VHDL

Coding and Logic Synthesis with SYNOPSYS®

WENG FOOK LEE

VHDL
Coding and Logic Synthesis
with Synopsys®

This Page Intentionally Left Blank

VHDL

Coding and Logic Synthesis with Synopsys®

Weng Fook Lee
Advanced Micro Devices, Inc.

ACADEMIC PRESS
A Harcourt Science and Technology Company

San Diego San Francisco New York Boston London Sydney Tokyo

This book is printed on acid-free paper. ♾

ACADEMIC PRESS
A Harcourt Science and Technology Company
525 B Street, Suite 1900, San Diego, CA 92101-4495, USA
http://www.academicpress.com

Academic Press
Harcourt Place, 32 Jamestown Road, London, NW1 7BY, UK

Library of Congress Catalog Number: 00-103964

ISBN: 0-12-440651-3

Printed in the United States of America
00 01 02 03 04 05 MB 9 8 7 6 5 4 3 2 1

Dedicated to my late father,
the greatest man I have ever known.

This Page Intentionally Left Blank

TABLE OF CONTENTS

LIST OF FIGURES

This Page Intentionally Left Blank

LIST OF TABLES

This Page Intentionally Left Blank

LIST OF EXAMPLES

PREFACE

In today's world, faster and less costly ASIC chips are being designed at a much quicker rate than before. This requires that ASIC designers be able to design much more efficiently than before. Designers are constantly under pressure to come up with faster performing designs, but with fewer resources.

This has led to the development of many EDA tools that help designers to complete a design in a much shorter time frame. These EDA tools are based on the concept of designing ASIC components utilizing Hardware Description Language (HDL).

Today, a designer does not need to spend much time manually drawing the circuitry involved in a design but instead can write synthesizable HDL code. A common form of HDL code used in the ASIC industry for synthesis is ***V***ery High-Speed Integrated Circuit ***H***ardware ***D***escription ***L***anguage (VHDL) and Verilog. This book discusses only VHDL.

Synthesizable VHDL can be used as a form of input in synthesis tools such as Synopsys's Design Compiler. The synthesis tool can synthesize the logic circuit of the design with the functionality described by the VHDL code. This new methodology of design is a great asset to designers, as it increases both productivity and efficiency.

This book is divided into two parts. The first deals mainly with VHDL coding. Chapters 1–6 are included in the first part. In these chapters, the reader will see how simple and complex designs can be coded into synthesizable VHDL. Testbenches and timing diagrams are included to allow the reader to better understand the examples. The contents of this first part of the book will expose the reader to many examples of synthesizable code writing. In these examples, explanations and guidelines are included to give the reader an idea regarding the starting point required to write synthesizable VHDL code.

The examples in Chapter 3 are based on synthesizable code for simple and basic logic components that we see on a daily basis. Chapter 3 also discusses certain styling issues that are important for a designer to remember. Issues such as unwanted latch inference will be discussed in this chapter.

Chapter 4 discusses the usage of signal and variable in VHDL synthesis. Examples are included to bring the reader through the many stages involved in learning when to use signal and when to use variable.

Chapter 5 shows more examples of synthesizable VHDL code for complex logic components (shifter, counter, and memory module). Testbenches to exercise the examples are also included. Timing waveforms based on simulation results are drawn and discussed to enable the reader to obtain a better understanding of how each piece of synthesizable VHDL code translates into logic hardware.

Chapter 6 consists of a full-scale design project of a 3-stage pipeline microcontroller. This chapter begins with the definition of an instruction set for the microcontroller. This is followed by an architectural and microarchitectural definition of the microcontroller design.

Chapter 6 also shows the reader how the microcontroller is partitioned into functional blocks. The way these blocks can interface with each other to perform the instruction set execution is also discussed.

Synthesizable VHDL code for each functional block is then written and explained using simulation testbenches. Timing diagrams of simulation results are included to explain each functional block.

The second part of the book, which comprises Chapters 7–14, deals with logic synthesis using Synopsys's Design Compiler. Timing violations, microarchitectural tweaks, and synthesis options are all discussed to show the reader how a design that does not meet specification can be tweaked to obtain optimal results.

Chapter 7 discusses basic timing issues of which a designer should be aware prior to synthesizing a design. This chapter includes the topics of setup timing, hold timing, delay calculations, false paths, and multicycle paths. This chapter also discusses the general microarchitecural tweaks that can be done to obtain better timing performance.

Chapter 8 shows the reader the many different ways to optimize a design with different synthesis options using Synopsys's Design Compiler. Chapter 8 has many examples that shows the reader the different approaches to tweaking a design that does not meet timing requirements. Examples of designs that have timing violations (such as hold and setup violations) are fixed using the Design Compiler.

Chapter 9 discusses usage of GTECH components and how they can be used in VHDL code.

Chapter 10 discusses DesignWare components and how they can be inferred/ instantiated into VHDL code. This chapter also shows the reader how to create DesignWare components.

Chapter 11 discusses testability issues in synthesis and Chapter 12 gives an example of synthesizing for field programmable gate array (FPGA). Chapter 13 is a brief discussion of links from synthesis to layout while Chapter 14 provides several guidelines for a designer to follow when writing synthesizable code.

The many examples in this book, ranging from a simple description of basic logic components to a complex description of functional blocks within a pipeline micro-controller, show readers how each design can be transformed into synthesizable VHDL code. Each complex design is then synthesized and tweaked to obtain optimal synthesis results.

Upon completing this book, the reader will have a good understanding as to how designs can be coded into synthesizable VHDL, synthesized using Synopsys's Design Compiler, and tweaked to meet required specifications.

This book is targeted for engineers and students who want to learn how to write and synthesize VHDL code.

This Page Intentionally Left Blank

ACKNOWLEDGMENT

This book would not have been possible without a number of people whose help I truly appreciate. Among them are my AMD colleagues whose constant feedback has allowed me to improve the contents of this book. Mike Van Buskirk, Colin Bill, Vince Pitruzella, Brad Potts, Ben Oliver, Joe Kucera, Ed Bautista, Feng Pan, Azrul Halim, Santosh Yachareni, Chen Tien Min, Jeff Ferris, and Jerry Isaac have been great and there always to render support throughout the entire process of writing this book.

Special thanks to Sun Chong See, Yu Cheng Chang, Kar Keng Chua and Martin Wang for their time and effort in reviewing this book. A word of appreciation as well to Soo Me Goak and Mona Chee, who helped in correcting my many grammatical errors throughout this book. And of course, a special acknowledgement to Bryn Ekrood and the staff at Synopsys for their help in reviewing the synthesis portion of this book, the publishing staff at Academic Press and Dianne Littwin for their patience and support throughout the entire process of writing this book.

TRADEMARKS

PART I

VHDL CODING

1 INTRODUCTION

1.1 CONVENTIONAL DESIGN — SCHEMATIC CAPTURE

Since the 1980s, when schematic capture was introduced into the world of VLSI design, it has been a widely used design format. Today there are still many design houses in the industry using this concept of design.

In the concept of schematic capture, logic gates that will be used to design a certain circuit are hand-drawn using a computer-aided design (CAD) tool. Upon completion of schematics drawing, a database is stored based on the hand-drawn schematics. A common format used for the database is electronic database interchange format (EDIF).

Simulation tools are used to simulate the design based on the database. The designer determines if the design is funtionally correct based on the simulation results. If the functionality is wrong, the designer must then edit the hand-drawn schematics and re-run the simulation. This is performed in a loop until the designer is satisfied that the design has the right functionality.

Schematic capture was a good design methodology in the 1980s but it failed miserably in the 1990s when the number of logic gates involved in a design increased to hundreds of thousands. With IC chips becoming more complex by the day and the number of logic gates increasing tremendously with increased complexity of a design, schematic capture is becoming more an obstacle than a tool to help the designer.

Furthermore, in today's technological world, a new IC chip in the market will become obsolete very quickly. The window allowed for the IC chip to gain in sales and profits has become smaller. This smaller "market-window" is forcing designers to design a product for a much shorter time frame. Designers need to work faster and more efficiently in order to come out with better products in a shorter time frame.

This is where hardware description language enters the scene.

1.2 HARDWARE DESCRIPTION LANGUAGE

With rampant growth in technology, electronic design automation (EDA) software has been growing at an incredible rate during the past several years. This growth brings about faster and more efficient ways to design logic IC chips. And this is accomplished through the usage of HDL (hardware description language).

The most favorable type of HDL used today is very high-speed integrated circuit hardware description language (VHDL) and Verilog. This book discusses VHDL.

In VHDL design, a designer will code the design in terms of VHDL code as opposed to the conventional method of schematic capture. This code can then be synthesized using VHDL synthesis tools. The synthesized circuit can be stored in a netlist database.

Among the tools commonly used for VHDL synthesis are Synopsys's Design Compiler, Mentor Graphics' Autologix, Exemplar, Synplicity's Synplify, Cadence's Ambit and many others. This book discusses only the use of Synopsys's Design Compiler to synthesize and tweak the VHDL code.

1.3 VHDL DESIGN STRUCTURE

Before we proceed further with discussions about VHDL synthesis, it is imperative to have a basic understanding of VHDL design structure.

A VHDL design consists of three sections. They are the *entity, architecture,* and *configuration*:

- ***entity*** — the portion that declares the input/output/inout ports to the design;
- ***architecture*** — where the VHDL code is written to describe the internal architecture of the design; and
- ***configuration*** — this portion declares the *entity* and *architecture* for different submodules within the design.

In general, a VHDL file template looks a lot like this:

```
LIBRARY <library_name>;
USE <library_name>.<sub_library_ name>.ALL;
ENTITY <entity_name> IS
PORT (
<port_name> : <port_direction> <port_type>;
<port_name> : <port_direction> <port_type>
END <entity_name>;

ARCHITECTURE <architecture_name> OF <entity_name> IS
SIGNAL <signal_name> : <signal_type> := "<starting_
value>";
```

```
BEGIN
...............your VHDL code .......................
END <architecture_name>;

CONFIGURATION <configuration_name> OF <entity_name> IS
FOR <architecture_name>inputs
...... configuration of components .....
END FOR;
END <configuration_name>;
```

Whereby:

1. `<library_name>` is the name of library to use. An example is IEEE.
2. `<sub_library_name>` is the name of the `sub_library` used. An example would be `std_logic_1164`
3. `<entity_name>` is the name of the entity
4. `<port_name>` is the name of ports of a design
5. `<port_direction>` is `IN, OUT, INOUT` or `BUFFER`
6. `<port_type>` is the type declaration which can be `std_logic, std_logic_vector, bit, bit-vector` and others as defined in `library std_logic_1164` from IEEE
7. `<architecture_name>` is the name of the architecture
8. `<signal_name>` is name of a signal
9. `<signal_type>` is the signal type declaration which can be `std_logic, std_logic_vector, bit, bit_vector` and others as defined in library `std_logic_1164` from IEEE
10. `<starting_value>` is the initial value of the signal. This is optional.
11. `<configuration_name>` is the name of the configuration of the design.

For each VHDL design, there is only one *entity* declaration. There are designs that can have multiple *architecture* declarations and multiple *configuration* declarations.

Most designs consist only of an *entity*, an *architecture* and a *configuration* declaration. Figure 1 is a diagram of the single architecture. This coding style is encouraged as it is simple and easy to visualize. Furthermore, in many of today's VHDL simulation and synthesis tools, the *configuration* declaration is often not necessary if the design consists only of an *architecture* declaration without any component instantiations.

For designers who wish to have more control over the form of coding, there can be multiple *architecture* declarations for a given *entity*. See Fig. 2 for the diagram. But each *architecture* declaration must have its own corresponding *configuration* declaration. However multiple *architecture* declarations are seldom used in synthesizable VHDL code.

Example 1 shows the *entity* ***multiple_arch_ent*** having two *architectural* declarations, ***multiple_arch_ent_arch1*** and ***multiple_arch_ent_arch2***. For each of these *architectural* declarations, ***config_1*** and ***config_2*** are the corresponding *configuration* declarations.

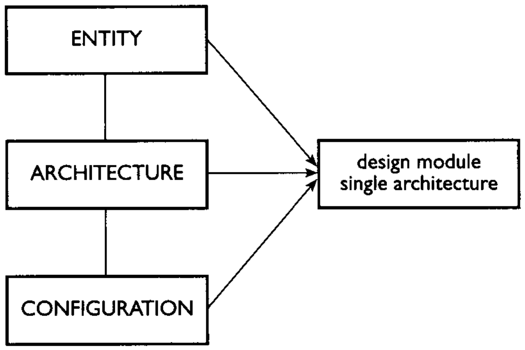


FIGURE 1 Diagram Showing a Single Architecture VHDL Design.

EXAMPLE 1 VHDL Code for Multiple Architecture

IEEE **library declaration**

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

Port declaration as input/output and `std_logic` **types**

```
ENTITY multiple_arch_ent IS
PORT (
        input : IN std_logic;
        output : OUT std_logic
      );
END multiple_arch_ent;
```

First architecture declaration

```
ARCHITECTURE multiple_arch_ent_arch1 OF multiple_arch_ent IS
BEGIN
   — your code for the architecture
END multiple_arch_ent_arch1 ;
```

Second architecture declaration

```
ARCHITECTURE multiple_arch_ent_arch2 OF multiple_arch_ent IS
BEGIN
   — your other code for the architecture
END multiple_arch_ent_arch2;
```

First configuration declaration

```
CONFIGURATION config_1 OF multiple_arch_ent IS
FOR multiple_arch_ent_arch1
END FOR;
END config_1;
```

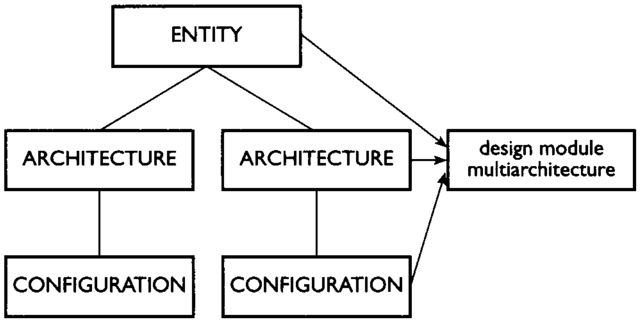


FIGURE 2 Diagram Showing Multiple Architecture VHDL Design.

```
CONFIGURATION config_2 OF multiple_arch_ent IS
FOR multiple_arch_ent_arch2
END FOR;
END config_2;
```

← Second configuration declaration

Note:
Hierarchical designs are based on the same VHDL structure, which is composed of *entity, architecture,* and *configuration*. Each submodule in the hierarchy (including the TOP level design) is represented by an entity, an architecture, and a configuration declaration. The submodules are instantiated into TOP-level VHDL code. If the submodules consist of even lower level sub-submodules, those sub-submodules are each represented by an entity, architecture, and a configuration. Each of them would then be instantiated into the submodule that glues them together.

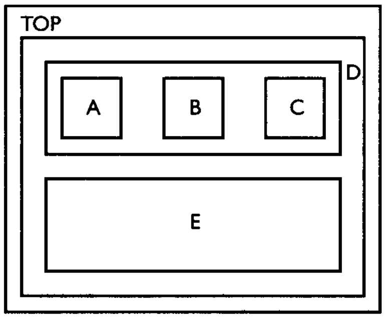


FIGURE 3 Diagram Showing a Hierarchical Design.

