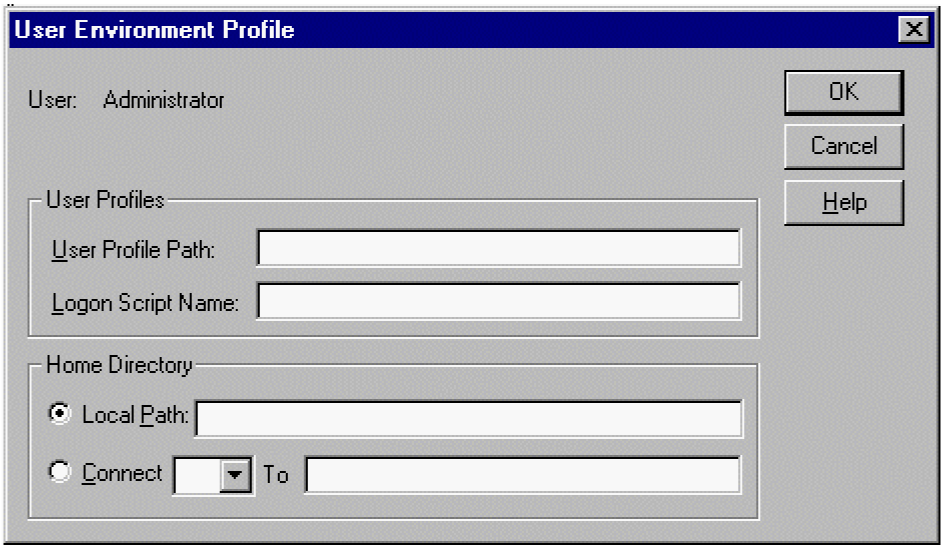
The User Environment Profile dialog box appears, shown in Figure C-2 on page C-3.

8. In the Home Directory section of the User Environment Profile dialog box, enter the Local Path:

```
C:\users\%username%
```

9. Click OK to save the information, and close the User Environment Profile dialog box.

Figure C-2 User Environment Profile Dialog Box



User Environment Profile

User: Administrator

User Profiles

User Profile Path:

Logon Script Name:

Home Directory

● Local Path:

○ Connect ▾ To

10. Click OK to close the User Properties dialog box, then click the close box to close the User Manager window.

11. Click the Users folder. This is your home directory.

D

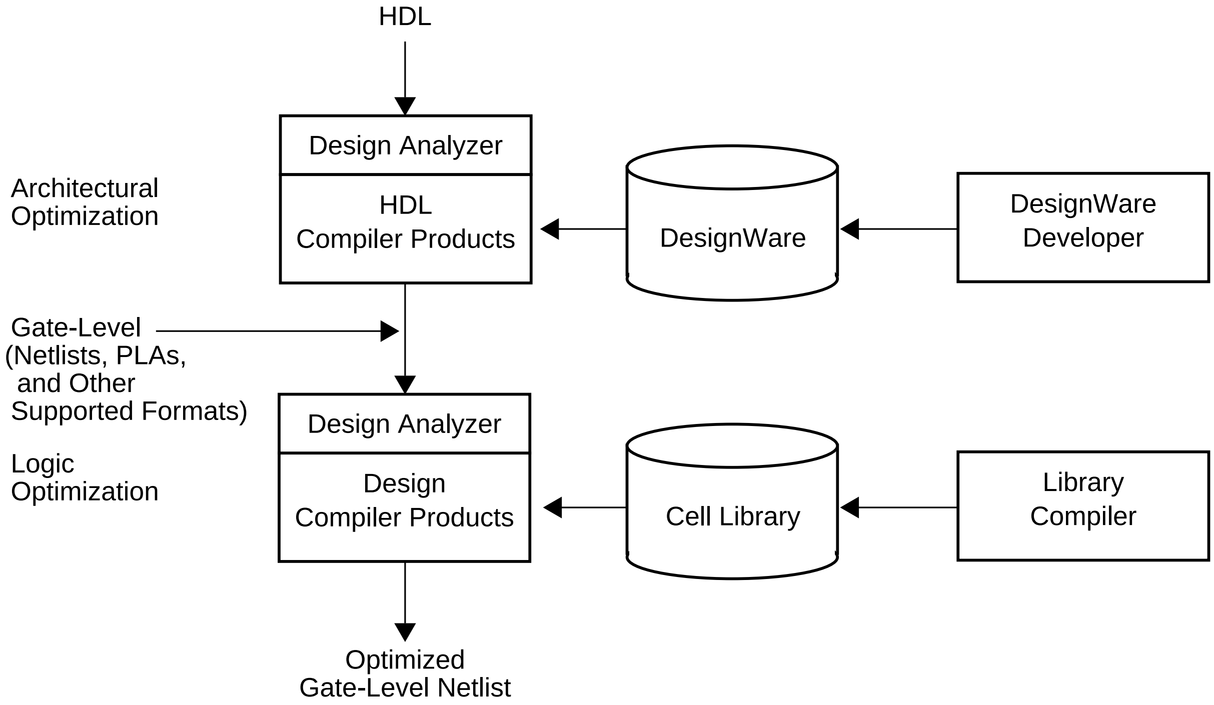
Synthesis Programs and Tools

Synthesis is the process of transforming a circuit defined at one level of abstraction into a lower-level definition. During this transformation, user-defined constraints define the goal of Design Compiler synthesis. Design Compiler software offers

- Architectural synthesis based on a hardware description language (HDL)
- Logic synthesis based on a gate-level description

Figure D-1 shows the synthesis of an HDL format, such as Verilog or VHDL, to a gate-level format.

Figure D-1   Logic Synthesis



Existing designs described in gate-level netlist formats can be reoptimized by using gate-level optimization.

Figure D-1 shows the following tools and libraries:

Design Analyzer

Consists of a graphical menu-based interface to the Synopsys tool set.

HDL Compiler products

Read and optimize Verilog and VHDL designs at the architectural level. A design is optimized before the gate-level netlist is passed to Design Compiler. Architectural-level optimization includes these processes:

- Arithmetic optimization

- Timing-and-area-based resource sharing
- Subexpression removal
- Constraint-driven resource selection and parameterization
- Inference of synthetic parts (DesignWare)

For more information about HDL Compiler products, see the *VHDL Compiler Reference Manual* and the *HDL Compiler for Verilog Reference Manual*.

DesignWare

Provides a library of operator-level components, such as adders and multipliers. HDL Compiler selects the correct type of component, based on the HDL description and your area and timing goals.

DesignWare Developer

Creates DesignWare libraries.

Design Compiler products

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 from your target cell library.

Cell Library

Library of cells, such as AND and OR cells, used by Design Compiler. For FPGA Compiler, this library can contain more complex cells, such as Xilinx configurable logic blocks (CLBs) and IOBs.

Library Compiler

Creates cell libraries.

Index