From Fig. 3, ***A, B***, and ***C*** are submodules to ***D*** while ***D*** and ***E*** are submodules to ***TOP***.

1.4 COMPONENT INSTANTIATION WITHIN A VHDL DESIGN STRUCTURE

For designs with component instantiations, *configuration* must be declared with association to each component in the architecture. For the design module of Fig. 4, Example 2 shows the corresponding VHDL code for the instantiated components.

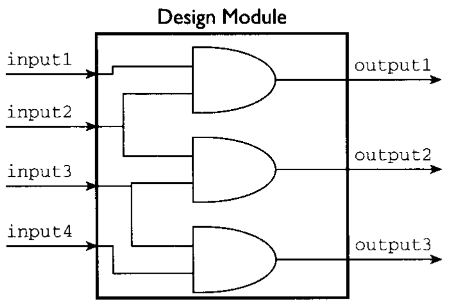


FIGURE 4 Diagram Showing a 3 AND Gate Component Instantiated Design.

EXAMPLE 2 VHDL Code for a 3 AND Gate Instantiated Component Design

***IEEE* library declaration**

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

Port declaration for input/output and type

```
ENTITY design_module_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        input3 : IN std_logic;
        input4 : IN std_logic;
        output1 : OUT std_logic;
        output2 : OUT std_logic;
        output3 : OUT std_logic
     );
END design_module_ent;


ARCHITECTURE design_module_arch OF design_module_ent IS
```

```
COMPONENT AND_gate_ent
PORT (
        AND_input1 : IN std_logic;
        AND_input2 : IN std_logic;
        AND_output : OUT std_logic
      );
END COMPONENT;
```

Declaration of usage of component **AND_gate_ent**

```
BEGIN
        logic_AND_1: AND_gate_ent PORT MAP (input1, input2, output1);
        logic_AND_2: AND_gate_ent PORT MAP (input2, input3, output2);
        logic_AND_3: AND_gate_ent PORT MAP (input3, input4, output3);
END design_module_arch;
```

Port mapping the instantiated components

```
CONFIGURATION design_module_config OF design_module_ent IS
FOR design_module_arch
        FOR logic_AND_1: AND_gate_ent
             USE ENTITY WORK.AND_gate_ent(AND_gate_arch);
        END FOR;
        FOR logic_AND_2: AND_gate_ent
             USE ENTITY WORK.AND_gate_ent(AND_gate_arch);
        END FOR;
        FOR logic_AND_3: AND_gate_ent
             USE ENTITY WORK.AND_gate_ent(AND_gate_arch);
        END FOR;
END FOR;
END design_module_config;
```

Configuration declaration for the instantiated components

Note: The code from Example 2 assumes the existence of a precompiled AND gate in the library ***WORK*** with the *entity* name ***AND_gate_ent*** and *architecture* name ***AND_gate_arch***.

Steps to precompile an AND gate with *entity* name ***AND_gate_ent*** and *architecture* name ***AND_gate_arch*** into library ***WORK***.

1. Create a directory ***WORK*** and link it to a VHDL library through whatever VHDL simulation/synthesis tools you are using.
2. Create another directory called ***SOURCE.***
3. Create a file <filename> in the ***SOURCE*** directory.
4. In the file <filename>, type the contents:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```
ENTITY AND_gate_ent IS
PORT (
      AND_input1 : IN std_logic;
      AND_input2 : IN std_logic;
      AND_output1 : OUT std_logic
);
END AND_gate_ent;

ARCHITECTURE AND_gate_arch OF AND_gate_ent IS
BEGIN
      output1 <= input1 AND input2;
END AND_gate_arch;
```

5. Compile the file <filename> into the directory ***WORK*** by using the VHDL library you defined in Step 1.
6. When you have completed Step 1 until Step 5, you will have a VHDL library defined and link to your directory ***WORK*** which contain the precompiled components of ***AND_gate_ent***.
7. Readers should also note that the steps needed to define and link VHDL libraries are different for various tools. The reader should check with the tool's manual on steps needed for defining and linking VHDL libraries.

A brief explanation on the template from Example 2:

- On the first two lines of the code is the declaration of the usage of ***std_logic_1164*** from the *IEEE* library.
- The *entity* portion contains the declaration input ports (***input1***, ***input2***, ***input3***, ***input4***) and output ports (***output1***, ***output2***, ***output3***) as `std_logic` type.
- The *architecture* portion contains the component instantiation of three AND gates, each with *entity* name ***AND_gate_ent*** and *architecture* name ***AND_gate_arch***.
- The *configuration* portion contains the *configuration* declaration of the instantiated components.

1.5 STRUCTURAL, BEHAVIORAL, AND SYNTHESIZABLE VHDL DESIGN STRUCTURE

In general, a VHDL design can be categorized into three different groups. Each has its own distinct characteristics and style of coding. The circuit in Fig. 5 shall be used as a reference to differentiate the coding style for each of these categories.

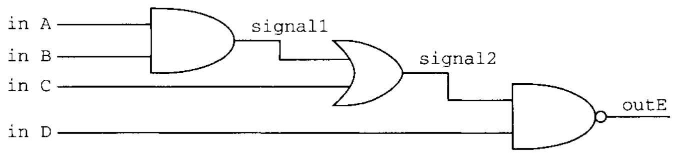


FIGURE 5 Figure Showing Schematic for "My Module."

1.5.1 Structural VHDL

Structural VHDL is a data type structure that is best described as a netlist of a design or schematic. It has declarations of all the types of components used in the design and interconnects to connect all the different components.

EXAMPLE 3 Structural VHDL Code for "My Module"

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY structural_code_ent IS
PORT (
        inA : IN std_logic;
        inB : IN std_logic;
        inC : IN std_logic;
        inD : IN std_logic;
        outE : OUT std_logic
     );
END structural_code_ent;

ARCHITECTURE structural_code_arch OF structural_code_ent IS

COMPONENT AND_gate_ent
PORT (
        AND_input1 : IN std_logic;
        AND_input2 : IN std_logic;
        AND_output : OUT std_logic
     );
END COMPONENT;

COMPONENT OR_gate_ent
PORT (
        OR_input1 : IN std_logic;
        OR_input2 : IN std_logic;
        OR_output : OUT std_logic
     );
END COMPONENT;
```

```
COMPONENT NAND_gate_ent
PORT (
        NAND_input1 : IN std_logic;
        NAND_input2 : IN std_logic;
        NAND_output : OUT std_logic
     );
END COMPONENT;

SIGNAL signal1, signal2 : std_logic;

BEGIN
        logic_AND: AND_gate_ent PORT MAP (inA, inB, signal1);
        logic_OR: OR_gate_ent PORT MAP (signal1, inC,
                                        signal2);
        logic_NAND: NAND_gate_ent PORT MAP (signal2, inD,
                                        outE);
END structural_code_arch;

CONFIGURATION structural_code_config OF structural_code_ent IS
FOR structural_code_arch
        FOR logic_AND: AND_gate_ent
             USE ENTITY WORK.AND_gate_ent(AND_gate_arch);
        END FOR;
        FOR logic_OR: OR_gate_ent
             USE ENTITY WORK.OR_gate_ent(OR_gate_arch);
        END FOR;
        FOR logic_NAND: NAND_gate_ent
             USE ENTITY WORK.NAND_gate_ent(NAND_gate_arch);
        END FOR;
END FOR;
END structural_code_config;
```

Note: The code from Example 3 assumes precompiled AND, OR and NAND gate in library ***WORK***.

1.5.2 Behavioral VHDL

This type structure describes the design in a behavioral manner, mimicking its performance and functionality. A design coded behaviorally is just a black box. The code is written in such a way as to generate the specified output signals for a given set of input signals. This form of coding is nonsynthesizable and is normally used only for system testing.

Using Fig. 5, a truth table (Table 1) representing the functionality of **"My Module"** is created. From this functionality, an output value is mapped to every input value and the code is then behaviorally coded according to the functionality mapped.

TABLE 1 Truth Table for "My Module"

inA	*inB*	*inC*	*InD*	*signal1*	*signal2*	*outE*
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	0	1
0	1	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	1
1	0	1	1	0	1	0
1	1	0	0	1	1	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	0

EXAMPLE 4 Behavioral Code for "My Module"

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY behavior_ent IS
PORT (
        inA : IN std_logic;
        inB : IN std_logic;
        inC : IN std_logic;
        inD : IN std_logic;
        outE : OUT std_logic
      );
END behavior_ent;
```

```
ARCHITECTURE behavior_arch OF behavior_ent IS
BEGIN

outE <= '0' WHEN (((inA = '1') AND (inB = '1') AND (inC =
                '0') AND (inD = '0'))
                OR ((inA = '1') AND (inB = '1') AND (inC = '0')
AND (inD = '1'))
                OR ((inA = '1') AND (inB = '1') AND (inC = '1')
AND (inD = '0'))
                OR ((inA = '1') AND (inB = '1') AND (inC = '1')
AND (inD = '1')))
              ELSE '1' WHEN(((inA = '0') AND (inB = '0') AND (inC =
'0') AND (inD = '0'))
                OR ((inA = '0') AND (inB = '0') AND (inC = '0')
AND (inD = '1'))
                OR ((inA = '0') AND (inB = '0') AND (inC = '1')
AND (inD = '0'))
                OR ((inA = '0') AND (inB = '0') AND (inC = '1')
AND (inD = '1'))
                OR ((inA = '0') AND (inB = '1') AND (inC = '0')
AND (inD = '0'))
                OR ((inA = '0') AND (inB = '1') AND (inC = '0')
AND (inD = '1'))
                OR ((inA = '0') AND (inB = '1') AND (inC = '1')
AND (inD = '0'))
                OR ((inA = '0') AND (inB = '1') AND (inC = '1')
AND (inD = '1'))
                OR ((inA = '1') AND (inB = '0') AND (inC = '0')
AND (inD = '0'))
                OR ((inA = '1') AND (inB = '0') AND (inC = '0')
AND (inD = '1'))
                OR ((inA = '1') AND (inB = '0') AND (inC = '1')
AND (inD = '0'))
                OR ((inA = '1') AND (inB = '0') AND (inC = '1')
AND (inD = '1')))
ELSE 'H' WHEN ((inA='X') OR (inB='X') OR (inC='X') OR
                (inD='X'))
ELSE 'L' WHEN ((inA='U') OR (inB='U') OR (inC='U') OR
                (inD='U'))
ELSE 'Z' AFTER 1.5 ns; — propagation delay is 1.5 ns
```

This is sensitivity list for the PROCESS. See the **Note.**

```
PROCESS (inA, inB, inC, inD)
BEGIN
         IF ((inA='U') OR (inB='U') OR (inC='U') OR (inD='U'))
                THEN
```

```
              ASSERT FALSE
              REPORT "One of the inputs is at 'U'. Output is
              driven as L."
              SEVERITY WARNING;
        ELSIF ((inA='X') OR (inB='X') OR (inC='X') OR
              (inD='X')) THEN
              ASSERT FALSE
              REPORT "One of the inputs is at 'X'. Output is
              driven as H."
              SEVERITY WARNING;
        END IF;
END PROCESS;

END behavior_arch;
```

Note: Sensitivity list is the list of signals whereby the sequential ***PROCESS*** would be triggered/executed when there is a change in any of the signals listed in the sensitivity list. Therefore it is important that any signals that are used in the ***PROCESS*** be specified in the sensitivity list.

1.5.3 RTL Code

This is the most complicated form of coding as it describes a design in a high-level manner through a subset of VHDL syntax. This form of coding is somewhere between structural and behavioral code. It is at a higher level of description compared to structural VHDL but at a lower level of description compared to that of behavioral VHDL.

EXAMPLE 5 Descriptive VHDL Code for "My Module"

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

IEEE library declaration

```
ENTITY descriptive_ent IS
PORT (
        inA : IN std_logic;
        inB : IN std_logic;
        inC : IN std_logic;
        inD : IN std_logic;
        outE : OUT std_logic
     );
END descriptive_ent;
```

Port declaration as input/output and type

Description of ***"My Module"*** in synthesizable code

```
ARCHITECTURE descriptive_arch OF descriptive_ent IS
BEGIN
        PROCESS (inA, inB, inC, inD)
        BEGIN
             IF (((inA = '0') AND (inB = '0') AND (inC = '1')
                    AND (inD = '1')) OR
                 ((inA = '0') AND (inB = '1') AND (inC = '1')
                    AND (inD = '1')) OR
                 ((inA = '1') AND (inB = '0') AND (inC = '1')
                    AND (inD = '1')) OR
                 ((inA = '1') AND (inB = '1') AND (inC = '0')
                    AND (inD = '1')) OR
                 ((inA = '1') AND (inB = '1') AND (inC = '1')
                    AND (inD = '1'))) THEN
                    outE <= '0';
             ELSE
                    outE <= '1';
             END IF;
        END PROCESS;
END descriptive_arch;
```

Note: There are many different styles to write RTL code. Example 5 is an example of an RTL coding style. However, there are simpler and more efficient styles to write the code for Example 5. Different designers often have different styles of coding.

Another style to write the code for Example 5:

```
PROCESS (inA, inB, inC, inD)
BEGIN
   IF ((((inA = '1') AND (inB = '1')) OR (inC = '1')) AND (inD =
   '1')) THEN
   OutE <= '0';
   ELSE
        OutE <= '1';
   END IF;
END PROCESS;
```

Did you notice how this style of coding is similar to the combinational logic gate diagram of Fig. 5?

By synthesizing Example 5, the same combinational logic gate of Fig. 5 is obtained.

To confirm that the results obtained from synthesizing Example 5 are synthetically the most optimum, a Karnaugh Map (Table 2) is generated from Table 1.

TABLE 2 Karnaugh Map for "My Module"

		InA & ***inB***			
		00	01	11	10
inC & ***inD***	00	1	1	1	1
	01	1	1	0	1
	11	0	0	0	0
	10	1	1	1	1

Optimized to
`NOT (D(AB + C))`

Optimized output of the Karnaugh map shows the same result as that of Fig. 5.

In dealing with real designs, the reader needs to concern himself/herself only with writing synthesizable VHDL code. Synthesis optimizations can be performed in a sort of "automated way" using synthesis tools such as Synopsys's Design Compiler. Synthesis methods to obtain optimal synthesis results are explained in detail in Chapter 8.

1.6 USAGE OF LIBRARY DECLARATIONS IN VHDL DESIGN STRUCTURE

You might have already noticed that the earlier examples have the following two lines of the code at the beginning of the VHDL code.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL
```

These two lines declare the usage of functions and procedures from the library ***std_logic_1164***.

The library ***std_logic_1164*** also contains declarations for different types used to declare a port, *signal,* or *variable* in VHDL. Several examples are:

- ***BIT*** — declaration of this type may only have a value of '0' or '1';
- ***Boolean*** — declaration of this type can only have value of *FALSE* or *TRUE*;
- ***Integer*** — declaration of this type allows an integer value ranging from negative (2^{31} –1) to positive (2^{31} –1).
- ***Std_ulogic*** — declarations of this type can have 9 different values; they are
  - '1' — forcing logical '1'
  - '0' — forcing logical '0'
  - 'H' — weak '1'
  - 'L' — weak '0'
  - 'X' — forcing unknown
  - 'U' — uninitialized
  - 'Z' — high impedance
  - '—' — don't care
  - 'W' — weak unknown
- ***Std_logic*** — this is the resolved version of ***std_ulogic*** type. This is the most commonly used type in synthesizable code. It is advisable to the designer

to try to use only one type when coding synthesizable VHDL. The usage of just one type would reduce the necessity of using conversion functions to convert from one type to another before integrating different modules together.

Apart from the library ***std_logic_1164***, ***std_logic_arith*** is another commonly used ***IEEE*** library.

Std_logic_arith contain functions that are very useful in VHDL, especially when a designer is integrating modules/submodules that have different port/signal type. In cases like these, conversion functions are needed to convert the port/signal from one type to another. For example, the function ***CONV_INTEGER*** converts a signal to type ***INTEGER***. Another example is ***CONV_STD_LOGIC_VECTOR***, which converts a signal to type ***STD_LOGIC_VECTOR***.

Example 6 shows how a design can use the function ***CONV_STD_LOGIC_VECTOR*** to convert a signal "***internal***" from type ***INTEGER*** to type ***STD_LOGIC_VECTOR***.

EXAMPLE 6 Using the Conversion Function in Writing Synthesizable VHDL

IEEE library

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY convert_ent IS
PORT (
        input : IN std_logic_vector (1 downto 0);
   ........ other inputs ........
```

Port declaration as input/output and port type

```
        output : OUT std_logic_vector(3 downto 0);
   .......... other outputs ..........
        );
END convert_ent;
```

Declaration of *internal* as *integer* type

```
ARCHITECTURE convert_arch OF convert_ent
SIGNAL internal : INTEGER;
BEGIN
..................
        PROCESS (........list of sensitivity list ..........)
        BEGIN
................ VHDL code to assign value to signal internal ........
        END PROCESS;
...................
```

Conversion of *integer* to 4-bit *std_logic_vector*

```
output <= CONV_STD_LOGIC_VECTOR(internal,4);
END convert_arch;
```

To get a better idea of how an ***IEEE*** library code looks, you can refer to ***Appendix A***, which consists of the VHDL code for the library ***std_logic_1164***.

2

VHDL SIMULATION AND SYNTHESIS FLOW

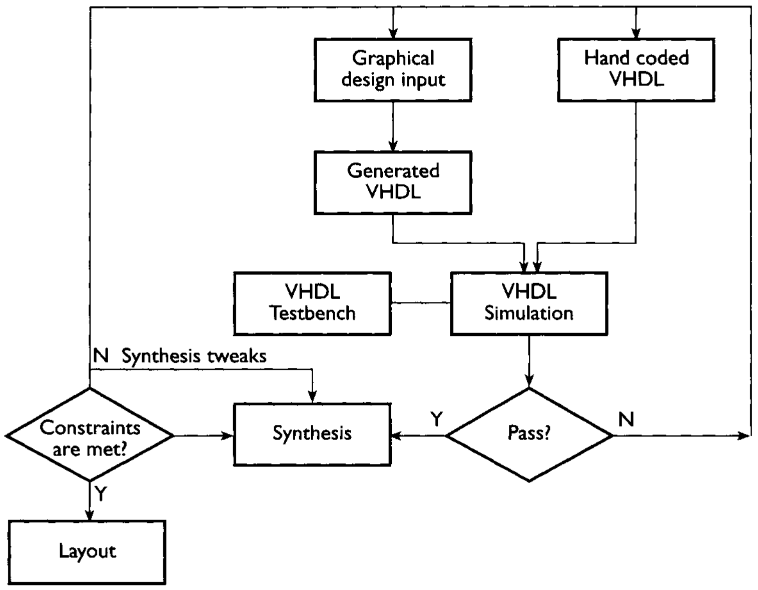


FIGURE 6 Diagram Showing a Generic HDL Simulation and Synthesis Flow.

The generic HDL simulation and synthesis flow as diagrammed in Fig. 6 is divided into three major portions.

- ***Coding*** This is the beginning of the flow where a designer writes the code for a specific design. This code can either be hand-written or tool generated. Several of the more commonly used tools to generate HDL code are Summit's Visual HDL and Mentor Graphics' Renoir.

- ***Simulation*** Simulation is performed when the HDL code is completed and the designer is ready to simulate the design. This portion of the flow concentrates on checking the functionality of the coded design. During simulation, a common practice to exercise the functionality of a design is to use a testbench (see Fig. 7). A testbench is a "wrap-around" of the design whereby input stimulus are injected into the design while monitoring for expected output waveforms. The simulation result shows an error in the design if the output signals do not match the expected waveforms. When this occurs, the designer must then move back to the HDL coding phase, whereby the code is changed to fix the functional mismatch between the output signals and the expected waveform. This act of simulation and recoding is performed in a loop until the design's output signals match the expected waveform.
- ***Synthesis*** When the designer is satisfied with the design and has completed simulation, the next step would be to synthesize the design. Synthesis is also iterated in a loop until the synthesized design meets certain design specifications. Several of the more common criteria to be evaluated are performance and area utilization. If these criteria are not met, the synthesis tool can be used to perform more optimization on the synthesized database. If upon optimization, the design still does not meet the mentioned criteria, the microarchitecture implementation of the design must be changed. The designer will then proceed to recode the HDL. When constraints are met, the synthesized database and timing requirement are passed over to layout.

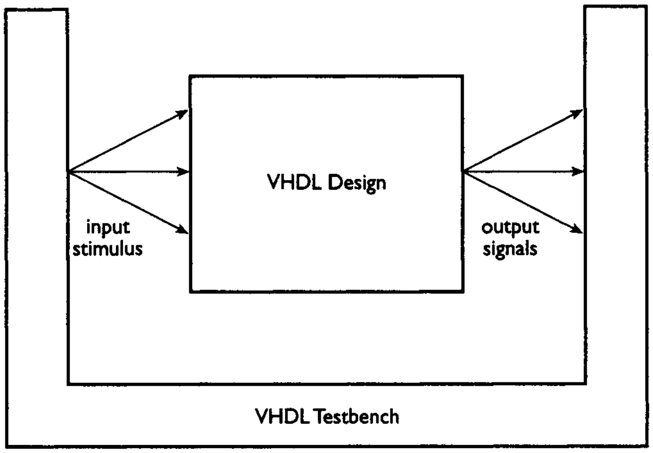


FIGURE 7 Diagram Showing a Testbench around a VHDL Design.

3

SYNTHESIZABLE CODE FOR BASIC LOGIC COMPONENTS

The RTL synthesizable code is written using a simple high-level descriptive manner of the design's functionality. The following examples are synthesizable code for very basic logic components. Most of these basic logic components, such as AND gate, OR gate and NOT gate can be coded in VHDL by using VHDL keywords such as AND, OR, NOT and others. The synthesis tool can recognize these keywords and map them to logic gates. However as a start to writing synthesizable VHDL code, these basic logic components are coded in RTL form to give the reader a basic idea of how synthesizable code may look.

3.1 AND LOGIC

To begin coding an AND logic, first generate a truth table for the logic.

TABLE 3 Truth Table for AND Logic Function

Input1	*Input2*	*output1*
0	0	0
0	1	0
1	0	0
1	1	1

EXAMPLE 7 Example of AND Logic Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY and_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        output1 : OUT std_logic
     );
END and_ent;

ARCHITECTURE and_arch OF and_ent IS
BEGIN
        PROCESS (input1, input2)
        BEGIN
             IF ((input1 = '1') AND (input2 = '1')) THEN
                  output1 <= '1';
             ELSE
                  output1 <= '0';
             END IF;
        END PROCESS;
END and_arch;
```

Notice that the **`output1`** value assignment is based on the truth table (Table 3) generated earlier for the ***AND*** logic. An easier way to code this would be to use the keyword ***AND***: **`output1 <= input1 AND input2;`**

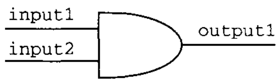


FIGURE 8 Diagram for Synthesized AND Logic.

3.2 OR LOGIC

For an OR logic, the output is a logical '1' if any one of the inputs has a logical '1' (see Table 4, the truth table). This feature of OR logic can be directly coded into VHDL to synthesize an OR logic as shown in Example 8.

TABLE 4 Truth Table for OR Logic Function

Input1	*Input2*	*Output1*
0	0	0
0	1	1
1	0	1
1	1	1

EXAMPLE 8 Example of OR Logic Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY or_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        output1 : OUT std_logic
     );
END or_ent;

ARCHITECTURE or_arch OF or_ent IS
BEGIN
        PROCESS (input1, input2)
        BEGIN
             IF ((input1 = '1') OR (input2 = '1')) THEN
                  output1 <= '1';
             ELSE
                  output1 <= '0';
             END IF;
        END PROCESS;
END or_arch;
```

Describing the function of an ***OR*** logic whereby if any of the input has a logical '1', the output is assigned to a logical '1' as well. An easier way to code this would be to use the keyword ***OR***: ***output1 <= input1 OR input2;***

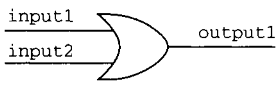


FIGURE 9 Diagram for Synthesized OR Logic.

3.3 NOT LOGIC

The NOT logic with the output always at an opposite logical state to the input is easily described as RTL code in Example 9. See Figure 10 for the synthesized NOT logic diagram.

EXAMPLE 9 Example of NOT Logic Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY not_ent IS
PORT (
        input1 : IN std_logic;
        output1 : OUT std_logic
     );
END not_ent;

ARCHITECTURE not_arch OF not_ent IS
BEGIN
        PROCESS (input1)
        BEGIN
             IF (input1 = '1') THEN
                  output1 <= '0';
             ELSE
                  output1 <= '1';
             END IF;
        END PROCESS;
END not_arch;
```

Description to invert the input to generate the output. An easier way is to use the keyword ***NOT***.
output1 <= NOT (input1);

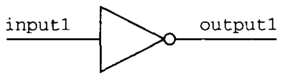


FIGURE 10 Diagram for Synthesized NOT Logic.

3.4 NAND LOGIC

A NAND logic functions like an AND logic connected to a NOT logic. Example 10 describes the results of the truth table (Table 5) for the NAND logic.

For an AND logic, the output is a logical '1' when all the inputs are at logical '1'. However for NAND logic, the output is at a logical '0' if all the inputs are at a logical '1'.

TABLE 5 Truth Table for a NAND Logic Function

Input1	*Input2*	*output1*
0	0	1
0	1	1
1	0	1
1	1	0

EXAMPLE 10 Example of NAND Logic Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY nand_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        output1 : OUT std_logic
     );
END nand_ent;

ARCHITECTURE nand_arch OF nand_ent IS
BEGIN
        PROCESS (input1, input2)
        BEGIN
             IF ((input1 = '1') AND (input2 = '1')) THEN
                  output1 <= '0';
             ELSE
                  output1 <= '1';
             END IF;
        END PROCESS;
END nand_arch;
```

The logic values driven by the output of the **NAND** logic are directly opposite to that of the **AND** logic. An easier way to code this would be: **`output1 <= NOT (input1 AND input2);`**

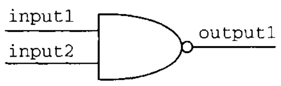


FIGURE 11 Diagram for Synthesized NAND Logic Gate.

3.5 NOR LOGIC

Similar to the NAND logic, which is the logical opposite of the AND logic, the NOR logic is the logical opposite of the OR logic (see Table 6).

TABLE 6 Truth Table for a NOR Logic Function

Input1	*input2*	*Output1*
0	0	1
0	1	0
1	0	0
1	1	0

EXAMPLE 11 Example of NOR Logic Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY nor_ent IS
PORT (
        input1 : IN std_logic;
        input2 IN std_logic;
        output1 : OUT std_logic
     ); END nor_ent;

ARCHITECTURE nor_arch OF nor_ent IS
BEGIN
        PROCESS (input1, input2)
        BEGIN
             IF ((input1 = '1') OR (input2 = '1')) THEN
                  output1 <= '0';
             ELSE
                  output1 <= '1';
             END IF;
        END PROCESS;
END nor_arch;
```

The logic values driven by the output are directly opposite to that of the ***OR*** logic. By using the keyword ***OR***: ***output1 <= NOT (input1 OR input2);***

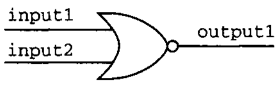


FIGURE 12 Diagram for Synthesized NOR Logic.

3.6 TRISTATE BUFFER LOGIC

A tristate buffer is used within designs that consist of tristate buses (see Table 7). When the *selector* pin of a tristate buffer is at a logical '1', the output will drive a logical equivalent of the input. If the selector pin is a logical '0', the output is high impedance.

TABLE 7 Truth Table for Tristate Buffer Logic Function

input	*selector*	*output*
0	0	Z
0	1	0
1	0	Z
1	1	1

EXAMPLE 12 Example of Tristate Buffer Logic Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY tristate_ent IS
PORT (
        input1 : IN std_logic;
        selector : IN std_logic;
        output1 : OUT std_logic
     );
END tristate_ent;
ARCHITECTURE tristate_arch OF tristate_ent IS
BEGIN
        PROCESS (input1, selector)
        BEGIN
             IF (selector = '1') THEN
                  output1 <= input1;
             ELSE
                  output1 <= 'Z';
             END IF;
        END PROCESS;
END tristate_arch;
```

Description of tristate buffer where **input1** is driven at **output1** when **selector** is at logical '1'.

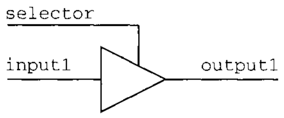


FIGURE 13 Diagram for Synthesized Tristate Buffer Logic.

3.7 COMPLEX LOGIC GATE

To write the code for a complex logic gate, it is important to know the functionality of the complex gate. This can be achieved by using a truth table to map the output for all possible logical input. Once the truth table is defined, the code can then be written to describe the output of the complex gate based on a given set of input patterns.

Truth table (Table 8) shows the functionality of a complex gate. From this table, the RTL VHDL code is written as shown in Example 13.

TABLE 8 Truth Table for a Complex Logic Gate Function

input1	*input2*	*input3*	*output1*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

EXAMPLE 13 Example of a Complex Logic Gate Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY complex_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        input3 : IN std_logic;
        output1 : OUT std_logic
     );
END complex_ent;

ARCHITECTURE complex_arch OF complex_ent IS
BEGIN
        PROCESS (input1, input2, input3)
        BEGIN
             IF (input3 = '1') THEN
                  output1 <= '1';
             ELSIF (input3 = '0') THEN
                  IF ((input1 = '1') AND (input2 = '1')) THEN
                       output1 <= '1';
```

Description of functionality of the complex logic gate

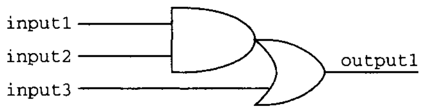


FIGURE 14 Diagram for Synthesized Complex Logic Gate.

```
                    ELSE
                        output1 <= '0';
                    END IF;
                ELSE
                    output1 <= '0';
                END IF;
            END PROCESS;
END complex_arch;
```

> *Note:* A simpler way to write this code would be to use the keywords AND and OR:
> `Output1 <= ((input1 AND input2) OR (input3));`

3.8 LATCH

A latch allows the input data to be passed on to the output whenever the clock is at a logical '1'. However, the last value at the output of the latch is kept throughout the entire period of the clock when low.

EXAMPLE 14 Example of Latch Synthesized Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY latch_ent IS
PORT (
        data_in : IN std_logic;
        clock : IN std_logic;
        data_out : OUT std_logic
    );
END latch_ent;

ARCHITECTURE latch_arch OF latch_ent IS
BEGIN
```

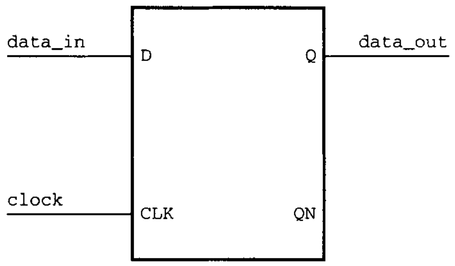


FIGURE 15 Diagram for Synthesized Latch.

Clock is used to latch in the input data. When being synthesized, the term `clock = '1'` will allow the synthesis tool to know that a latch is required.

```
          PROCESS (data_in, clock)
          BEGIN
               IF (clock = '1') THEN
                    data_out <= data_in;
               END IF;
          END PROCESS;
END latch_arch;
```

3.8.1 Avoiding Latches in Your Code

Many designers who are learning to code synthesizable VHDL very often find that their designs have unwanted latches. These latches are automatically inferred by a synthesis tool into the design if a designer misses several important structures in the code.

Latch inference occurs when a designer has written VHDL code that does not have complete coverage. For example, when using an ***IF*** statement that does not cover all possible logical combinations, latches will be inferred.

EXAMPLE 15 Example of Code of *IF* Statement Inferring Latch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY ifwl_ent IS
PORT (
          input1 : IN std_logic;
          input2 : IN std_logic;
          input3 : IN std_logic;
          input4 : IN std_logic;
          selector : IN std_logic_vector (1 downto 0);
          output1 : OUT std_logic
     );
END ifwl_ent;
```

```
ARCHITECTURE ifwl_arch OF ifwl_ent IS
SIGNAL internal : std_logic := '0';
BEGIN
        PROCESS (input1, input2, input3, input4, selector)
        BEGIN
             IF (selector = "00") THEN
                  internal <= input1;
             ELSIF (selector = "01") THEN
                  internal <= input2;
             ELSIF (selector = "10") THEN
                  internal <= input3;
             END IF;
        END PROCESS;
output1 <= internal;
END ifwl_arch;
```

There is no complete coverage of possible selector combinations. Therefore latch will be inferred.

EXAMPLE 16 Example of Code of *IF* Statement without Inferring a Latch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY ifwol_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        input3 : IN std_logic;
        input4 : IN std_logic;
        selector : IN std_logic_vector (1 downto 0);
        output1 : OUT std_logic
     );
END ifwol_ent;

ARCHITECTURE ifwol_arch OF ifwol_ent IS
SIGNAL internal : std_logic := '0';
BEGIN
        PROCESS (input1, input2, input3, input4, selector)
        BEGIN
             IF (selector = "00") THEN
                  internal <= input1;
             ELSIF (selector = "01") THEN
                  internal <= input2;
             ELSIF (selector = "10") THEN
                  internal <= input3;
             ELSIF (selector = "11") THEN
```

All possible combinations of selector are identified. No latch will be inferred in this code.

```
                internal <= input4;
            END IF;
        END PROCESS;
output1 <= internal;
END ifwol_arch;
```

Latch inference also occurs when a designer uses a ***CASE*** statement but does not cover all the possible logical conditions of the ***CASE***.

EXAMPLE 17 Example of Code of *CASE* Statement Inferring a Latch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY casewl_ent IS
PORT (
      input1 : IN std_logic;
      input2 : IN std_logic;
      input3 : IN std_logic;
      input4 : IN std_logic;
      selector : IN std_logic_vector (2 downto 0);
      output1 : OUT std_logic
      );
END casewl_ent;

ARCHITECTURE casewl_arch OF casewl_ent IS
BEGIN
        PROCESS (input1, input2, input3, input4, selector)
        BEGIN
            CASE selector IS
                WHEN "000" =>
                    output1 <= input1;
                WHEN "001" =>
                    output1 <= input2;
                WHEN "010" =>
                    output1 <= input3;
                WHEN OTHERS =>
                    NULL;
            END CASE;
        END PROCESS;
END casewl_arch;
```

Not all possible ***CASE*** conditions are specified. As such a latch is inferred to store the previous value of ***output1***.

EXAMPLE 18 Example of Code of *CASE* Statement without Inferring a Latch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY casewol_ent IS
PORT (
      input1 : IN std_logic;
      input2 : IN std_logic;
      input3 : IN std_logic;
      input4 : IN std_logic;
      selector : IN std_logic_vector (2 downto 0);
      output1 : OUT std_logic
      );
END casewol_ent;

ARCHITECTURE casewol_arch OF casewol_ent IS
BEGIN
       PROCESS (input1, input2, input3, input4, selector)
       BEGIN
            CASE selector IS
                 WHEN "000" =>
                      output1 <= input1;
                 WHEN "001" =>
                      output1 <= input2;
                 WHEN "010" =>
                      output1 <= input3;
                 WHEN "011" =>
                      output1 <= input4;
                 WHEN OTHERS =>
                      Output1 <= '0';
            END CASE;
       END PROCESS;
END casewol_arch;
```

All possible conditions for the *CASE* statement are specified. Thus the synthesized circuit is purely logical and no latch is inferred.

Apart from using full coverage in the *IF* statement or *CASE* statement to avoid inference of latches, the designer can also assign default values to a signal to ensure that no latch is inferred. However, using full coverage of *IF/CASE* statements is a good practice to follow in VHDL coding.

3.9 FLIP-FLOP

Flip-flop is similar to a latch except that a flip-flop only latches in the input when the clock transitions from a low to high (positive edge triggered flip-flop) or from high to low (negative edge triggered flip-flop). See Fig. 16 for a diagram of synthesized flip-flop.

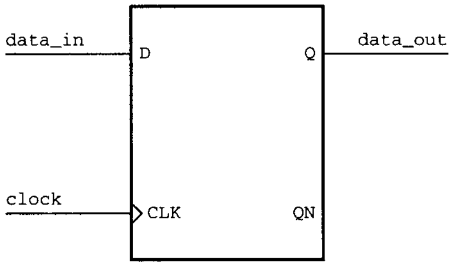


FIGURE 16 Diagram for Synthesized Flip-Flop.

EXAMPLE 19 Example of Flip-Flop Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY flop_ent IS
PORT (
        data_in : IN std_logic;
        clock : IN std_logic;
        data_out : OUT std_logic
     );
END flop_ent;

ARCHITECTURE flop_arch OF flop_ent IS
BEGIN
        PROCESS (data_in, clock)
        BEGIN
             IF (clock = '1' and clock'EVENT) THEN
                  data_out <= data_in;
             END IF;
        END PROCESS;
END flop_arch;
```

Indication of ***clock*** transition from logic '0' to logic '1'. If negative edge is required, simply use ***"clock = '0' and clock 'EVENT"***

3.10 DECODER

The truth table for a decoder with 3 inputs and 8 outputs is created as shown in Table 9. The code is then written to describe the output results of the truth table. See Fig. 17 for a synthesized decoder diagram.

TABLE 9 Truth Table for a Decode Logic Function

input(2)	*input(1)*	*input(0)*	*Output(7 downto 0)*
0	0	0	00000001
0	0	1	00000010
0	1	0	00000100
0	1	1	00001000
1	0	0	00010000
1	0	1	00100000
1	1	0	01000000
1	1	1	10000000

EXAMPLE 20 Example of Decoder Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY decoder_ent IS
PORT (
        input : IN std_logic_vector (2 downto 0);
        output : OUT std_logic_vector (7 downto 0)
     );
END decoder_ent;

ARCHITECTURE decoder_arch OF decoder_ent IS
BEGIN
        PROCESS (input)
        BEGIN
             CASE input IS
                  WHEN "000" =>
                       output <= "00000001";
                  WHEN "001" =>
                       output <= "00000010";
                  WHEN "010" =>
                       output <= "00000100";
                  WHEN "011" =>
                       output <= "00001000";
                  WHEN "100" =>
                       output <= "00010000";
                  WHEN "101" =>
                       output <= "00100000";
                  WHEN "110" =>
                       output <= "01000000";
```

Assignment value for each combination of input

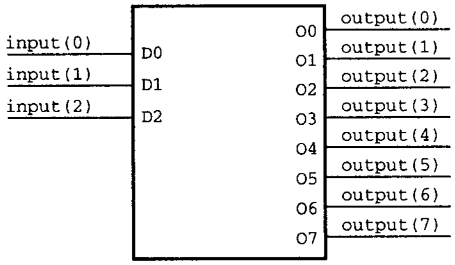


FIGURE 17 Diagram for Synthesized Decoder.

```
                WHEN "111" =>
                    output <= "10000000";
                WHEN OTHERS =>
                    NULL;
            END CASE;
        END PROCESS;
END decoder_arch;
```

3.11 ENCODER

An encoder has the opposite functionality of a decoder. For an encoder with 2^N input, it would encode the input into an N-bit output. For example, an 8-bit input encoder will have 3 output bits. See Table 10 and Fig. 18.

TABLE 10 Truth Table for an 8-Bit Input Encoder

input(7 downto 0)	output(2)	output(1)	output(0)
00000001	0	0	0
00000010	0	0	1
00000100	0	1	0
00001000	0	1	1
00010000	1	0	0
00100000	1	0	1
01000000	1	1	0
10000000	1	1	1

EXAMPLE 21 Example of Encoder Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```
ENTITY encoder_ent IS
PORT (
        input : IN std_logic_vector (7 downto 0);
        output : OUT std_logic_vector (2 downto 0)
     );
END encoder_ent;

ARCHITECTURE encoder_arch OF encoder_ent IS
BEGIN
        PROCESS (input)
        BEGIN
             IF (input = "00000001") THEN
                  output <= "000";
             ELSIF (input = "00000010") THEN
                  output <= "001";
             ELSIF (input = "00000100") THEN
                  output <= "010";
             ELSIF (input = "00001000") THEN
                  output <= "011";
             ELSIF (input = "00010000") THEN
                  output <= "100";
             ELSIF (input = "00100000") THEN
                  output <= "101";
             ELSIF (input = "01000000") THEN
                  output <= "110";
             ELSIF (input = "10000000") THEN
                  output <= "111";
             ELSE
                  Output <= "000";
             END IF;
        END PROCESS;
END encoder_arch;
```

For each input stimulus, a set of output patterns is assigned to the output.

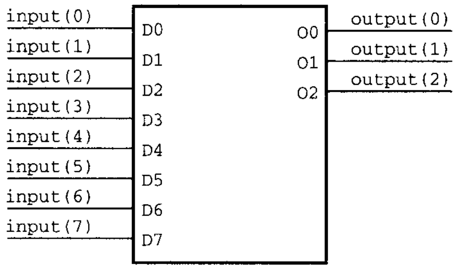


FIGURE 18 Diagram for Synthesized Encoder.

3.12 MULTIPLEXER

Multiplexers can range from 2 to 4 to 8 to 16 inputs and so on. In general, for a 2^N input multiplexer, an N-bits-wide selector is needed. The selector field is used as a mechanism to select the input to be directed to the output of the multiplexer (see Table 11 and Fig. 19).

For a 4-input multiplexer, a 2-bits selector field is required.

TABLE 11 Truth Table for a Multiplexer Logic Function

selector	Output1
00	Input1
01	Input2
10	Input3
11	Input4

EXAMPLE 22 Example of Multiplexer Logic Synthesizable Code

CASE statement to describe output selections for different selector values.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY mux_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        input3 : IN std_logic;
        input4 : IN std_logic;
        selector : IN std_logic_vector (1 downto 0);
        output1 : OUT std_logic
     );
END mux_ent;

ARCHITECTURE mux_arch OF mux_ent IS
BEGIN
        PROCESS (input1, input2, input3, input4, selector)
        BEGIN
            CASE selector IS
                WHEN "00" =>
                    output1 <= input1;
                WHEN "01" =>
                    output1 <= input2;
                WHEN "10" =>
                    output1 <= input3;
```

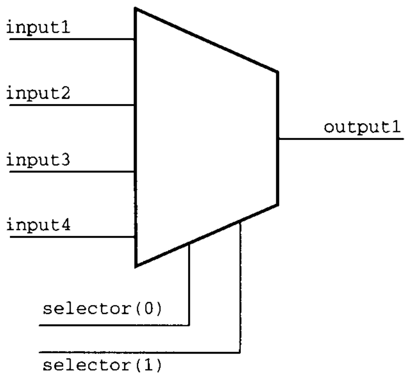


FIGURE 19 Diagram for Synthesized Multiplexer Logic.

```
                    WHEN "11" =>
                         output1 <= input4;
                    WHEN OTHERS =>
                         NULL;
               END CASE;
          END PROCESS;
END mux_arch;
```

If an 8-bit input multiplexer is needed, the **selector** bits are enlarged to 3 bits ($2^3 = 8$). The ***CASE*** statement in the VHDL code is also enlarged to cover all the possible logical combinations involving the 3-bit selector field.

The reader must be aware that when using large/wide multiplexers, Design Compiler can sometimes fail to translate VHDL code into multiplexers. Design Compiler would instead build the logic using combinational logic gates. In order to ensure that Design Compiler is able to synthesize a large/wide multiplexer, it is advisable for the designer to use GTECH components for these large/wide multiplexers. The GTECH components are technology independent but functionally accurate components that are mapped into logic gates by Design Compiler. If the equivalent logic gates do not exist in the synthesis library, then Design Compiler will use existing logic gates in the library to build a logic component with the same function. The GTECH components are explained in detail in Chapter 9.

3.13 PRIORITY ENCODER

A priority encoder can be coded using IF statement with the first condition in the IF statement the signal with the highest priority.

TABLE 12 Truth Table For a Single-Bit Selector Priority Encoder

input1	input2	input3	input4	Output1
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

EXAMPLE 23 Example of Priority Encoder Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY priority_ent IS
PORT (
        input1 : IN std_logic;
        input2 : IN std_logic;
        input3 : IN std_logic;
        input4 : IN std_logic;
        output1 : OUT std_logic
     );
END priority_ent;

ARCHITECTURE priority_arch OF priority_ent IS
BEGIN
        PROCESS (input1, input2, input3, input4)
        BEGIN
             IF (input1 = '1') THEN
                  output1 <= '1';
             ELSIF (input2 = '1') THEN
```

input1 has highest priority

```
                output1 <= '0';
            ELSIF (input3 = '1') THEN
                output1 <= '1';
            ELSIF (input4 = '1') THEN
                output1 <= '0';
            ELSE
                output1 <= '1';
            END IF;
        END PROCESS;
END priority_arch;
```

input4 has lowest priority

Priority encoders are often used when the designer knows exactly which signal arriving at the priority encoder is late compared to the other signals. With reference to Example 23, from the synthesized circuit of Fig. 20, **input1** (the highest priority signal) is obviously a late arriving signal compared to **input2**, **input3**, and **input4**. Therefore, the logic has **input2**, **input3**, and **input4** being decoded in advance while awaiting the late arriving signal **input1**.

Figure 21 shows 5 input signals (***A***, ***B***, ***C***, ***D*** and ***E***) going to a logic component. Signal ***E*** is the late arriving signal compared to ***A***, ***B***, ***C*** and ***D***. For cases like these, it is preferable for the designer to use priority encoders. The VHDL code can be written with IF statement with the first condition in the statement being the signal with highest priority (in this case ***E***).

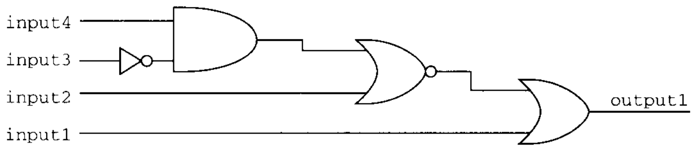


FIGURE 20 Diagram for Synthesized Priority Encoder.

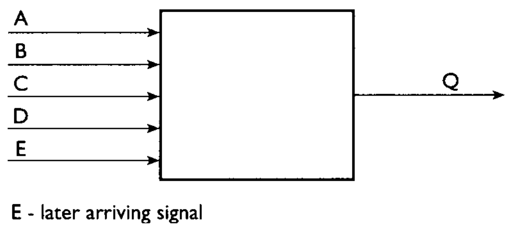


FIGURE 21 Diagram Showing a Logic Component with Late Arriving Signal ***E***.

3.14 MEMORY CELL

To design a memory cell, the designer can use a flip-flop and a multiplexer to represent one bit of memory. Writing RTL VHDL code for such a design is relatively easy. See Fig. 22 for the diagram of a memory cell.

Example 24 shows the code that is used to synthesize a memory cell. Notice how it only describes passing of **data_in** to **data_out** when the **write** port is at a logical '1', which would occur only during the rising edge of **clock**.

EXAMPLE 24 Example of Synthesizable Code for a Memory Cell

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY memory_ent IS
PORT (
        data_in : IN std_logic;
        clock : IN std_logic;
        write : IN std_logic;
        data_out : OUT std_logic
     );
END memory_ent;

ARCHITECTURE memory_arch OF memory_ent IS
BEGIN
        PROCESS (clock)
        BEGIN
             IF (clock = '1' AND clock'EVENT) THEN
                  IF (write = '1') THEN
                       data_out <= data_in;
                  END IF;
             END IF;
        END PROCESS;
END memory_arch;
```

Clock transitions from low to high

Qualifier signal **write** must be a '1' to allow **data_in** to be passed to **data_out**.

Such a memory cell design seems to be a waste of silicon area. Using transistors to design a 1-bit memory cell seems much more economical compared to using synthesizable VHDL. However, the schematic in Fig. 22 is normally used in designs that require storage of bits. In other words, it is used in designs that require memory cells for storage of certain bits of data.

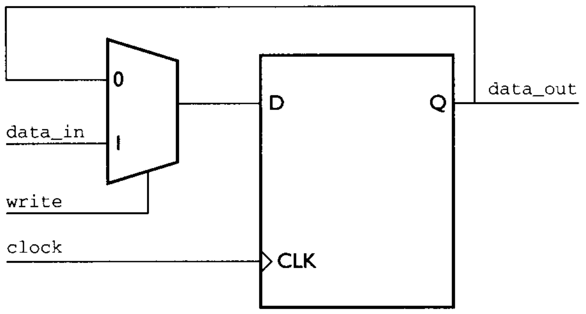


FIGURE 22 Diagram for Synthesized Memory Cell Using Flip-Flop.

3.15 ADDER

Adders are basic logic components that are used in almost every single kind of design. Most synthesis libraries customized for a certain synthesis tool more often than not will have precompiled adders in them. Utilizing these is easily done in VHDL code by a term called **`component inference`** (refer to Chapter 3.16).

If a design needs to use an adder, the VHDL code can use the sign '+'. The synthesis tool will infer an adder upon seeing this symbol.

For the sake of learning synthesizable VHDL, Example 25 shows a descriptive way to describe an adder with the logic function of Table 13. With this code, the synthesis tool will build an adder out of logic gates. The adder's functionality is as described in the code. See Fig. 23 for the block diagram of a synthesized adder.

TABLE 13 Truth Table for an Adder Logic Function

A	*B*	*Cin*	*Sum*	*Cout*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EXAMPLE 25 Example of Adder Synthesizable Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```
ENTITY adder_ent IS
PORT (
        A : IN std_logic;
        B : IN std_logic;
        Cin : IN std_logic;
        Sum : OUT std_logic;
        Cout : OUT std_logic
     );
END adder_ent;

ARCHITECTURE adder_arch OF adder_ent IS
BEGIN
        Sum <= '1' WHEN (((A='0') AND (B='0') AND (Cin='1')) OR
                   ((A='0') AND (B='1') AND (Cin='0')) OR
                   ((A='1') AND (B='0') AND (Cin='0')) OR
                   ((A='1') AND (B='1') AND (Cin='1')))
             ELSE '0';

        Cout <= '1' WHEN (((A='0') AND (B='1') AND (Cin='1')) OR
                   ((A='1') AND (B='0') AND (Cin='1')) OR
                   ((A='1') AND (B='1') AND (Cin='0')) OR
                   ((A='1') AND (B='1') AND (Cin='1')))
             ELSE '0';

END adder_arch;
```

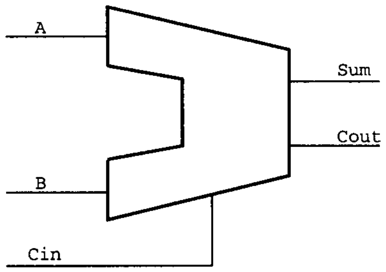


FIGURE 23 Block Diagram for a Synthesized Adder.

Figure 24 shows one method of using logic gates to build an adder. There are many other different methods using logic gates to build an adder. Which method used would differ from one synthesis tool to another, depending on the in-built synthesis algorithm. The different sets of design constraints set upon the design will also influence the outcome of the synthesized schematics of the adder.

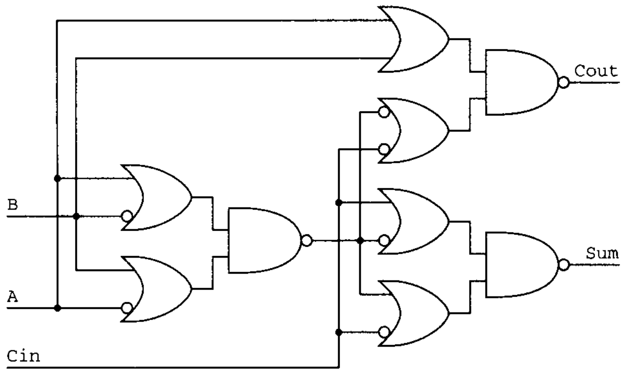


FIGURE 24 Schematic Diagram of an Adder Built of Logic Gates.

However for synthesis of a large adder that requires many bits, using combinational logic gates to build an adder is simply inefficient. It would be much easier for the designer to infer an adder using keyword `'+'`. Component inference is discussed in the next section (3.16).

3.16 COMPONENT INFERENCE

Many examples have been shown in Chapter 3. Each of these examples describes the functionality of a logic component to allow the synthesis tool to synthesize that logic component. However, in real-life design, many of these examples (description of basic logic gates) are not needed.

For example, if a designer needs to use a NOR gate, he/she does not need to write RTL code for a NOR gate. Similarly, this is true for other logic gates such as the AND gate and NOT gate.

These examples are given merely to allow the reader to gain a better understanding of how synthesizable code is written.

Many synthesis tools have a precompiled synthesis library containing all of these basic logic gates. The designer needs only to use certain symbols (or keywords) to utilize these precompiled logic gates. This form of usage of precompiled logic gates is referred to as ***component inference***.

Example 26 shows how a code can be used to infer an AND gate that already exists in the precompiled synthesis library.

EXAMPLE 26 VHDL Code Showing AND Gate Inference

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY design_module_ent IS
PORT (
      input1 : IN std_logic;
        input2 : IN std_logic;
        output : OUT std_logic
     );
END design_module_ent;

ARCHITECTURE design_module_arch OF design_module_ent IS
SIGNAL internal : std_logic;
BEGIN
        output <= input1 AND input2;
END design_module_arch;
```

Inference of an AND gate through the usage of the keyword AND.

Upon seeing the keyword AND, the synthesis tool would infer an AND logic gate.

Table 14 lists the other symbols/keywords that are often used for precompiled component inference from the synthesis tool's library.

TABLE 14 Symbols/Keywords Used for Logic Component Inference

Symbol/Keyword	Inferred Component
AND	AND gate
OR	OR gate
NOT	NOT gate
NOR	NOR gate
NAND	NAND gate
+	Adder
–	Subtractor
*	Multiplier
/	Divider

Note: Keep in mind that component inference using symbols/keywords differs from one synthesis tool to another.

4 SIGNAL VERSUS VARIABLE

Variables and signals are often assigned and used widely in VHDL code. Referring to the many examples in Chapter 3, note that some examples use ***signal*** declarations while others use ***variable*** declarations. Each declaration is used for different purposes. Whichever is used depends on what is to be achieved by the code.

In essence, a VHDL simulator uses simulation ticks to evaluate a piece of code and determine the conditions of the design being simulated. By using ***signal*** and ***variable***, the designer is able to control the update of the design information based on those simulation ticks.

In general, a rule of thumb to remember is that the assigned value of a ***variable*** is instantaneous, while for a ***signal*** the assigned value occurs only on the next simulation tick. This might not appear to be a big difference but it can cause quite a number of problems when a ***variable*** or ***signal*** is not used correctly.

4.1 VARIABLE

For Example 27, the **variable var** is incremented by one when input is “00”.

```
IF (input = "00") THEN
var := var + 1;
```

The assignment (**var := var + 1**) is instantaneous. Similarly for the ***IF*** statement that detects when ***var*** reaches 15, ***var*** is reset to 0 immediately.

```
IF (var = 15) THEN
var := 0;
```

The designer must also bear in mind that the usage of ***variable*** is local to a ***process***. Therefore such declaration for a ***variable*** must be within the ***process*** itself (VHDL-87 does not provide a shared variable).

EXAMPLE 27 VHDL Code Showing the Usage of a Variable

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY variable_ent IS
PORT (
        input : IN std_logic_vector (1 downto 0);
        clock : IN std_logic;
        output : OUT std_logic_vector(3 downto 0)
    );
END variable_ent;

ARCHITECTURE variable_arch OF variable_ent IS
BEGIN
        PROCESS (input, clock)
        VARIABLE var : INTEGER := 0;
        BEGIN
            IF (clock = '1' AND clock'EVENT) THEN
                IF (input = "00") THEN
                    var := var + 1;
                    IF (var = 15) THEN
                        var := 0;
                    END IF;
                END IF;
                output <= CONV_STD_LOGIC_VECTOR(var,4);
            END IF;
        END PROCESS;
END variable_arch;
```

The assignment of the *variable* **var** is instantaneous.

Conversion procedure to convert from *integer* to a 4-bit *std_logic_vector*

4.2 SIGNAL

Unlike a ***variable***, a ***signal*** is global to an architecture and only needs to be declared in the *architecture*. It can be used across all sequential processes. The assignment of a value to a *signal* only occurs in the next simulation tick.

If a ***signal*** is assigned a value in a ***process*** and that same signal is used farther down in the same sequential ***process***, it will not have the updated value until the next simulation tick.

EXAMPLE 28 VHDL Code Showing the Usage of a Signal

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY signal_ent IS
PORT (
        input : IN std_logic_vector (1 downto 0);
        clock : IN std_logic;
        output : OUT std_logic_vector(3 downto 0)
     );
END signal_ent;

ARCHITECTURE signal_arch OF signal_ent IS
signal var : INTEGER := 0;
BEGIN
        PROCESS (input, var, clock)
        BEGIN
             IF (clock = '1' AND clock'EVENT) THEN
                  IF (input = "00") THEN
                       var <= var + 1;
                       IF (var = 15) THEN
                            var <= 0;
                       END IF;
                  END IF;
             END IF;
        END PROCESS;
        output <= CONV_STD_LOGIC_VECTOR(var,4);
END signal_arch;
```

Value assigned to a signal only occurs on the next simulation tick.

Although the code looks the same in Examples 27 and 28, when simulated and synthesized, both will yield different results.

Example 29 is a testbench written to simulate Examples 27 and 28. Notice that Example 29 is a VHDL code with two *architecture* and *configuration* declarations. The *architecture* **var_tb_arch** and *configuration* **config_var_tb** allow simulation of *entity* **variable_ent** (Example 27). The *architecture* **sig_tb_arch** and *configuration* **config_sig_tb** allow simulation of *entity* **signal_ent** (Example 28).

Please take note that testbenches are non-synthesizable. Testbenches are used as a surrounding mechanism to pump stimulus into a design under test. Testbenches are used mainly for simulation.

EXAMPLE 29 Testbench for Simulation of Signal and Variable Usage

A testbench is a wrapper around the design that it simulates. Therefore, a testbench usually does not have input and output ports.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY var_sig_tb_ent IS
END var_sig_tb_ent;


ARCHITECTURE sig_tb_arch OF var_sig_tb_ent IS

COMPONENT signal_ent
PORT (
        input : IN std_logic_vector (1 downto 0);
        clock : IN std_logic;
        output : OUT std_logic_vector(3 downto 0)
     );
END COMPONENT;

SIGNAL input : std_logic_vector (1 downto 0);
SIGNAL clock : std_logic := '0';
SIGNAL output : std_logic_vector (3 downto 0);

BEGIN

        DUT_signal : signal_ent
                  PORT MAP (input, clock, output);

        clock <= NOT clock AFTER 25 ns;

        PROCESS
        BEGIN
             input <= "00";
             wait for 1000 ns;
        END PROCESS;
END sig_tb_arch;

ARCHITECTURE var_tb_arch OF var_sig_tb_ent IS

COMPONENT variable_ent
PORT (
        input : IN std_logic_vector (1 downto 0);
        clock : IN std_logic;
        output : OUT std_logic_vector(3 downto 0)
     );
END COMPONENT;

SIGNAL input : std_logic_vector (1 downto 0);
SIGNAL clock : std_logic := '0';
SIGNAL output : std_logic_vector (3 downto 0);
```

```
BEGIN

        DUT_variable : variable_ent
                  PORT MAP (input, clock, output);

        clock <= NOT clock AFTER 25 ns;

        PROCESS
        BEGIN
             input <= "00";
             wait for 1000 ns;
        END PROCESS;
END var_tb_arch;

CONFIGURATION config_sig_tb OF var_sig_tb_ent IS
FOR sig_tb_arch
        FOR ALL: signal_ent
             USE ENTITY WORK.signal_ent(signal_arch);
        END FOR;
END FOR;
END config_sig_tb;

CONFIGURATION config_var_tb OF var_sig_tb_ent IS
FOR var_tb_arch
        FOR ALL: variable_ent
             USE ENTITY WORK.variable_ent(variable_arch);
        END FOR;
END FOR;
END config_var_tb;
```

Figure 25 shows the simulation results of *entity* **`variable_ent`**. At every rising edge of **`CLOCK`**, output of **`variable_ent`** increments by one. However, note that when the output reaches a value of hexadecimal E (decimal equivalent of 15), the following output rolls over to 0.

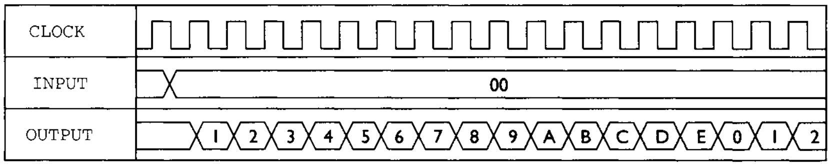


FIGURE 25 Timing Diagram Showing Simulation Result of Example 27 (Variable).

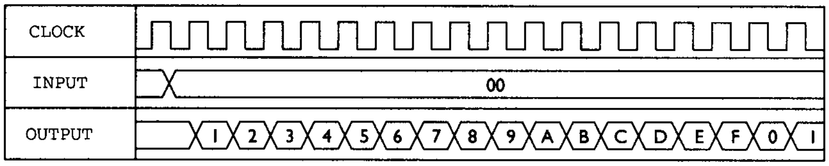


FIGURE 26 Timing Diagram Showing Simulation Results of Example 28 (Signal).

Figure 26 shows the simulation results of *entity* **signal_ent**. At every rising edge of **CLOCK**, output of **signal_ent** increments by one. However, when the output reaches a value of hexadecimal E (decimal equivalent of 15), the following output is hexadecimal F (decimal equivalent of 16) and not 0.

Entity **signal_ent** does not roll over to 0 when it reaches hexadecimal E because the assignment of value 0 to the *signal* **var** occurs only in the next simulation tick.

```
IF (clock = '1' AND clock'EVENT) THEN
                IF (input = "00") THEN
                    var <= var + 1;
                    IF (var = 15) THEN
                        var <= 0;
                    END IF;
                END IF;
            END IF;
```

However, in the next simulation tick the event **(clock = '1' AND clock'EVENT)** is no longer true. Hence the assignment of **var** to the value of 0 occurs only at the next rising edge of **CLOCK**. This results in the output of **signal_ent** incrementing to hexadecimal F before the roll over back to 0 occurs.

4.3 WHEN TO USE SIGNAL AND WHEN TO USE VARIABLE

As mentioned earlier, assignment to **variable** occurs instantaneously and assignment to **signal** only occurs on the next simulation tick. Therefore a good rule of thumb to determine the suitable conditions to use **signal/variable** would be to ask yourself, "Will I be using this assigned value in this same simulation tick?" If the answer is yes, obviously you need to use a **variable**; otherwise you can use a **signal**.

```
PROCESS (........)
BEGIN
    ...........
    var := 5;
    ............
    ............
    IF (var = 5) THEN
    ............
    ............
END PROCESS;
```

Assignment used in the same simulation tick in the sequential statement. $\longrightarrow$

It is also important for the reader to remember that usage of ***variable*** is local only to a ***PROCESS*** (VHDL 87 does not allow the usage of a shared variable). If an assigned value is to be shared among different ***PROCESS***, a ***signal*** is more appropriate.

```
PROCESS (........)
BEGIN
     ...........
     sig <= 5;
     ............
     ............
END PROCESS;
PROCESS (........)
BEGIN
     .............
     IF (sig = 5) THEN
     ............
     ............
END PROCESS;
```

Assignment is shared among different ***PROCESS***. Usage of ***signal*** is more appropriate.

However this does not mean that ***signal*** can only be used when an assigned value is to be shared among different ***PROCESS***. It can also be used in a ***PROCESS*** if the assigned value is only needed on the next simulation tick.

```
PROCESS (........)
BEGIN
     ...........
     sig <= 5;
     ............
     ............
END PROCESS;
```

The assigned value of ***sig*** occurs only on the next simulation tick.

4.4 USAGE OF LOOPBACK SIGNAL

Use of *signal* in a VHDL file, whether *structural, behavioral,* or *descriptive,* is simply unavoidable. Signals are normally used to connect between pins of different components or to interconnect modules with submodules. Another form for usage of *signal* is loopback.

Figure 27 shows a circuit using a loopback signal whereby the signal output is looked back internally.

The problem with using loopback signals is that output ports cannot be looped back. In VHDL, if a port is declared to be an output port, that port cannot be used internally in the design. It cannot be looped back into the design. If loopback is required, the port must be declared a ***BUFFER*** and not an ***OUT*** port.

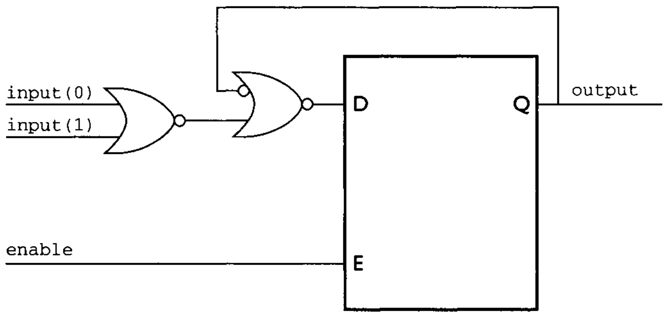


FIGURE 27 Diagram Showing a Loopback Signal.

EXAMPLE 30 VHDL Code Showing the Declaration of a BUFFER Port and an OUT Port

```
LIBRARY IEEE;
USE IEEE.std_logic-1164.ALL;

ENTITY BUFFER_OUT_ENT IS
PORT (
      input : IN std_logic_vector (7 downto 0);
      output1 : OUT std_logic;
      output2 : BUFFER std_logic
     );
END BUFFER_OUT_ENT;

ARCHITECTURE BUFFER_OUT_ARCH OF BUFFER_OUT_ENT IS
SIGNAL internal_signal : std_logic;
BEGIN
  — VHDL code here
  — .............
          internal_signal <= output2;
  — more VHDL code
  — ...............
END BUFFER_OUT_ARCH;
```

Port ***output2*** is being looped back.

Port declaration as ***BUFFER*** shown in Example 30 is not encouraged. When the module containing ***BUFFER*** declaration is put together in full chip with other modules that do not use ***BUFFER*** declarations, errors will occur during compilation. Any other module's port that is connected to ***output2*** must also be declared as a ***BUFFER***.

In real-life designs, a port is commonly declared only as ***IN*** for input ports, ***OUT*** for output ports, and ***INOUT*** for bi-directional ports. Use of ***BUFFER*** declaration is seldom encountered due to the fact that connectivity between different design modules normally uses only ***IN, OUT*** and ***INOUT*** declarations.

In general, if a port is declared as an ***OUT*** port and the design requires the ***OUT*** port to be looped back internally, a loopback ***signal*** is used.

EXAMPLE 31 Example Showing a Loopback Signal

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY loopback_ent IS
PORT (
       enable : IN std_logic;
       input : IN std_logic_vector (1 downto 0);
       clock : IN std_logic;
       output : OUT std_logic
     );
END loopback_ent;

ARCHITECTURE loopback_arch OF loopback_ent IS
SIGNAL loopback : std_logic;
BEGIN
       PROCESS (enable, input, loopback)
       BEGIN
            IF (enable = '1') THEN
                 IF (input = "00") THEN
                      loopback <= '0';
                 END IF;
            END IF;
       END PROCESS;
       output <= loopback;
END loopback_arch;
```

Declaration of a `signal` for loop back

Connecting **`loopback`** to output port.

The synthesis result of Example 31 is the circuit that was shown in Fig. 27. Did you notice how the signal was looped back internally in the design?

This Page Intentionally Left Blank

5

EXAMPLES OF COMPLEX SYNTHESIZABLE CODE

Chapter 3 provided many examples of synthesizable VHDL code for some very basic logic components. Chapter 4 gave explanations as to how variables and signals are used in VHDL and how logic components are inferred. However, synthesizable VHDL for complex components are much more complicated than the examples shown in these chapters.

In Chapter 5, there are four examples (shifter, counter, memory module, and car traffic controller) of synthesizable VHDL for complex logic. For each of these examples, testbenches and simulation results are shown to enable the reader to better understand how VHDL code is mapped into hardware logic.

5.1 SHIFTER

There are three different kinds of shifters.

- **Left shifter**
  Shift values to the left while the right-most bit is replaced by a zero.
- **Right shifter**
  Shift values to the right while the left-most bit is replaced by a zero.
- **Barrel shifter**
  Shift values in a loop.
  For barrel shift right, the most significant bit is replaced with the least significant bit.
  For barrel shift left, the least significant bit is replaced with the most significant bit.

For a data value of '1001' as an input to a shifter, the patterns seen on the output of the shifter during every rising edge of clock for each individual shift are shown in Table 15.

TABLE 15 Table Showing the Shifting of Data for Different Modes of Shifting

Clock	0	1	2	3	4	5	6	7
Shift Left	1001	0010	0100	1000	0000	0000	0000	0000
Shift Right	1001	0100	0010	0001	0000	0000	0000	0000
Barrel Shift Right	1001	1100	0110	0011	1001	1100	0110	0011
Barrel Shift Left	1001	0011	0110	1100	1001	0011	0110	1100

From Table 15, the least significant bit is tagged with a zero when shifting left. Similarly when shifting right, the most significant bit is tagged zero.

Table 16 shows the interface specification of a shifter module.

TABLE 16 Description of Pins and Descriptions for Shifter

Pins	Direction	Description
DATA	Input	Data to be input to the shifter
LOAD	Input	Asserted low to load in the input data
ENABLE	Input	Asserted low to enable the shift function
CLOCK	Input	***CLOCK*** input, shifting occurs when ENABLE is low and clock is at rising edge
MODE	Input	Mode to determine for left shift, right shift or barrel shift
		MODE(1:0) = "00" - shift left
		MODE(1:0) = "01" - shift right
		MODE(1:0) = "10" - barrel shift right
		MODE(1:0) = "11" - barrel shift left
OUTPUT	Output	Output data after shift

Figure 28 shows a top-level block diagram with the input and output pins of the design.

Before we proceed with the coding of the shifter, a flow chart (Fig. 29) is created to facilitate coding. With a flow chart, it is much easier to code the design based on the tracks of the flow chart. It is highly recommended that a new designer draw a state machine or flow chart to represent the functionality of a design before attempting to code it. If the designer is not comfortable with drawing flow charts prior to coding, the designer can always use "bubble" diagrams or even "pseudocode." Whichever approach is used, the end result is the same. Flow charts or bubble diagrams and pseudocode are just a means to guide the designer to code a given design.

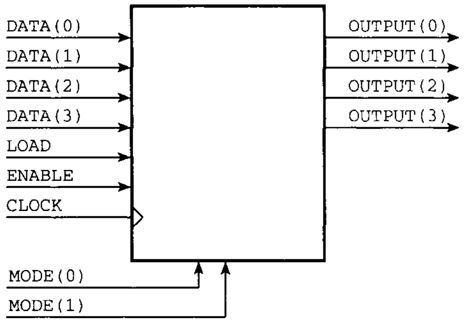


FIGURE 28 Pin Diagram for 4-Bit Shifter Design.

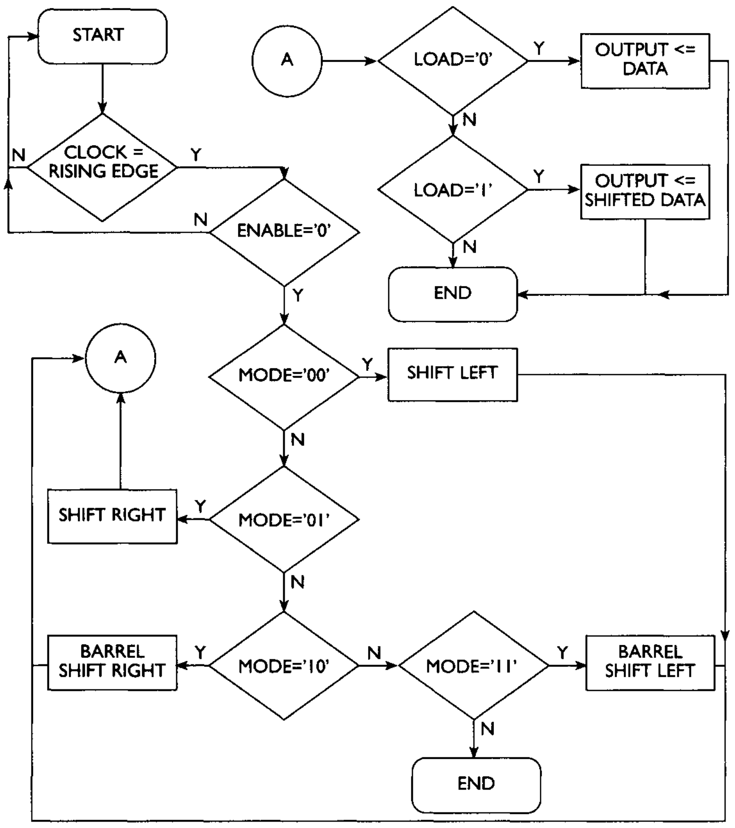


FIGURE 29 Diagram Showing Flow Chart for Shifter Design.

EXAMPLE 32 Synthesizable VHDL Code for a 4-Bit Shifter

Shift mode assignment

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;


ENTITY shifter_ent IS
PORT (
     data : IN std_logic_vector( 3 downto 0);
     load : IN std_logic;
     enable : IN std_logic;
     clock : IN std_logic;
     mode : IN std_logic_vector (1 downto 0);
     output : OUT std_logic_vector (3 downto 0)
     );
END shifter_ent;

ARCHITECTURE shifter_arch OF shifter_ent IS
SIGNAL internal_output : std_logic_vector (3 downto 0);
BEGIN
        PROCESS (clock)
        BEGIN
        IF (clock = '1' AND clock'EVENT) THEN
                  IF (enable = '0') THEN
                       IF (load = '0') THEN
                            internal_output <= data;
                       ELSE
                            IF (mode = "00") THEN
                                 -- shift left
                                 internal_output <=
                                 internal_output(2 downto 0) &
                                 '0';
                            ELSIF (mode = "01") THEN
                                 -- shift right
                                 internal_output <= '0' &
                                 internal_output (3 downto 1);
                            ELSIF (mode = "10") THEN
                                 -- shift barrel right
                                 internal_output <=
                                 internal_output(0) &
                                 internal_output (3 downto 1);
                            ELSIF (mode = "11") THEN
                                 -- shift barrel left
                                 internal_output <=
                                 internal_output(2 downto 0) &
                                 internal_output(3);
```

```
                        ELSE
                            internal_output <= "0000";
                        END IF;
                    END IF;
                ELSE
                    internal_output <= "0000";
                END IF;
            END IF;
        END PROCESS;

        output <= internal_output;

END shifter_arch;
```

Default assignment when none of the shift modes match.

Default assignment when shifter not enabled.

The code written in Example 32 is a synthesizable code for a 4-bit shifter. Enlarging the shifter to accommodate more bits can be done by enlarging the ***DATA*** input and ***OUTPUT*** ports.

A testbench is written to pump the stimulus into the shifter design to generate the output pattern. This output pattern is viewed to ensure the code written for the shifter design is functionally correct. The testbench also consists of a procedure ***check_data***, which would assert messages on the simulator window when the output values are not correct.

EXAMPLE 33 Testbench for the 4-Bit Shifter Design

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

-- PACKAGE for shifter

```
PACKAGE shifter_package IS

        CONSTANT CYCLE : TIME := 50 ns;
        SIGNAL sig_data : std_logic_vector (3 downto 0);
        SIGNAL sig_mode : std_logic_vector (1 downto 0);
        SIGNAL sig_output_before_shift : std_logic_vector (3 downto
                                                                0);
        SIGNAL sig_output_after_shift : std_logic_vector (3 downto
                                                                0);

        PROCEDURE load_data (
            SIGNAL LOAD : OUT std_logic;
            SIGNAL DATA : OUT std_logic_vector( 3 downto 0) ;
            SIGNAL MODE : OUT std_logic_vector (1 downto 0));
```

Declaration of procedure for checking ouput

```
    PROCEDURE check_data (
        SIGNAL MODE : OUT std_logic_vector (1 downto 0));

END shifter_package;

PACKAGE BODY shifter_package IS

    PROCEDURE check_data (
        SIGNAL MODE : OUT std_logic_vector (1 downto 0)) IS
    BEGIN
        MODE <= sig_mode;
        IF (sig_mode = "00") THEN
            -- shift left
            IF (sig_output_after_shift /=
                sig_output_before_shift(2 downto 0) & '0')THEN
                ASSERT FALSE
                REPORT "Error detected in shift left."
                SEVERITY WARNING;
            END IF;
        ELSIF (sig_mode = "01") THEN
            -- shift right
            IF (sig_output_after_shift /= '0' &
                sig_output_before_shift(3 downto 1)) THEN
                ASSERT FALSE
                REPORT "Error detected in shift right."
                SEVERITY WARNING;
            END IF;
        ELSIF (sig_mode = "10") THEN
            -- barrel shift right
            IF (sig_output_after_shift /=
                (sig_output_before_shift(0) &
                sig_output_before_shift (3 downto 1))) THEN
                ASSERT FALSE
                REPORT "Error detected in shift barrel
                    right."
                SEVERITY WARNING;
            END IF;
        ELSIF (sig_mode = "11") THEN
            -- barrel shift left
            IF (sig_output_after_shift /=
                (sig_output_before_shift(2 downto 0) &
                sig_output_before_shift(3))) THEN
                ASSERT FALSE
                REPORT "Error detected in shift barrel
                    left."
                SEVERITY WARNING;
                END IF;
            END IF;
        END check_data;

        PROCEDURE load_data (
            SIGNAL LOAD : OUT std_logic;
            SIGNAL DATA : OUT std_logic_vector (3 downto 0) ;
            SIGNAL MODE : OUT std_logic_vector (1 downto 0)) IS
        BEGIN
            DATA <= sig_data;
            MODE <= sig_mode;
            LOAD <= '1';
            wait for CYCLE;
            LOAD <= '0';
            wait for CYCLE;
            LOAD <= '1';
            wait for CYCLE;
        END load_data;
END shifter_package;
```

Declaration of procedure to load data into shifter.

-- Testbench for shifter

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.shifter_package.ALL;

ENTITY shifter_tb_ent IS
END shifter_tb_ent;

ARCHITECTURE shifter_tb_arch OF shifter_tb_ent IS

COMPONENT shifter_ent
PORT (
     DATA : IN std_logic_vector( 3 downto 0);
     LOAD : IN std_logic;
     ENABLE : IN std_logic;
     CLOCK : IN std_logic;
     MODE : IN std_logic_vector (1 downto 0);
     OUTPUT : OUT std_logic_vector (3 downto 0)
     );
END COMPONENT;

SIGNAL DATA : std_logic_vector (3 downto 0);
SIGNAL LOAD : std_logic;
SIGNAL ENABLE : std_logic;
SIGNAL CLOCK : std_logic := '0';
SIGNAL MODE : std_logic_vector (1 downto 0);
SIGNAL OUTPUT : std_logic_vector (3 downto 0);

BEGIN

DUT: shifter_ent
            port map( DATA, LOAD, ENABLE, CLOCK, MODE, OUTPUT);

CLOCK <= NOT CLOCK AFTER CYCLE/2;

PROCESS
BEGIN

-- enable the design
ENABLE <= '0';

-- load data "1010" into design
sig_data <= "1010";
sig_mode <= "00";
wait for CYCLE;
load_data (LOAD, DATA, MODE);

-- shift left
for i in 0 to 6 loop
        sig_output_before_shift <= OUTPUT;
        wait for CYCLE;
        sig_output_after_shift <= OUTPUT;
        wait for CYCLE;
        check_data (MODE);
end loop;

-- load data "1010" into design
sig_data <= "1010";
sig_mode <= "01";
wait for CYCLE;
load_data (LOAD, DATA, MODE);

-- shift right
for i in 0 to 6 loop
        sig_output_before_shift <= OUTPUT;
        wait for CYCLE;
        sig_output_after_shift <= OUTPUT;
        wait for CYCLE;
        check_data (MODE);
end loop;
```

```
-- load data "1001" into design
sig_data <= "1001";
sig_mode <= "10";
wait for CYCLE;
load_data (LOAD, DATA, MODE);

-- shift barrel right --
for i in 0 to 6 loop
       sig_output_before_shift <= OUTPUT;
       wait for CYCLE;
       sig_output_after_shift <= OUTPUT;
       wait for CYCLE;
       check_data (MODE);
end loop;

-- load data "1001" into design
sig_data <= "1001";
sig_mode <= "11";
wait for CYCLE;
load_data (LOAD, DATA, MODE);

-- shift barrel left --
for i in 0 to 6 loop
       sig_output_before_shift <= OUTPUT;
       wait for CYCLE;
       sig_output_after_shift <= OUTPUT;
       wait for CYCLE;
       check_data (MODE);
end loop;

END PROCESS;
END shifter_tb_arch;

CONFIGURATION shifter_tb_config OF shifter_tb_ent IS
  FOR shifter_tb_arch
    FOR ALL : shifter_ent
      USE ENTITY WORK.shifter_ent(shifter_arch);
    END FOR;
  END FOR;
END shifter_tb_config;
```

From the testbench code for the shifter design, input stimuli of '1' and '0' are pumped into the design to simulate different conditions as a way to check the design for functionality of different modes for shifting left, shifting right, and barrel shifting.

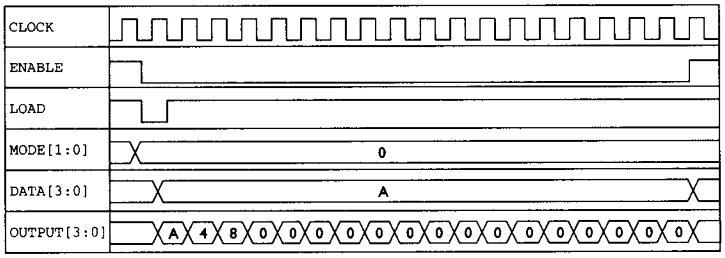


FIGURE 30 Timing Diagram for Shift Left.

Figure 30 shows the timing diagram for shifting left. When signal ***LOAD*** is pulsed '0', ***DATA*** is sent to ***OUTPUT***. and when ***LOAD*** is back to logical '1', at every rising edge of ***CLOCK*** shifts the data left by one bit. For every bit shifted left, the least significant bit is replaced with a '0'.

Figure 31 shows the timing diagram for shift right. ***DATA*** is loaded into the shifter when ***LOAD*** is pulsed '0'. When ***LOAD*** is pulsed back to a '1', at every rising edge of ***CLOCK***, ***DATA*** is shifted right. When the shifting occurs, the most significant bit is replaced with a '0'.

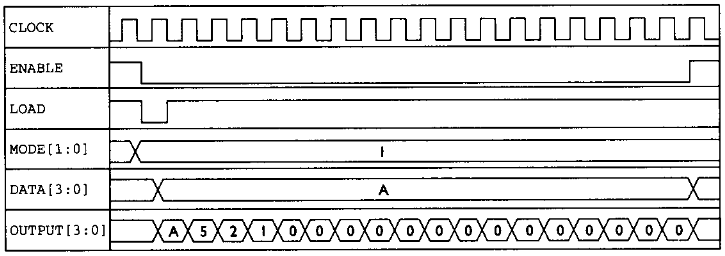


FIGURE 31 Timing Diagram for Shift Right.

Figure 32 shows the timing diagram for barrel shift right. After loading in ***DATA***, at every rising edge of ***CLOCK***, the most significant bit is replaced with the least significant bit.

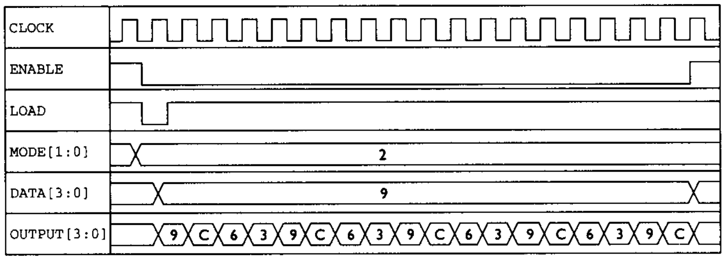


FIGURE 32 Timing Diagram for Barrel Shift Right.

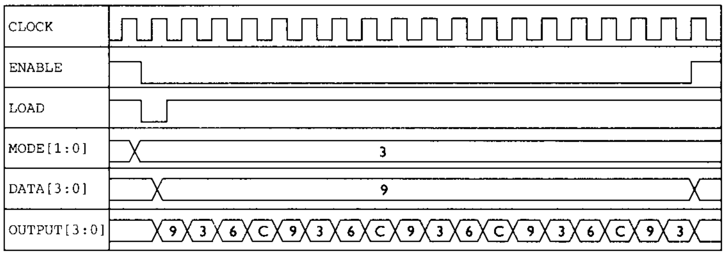


FIGURE 33 Timing Diagram for Barrel Shift Left.

Figure 33 shows the timing diagram for a barrel shift left. After loading in ***DATA***, at every rising edge of ***CLOCK***, the least significant bit is replaced with the most significant bit.

To synthesize this shifter design using Synopsys's Design Compiler, a set of design constraints first must be declared on the design. This set of constraints is to include input delays, output delays, and clock period information.

The synthesis script and synthesis results for the shifter are not shown here as timing issues topics during synthesis have yet to be discussed. Appendix B shows the full-scale synthesis of this shifter design, which includes design constraints and synthesis tweaks to obtain optimal synthesis results using Synopsys's Design Compiler.

5.2 COUNTER

A counter can be set to count up or count down. By defining a 4-bit counter with inputs ***LOAD***, ***ENABLE***, ***DATA***, ***CLOCK***, ***MODE*** and output pin ***OUTPUT***, a pin description table (Table 17) is created for the interface pins.

TABLE 17 Pin Description For Counter Design

Pin	Input/Output	Description
DATA	Input	Data input pins
LOAD	Input	When asserted low, it will drive the ***OUTPUT*** pins with ***DATA*** input values
ENABLE	Input	Asserted low to enable the design
CLOCK	Input	Count occurs on rising edge of ***CLOCK*** after ***DATA*** have been loaded in by pulsing ***LOAD*** low.
MODE	Input	***MODE*** = 0 for count up and ***MODE*** = 1 for count down
OUTPUT	Output	Output pins for counter design

The counter will roll over to '0000' when it has reached the maximum count of '1111' for count up mode. Similarly in countdown mode, the counter will roll over to '1111' when it has reached the minimum count value of '0000'.

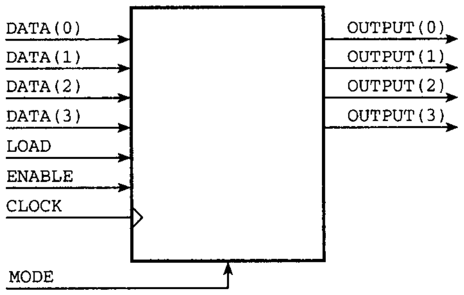


FIGURE 34 Pin Diagram for a 4-Bit Counter Design.

Figure 34 shows the block diagram for the counter with the interface pins.

EXAMPLE 34 Synthesizable VHDL Code for a 4-Bit Counter Design

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY counter_ent IS
PORT (
        data : IN std_logic_vector (3 downto 0);
        load : IN std_logic;
        enable : IN std_logic;
        clock : IN std_logic;
        mode : IN std_logic;
        output : OUT std_logic_vector (3 downto 0)
     );
END counter_ent;

ARCHITECTURE counter_arch OF counter_ent IS
SIGNAL internal_output : std_logic_vector (3 downto 0);
BEGIN
        PROCESS (clock)
        BEGIN
             IF (clock = '1' AND clock'EVENT) THEN
                  IF (enable = '0') THEN
                       IF (load = '0') THEN
                            internal_output <= data;
                       ELSE
                            IF (mode = '0') THEN
                                 internal_output <=
                                 signed(internal_output) +
                                 '1';
```

Counting up for count up mode.

```
                           ELSIF (mode = '1') THEN
                                internal_output <=
                                signed(internal_output) -
                                '1';
                           ELSE
                                internal_output <= "0000";
                           END IF;
                      END IF;
                 ELSE
                      internal_output <= "0000";
                 END IF;
            END IF;
       END PROCESS;

       output <= internal_output;

END counter_arch;
```

Counting down for countdown mode.

Default value of "0000".

Example 34 uses the symbol '+' to count up. When the synthesis tool sees this symbol, it will infer an adder. Some synthesis tools with incrementers in the precompiled library will infer an incrementer. Similarly, use of the symbol '-' for countdown will infer a subtractor (or decrementer).

If the synthesis tool does not have an adder/subtractor/incrementer/decrementer, it will use logic gates to construct the necessary logic required for the design.

Example 35 is a testbench written to exercise the design to ensure correct functionality of the design. The testbench contains a procedure **check_data** that would assert a message on the simulation window if an error is detected on the output of the counter with the expected output.

EXAMPLE 35 Testbench for 4-Bit Counter Design

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

-- PACKAGE for counter

PACKAGE counter_package IS

       CONSTANT CYCLE : TIME := 50 ns;
       SIGNAL sig_data : std_logic_vector (3 downto 0);
       SIGNAL sig_mode : std_logic;
       SIGNAL sig_output_before_add : std_logic_vector(3 downto
                                                          0);
```

```
          SIGNAL sig_output_after_add : std_logic_vector (3 downto
                                                               0);

          PROCEDURE check_data (
               SIGNAL MODE : OUT std_logic);

          PROCEDURE load_data (
               signal LOAD : out std_logic;
               signal DATA : out std_logic_vector( 3 downto 0) ;
               signal MODE : out std_logic);

END counter_package;

PACKAGE BODY counter_package IS

          PROCEDURE check_data (
               SIGNAL MODE : OUT std_logic) IS
          BEGIN
               MODE <= sig_mode;
               IF (sig_mode = '1') THEN
                    -- count up
                    IF (sig_output_after_add /=
signed(sig_output_before_add) + '1') THEN
          ASSERT FALSE
                         REPORT "Error in counting up"
                         SEVERITY WARNING;
                    END IF;
               ELSIF (sig_mode = '0') THEN
                    -- count down
                    IF (sig_output_after_add /=
signed(sig_output_before_add) - '1') THEN
                         ASSERT FALSE
                         REPORT "Error in counting down"
                         SEVERITY WARNING;
                    END IF;
               END IF;
          END check_data;

          PROCEDURE load_data (
               signal LOAD : out std_logic;
               signal DATA : out std_logic_vector( 3 downto 0) ;
               signal MODE : out std_logic) IS
          BEGIN
               DATA <= sig_data;
               MODE <= sig_mode;
               LOAD <= '1';
```

```
            wait for CYCLE;
            LOAD <= '0';
            wait for CYCLE;
            LOAD <= '1';
            wait for CYCLE;
        END load_data;
END counter_package;
```

-- Testbench for counter

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.counter_package.ALL;

ENTITY counter_tb_ent IS
END counter_tb_ent;

ARCHITECTURE counter_tb_arch OF counter_tb_ent IS
COMPONENT counter_ent
PORT (
     DATA : IN std_logic_vector( 3 downto 0);
     LOAD : IN std_logic;
     ENABLE : IN std_logic;
     CLOCK : IN std_logic;
     MODE : IN std_logic;
     OUTPUT : OUT std_logic_vector (3 downto 0)
     );
END COMPONENT;

SIGNAL DATA : std_logic_vector (3 downto 0);
SIGNAL LOAD : std_logic;
SIGNAL ENABLE : std_logic;
SIGNAL CLOCK : std_logic := '0';
SIGNAL MODE : std_logic;
SIGNAL OUTPUT : std_logic_vector (3 downto 0);

BEGIN

DUT: counter_ent
            port map( DATA, LOAD, ENABLE, CLOCK, MODE, OUTPUT);

CLOCK <= NOT CLOCK AFTER CYCLE/2;

PROCESS
BEGIN
```

```
-- enable the design
ENABLE <= '0';

-- load data "1010" into design
sig_data <= "1010";
sig_mode <= '0';
wait for CYCLE;
load_data (LOAD, DATA, MODE);

-- count up
for i in 0 to 6 loop
       sig_output_before_add <= OUTPUT;
       wait for CYCLE;
       sig_output_after_add <= OUTPUT;
       wait for CYCLE;
       check_data (MODE);
end loop;

-- load data "1010" into design
sig_data <= "1010";
sig_mode <= '1';
wait for CYCLE;
load_data (LOAD, DATA, MODE);

-- count down
for i in 0 to 6 loop
       sig_output_before_add <= OUTPUT;
       wait for CYCLE;
       sig_output_after_add <= OUTPUT;
       wait for CYCLE;
       check_data (MODE);
end loop;

END PROCESS;
END counter_tb_arch;

CONFIGURATION counter_tb_config OF counter_tb_ent IS
  FOR counter_tb_arch
    FOR ALL : counter_ent
      USE ENTITY WORK.counter_ent(counter_arch);
    END FOR;
  END FOR;
END counter_tb_config;
```

Figures 35 and 36 show the timing diagram for counting up and counting down.

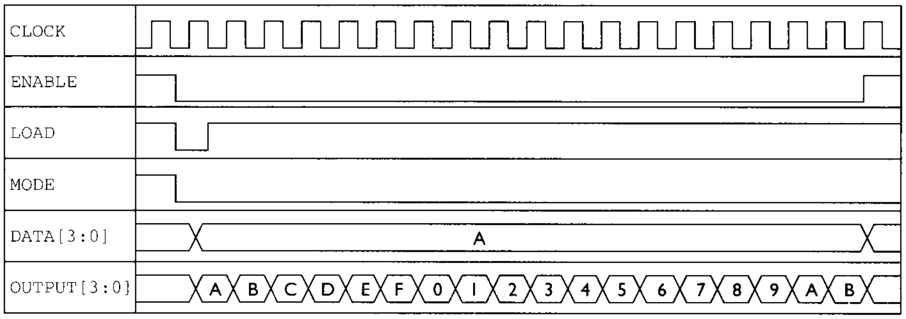


FIGURE 35 Timing Diagram Showing Count Up for Counter Design.

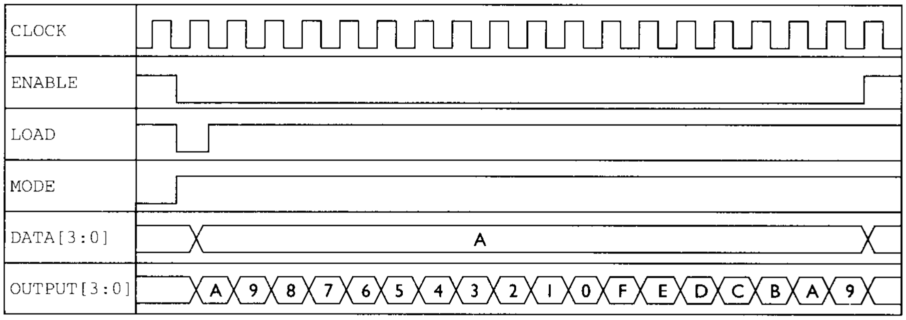


FIGURE 36 Timing Diagram Showing Countdown for Counter Design.

To synthesize this counter design using Synopsys's Design Compiler, first a set of design constraints must be declared on the design. This set of constraints includes input delays, output delays, and clock period information.

The synthesis script and synthesis result for the counter are not shown here as topics on timing issues during synthesis have yet to be discussed. Appendix C shows the full-scale synthesis of this counter design, which includes design constraints and synthesis tweaks to obtain optimal synthesis results using Synopsys's Design Compiler.

5.3 MEMORY MODULE

In Chapter 3.14, the synthesizable code for a 1-bit memory cell is shown. It is built out of a multiplexer and a flip-flop. In daily use, one memory cell does not really serve much purpose.

Daily encountered designs normally would require several tens of memory bits. To obtain these N bits, the example in Chapter 3.14 can be expanded into an array. However, in daily designs seldom will a designer synthesize a memory module as it

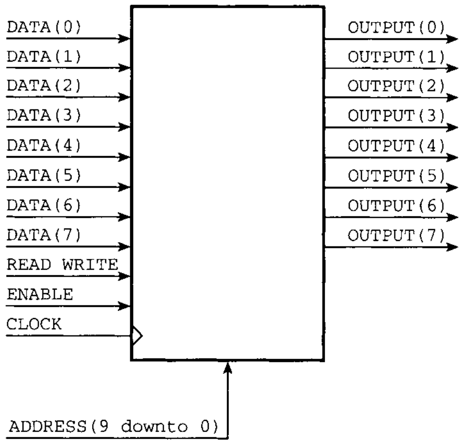


FIGURE 37 Pin Diagram for Memory Module.

would take up a lot of silicon area. It would make more sense for a designer to use a hand-drawn transistor level memory module. For the sake of discussion, however, this chapter will show the reader how to code an N-bit memory module that is synthesizable.

The VHDL coding style for synthesizing a memory module is pretty much the same as that for synthesizing a 1-bit memory cell.

Assuming the memory module of Fig. 37 is to be 1 Kbyte in size with each address location 8-bits wide, 10 address pins and 7 data pins are required. See Table 18 for pin description.

TABLE 18 Description of Pins for Memory Module Design

Pin	Input/Output	Description
DATA	Input	Input pins where data is written into the memory module
READ_WRITE	Input	When asserted low, it is a for a write cycle and when asserted high, it is for a read cycle
ENABLE	Input	Asserted low to enable the design
CLOCK	Input	A write cycle or read cycle is initiated at every rising edge of ***CLOCK*** when ***ENABLE*** is asserted low
ADDRESS	Input	Address inputs to the memory module
OUTPUT	Output	Output pins on which the data being read is driven

Again in this design approach, a flow chart (see Fig. 38) is used to represent design functionality. If the designer is not comfortable with flow charts, he/she may use bubble diagrams or pseudocode. Whichever is used, the end result is the same. Flow charts, bubble diagrams or pseudocode is only meant to guide the designer to code the VHDL.

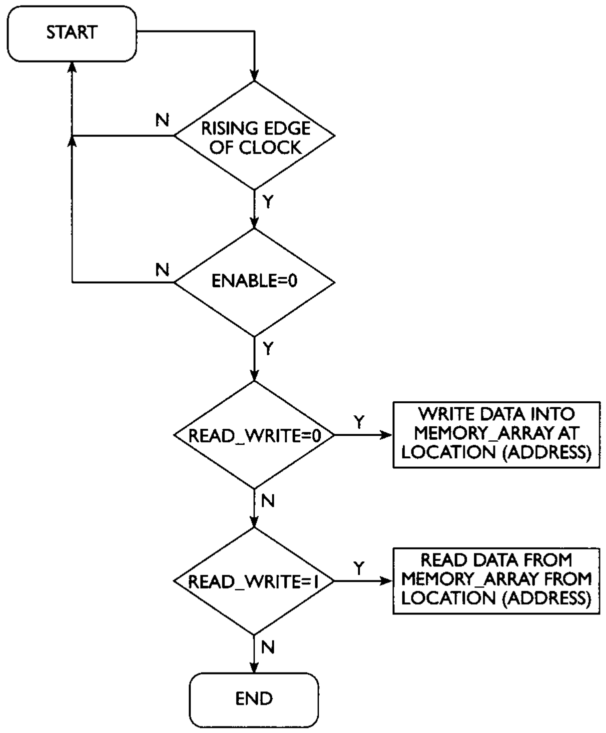

START
RISING EDGE OF CLOCK
N
Y
ENABLE=0
N
Y
READ_WRITE=0
Y
WRITE DATA INTO MEMORY_ARRAY AT LOCATION (ADDRESS)
N
READ_WRITE=1
Y
READ DATA FROM MEMORY_ARRAY FROM LOCATION (ADDRESS)
N
END

FIGURE 38 Flow Chart for Reading/Writing for a Memory Module.

EXAMPLE 36 Example of Synthesizable Code for a 1-Kbyte Memory Module Design

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY memmod_ent IS
PORT (
       DATA : IN std_logic_vector (7 downto 0);
       READ_WRITE : IN std_logic;
       ENABLE : IN std_logic;
       CLOCK : IN std_logic;
       ADDRESS : IN std_logic_vector (9 downto 0);
       OUTPUT : OUT std_logic_vector (7 downto 0)
     );
END memmod_ent;
```

Declaration of type **memory_array_type** as array size of 1 Kbyte with each array sector 8-bits wide.

Detection of rising edge of clock

Writing of data to address location **temp_var**

Reading of data from address location **temp_var**

```
ARCHITECTURE memmod_arch OF memmod_ent IS
TYPE memory_array_type IS array (0 to (2**(10) - 1)) OF
std_logic_vector(7 downto 0);
SIGNAL memory_array : memory_array_type;
BEGIN
        PROCESS (CLOCK, ENABLE, READ_WRITE)
        VARIABLE temp_var : INTEGER;
        BEGIN
             IF (CLOCK = '1' AND CLOCK'EVENT) THEN
                  IF (ENABLE = '0') THEN
                       IF (READ_WRITE = '0') THEN
                            temp_var := CONV_INTEGER(address);
                            memory_array(temp_var) <= data;
                       ELSIF (READ_WRITE = '1') THEN
                            temp_var := CONV_INTEGER(address);
                            OUTPUT <= memory_array(temp_var);
                       END IF;
                  END IF;
             END IF;
        END PROCESS;
END memmod_arch;
```

Did you notice any similarities between Example 36 and Example 24 (1-bit memory cell)? Did you notice the similarities in the coding style for both examples?

From both Examples 24 and 36, the rising edge of **CLOCK** is detected to perform a memory read or memory write sequence. This rising edge of **CLOCK** is transformed into a flip-flop in hardware logic.

In Example 36, a **TYPE** declaration is made on **memory_array_type**. This declaration creates a new TYPE called **memory_array_type** that represents an array 1 Kbyte in size with each sector of the array 8-bits wide.

The *signal* **memory_array** is declared using the *TYPE* **memory_array_type**. This means that the *signal* **memory_array** is an array 1 KByte in size with each array sector 8-bits wide.

The ***ADDRESS*** that is 10-bits wide is converted from **std_logic_vector** into **integer** and the result of this conversion is assigned to the *variable* **temp_var**. The value of **temp_var** decides from which of the sectors within the memory array data should be read or into which sector data should be written.

Example 36 is a very simple design for designers that uses VHDL code instead of conventional schematic capture. By only using several lines of code, a 1-Kbyte-memory array is synthesized. If a designer is using schematic capture, it would take a lot more than just a few lines of code.

A drawback to synthesis of a memory module is that the size of the memory module is rather large as compared to a hand-packed transistor level memory module. This is obvious because synthesis uses a multiplexer and a flip-flop to build 1 bit of memory, whereas in the conventional transistor level only 6 transistors are required.

The time needed to synthesize a memory module might be longer than would be required by many of the other examples in this book. The synthesis tool will have to generate all the necessary logic gates to build the 1-Kbyte memory module, consisting of 8192 (1024 × 8 bits) memory cells.

In real-life designs, a designer seldom synthesizes such a huge memory module. Synthesis of a memory module is often limited to only tens or hundreds of bits. One example of memory module synthesis would be for a small register file in a RISC processor.

In Example 37, a testbench is written to simulate and verify the functionality of the design.

EXAMPLE 37 VHDL Testbench for the Memory Module Design

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

-- PACKAGE for memory module

PACKAGE memmod_package IS

        CONSTANT CYCLE : TIME := 50 ns;
        SIGNAL sig_data : std_logic_vector (7 downto 0);
        SIGNAL sig_read_write : std_logic;
        SIGNAL sig_address : std_logic_vector (9 downto 0);

        PROCEDURE write_data (
              signal ADDRESS : OUT std_logic_vector (9 downto 0);
              signal DATA : OUT std_logic_vector( 7 downto 0) ;
              signal READ_WRITE : OUT std_logic);

        PROCEDURE read_data (
              signal ADDRESS : OUT std_logic_vector (9 downto 0);
              signal DATA : OUT std_logic_vector( 7 downto 0) ;
              signal READ_WRITE : OUT std_logic);

END memmod_package;

PACKAGE BODY memmod_package IS

        PROCEDURE write_data (
              signal ADDRESS : OUT std_logic_vector (9 downto 0);
              signal DATA : OUT std_logic_vector( 7 downto 0) ;
              signal READ_WRITE : OUT std_logic) IS
        BEGIN
```

```
          DATA <= sig_data;
          READ_WRITE <= sig_read_write;
          ADDRESS <= sig_address;
          wait for CYCLE;
     END write_data;

     PROCEDURE read_data (
          signal ADDRESS : OUT std_logic_vector (9 downto 0);
          signal DATA : OUT std_logic_vector( 7 downto 0) ;
          signal READ_WRITE : OUT std_logic) IS
     BEGIN
          ADDRESS <= sig_address;
          READ_WRITE <= sig_read_write;
          wait for CYCLE;
     END read_data;

END memmod_package;
```

-- Testbench for memory module

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.memmod_package.ALL;

ENTITY memmod_tb_ent IS
END memmod_tb_ent;

ARCHITECTURE memmod_tb_arch OF memmod_tb_ent IS
COMPONENT memmod_ent
PORT (
        DATA : IN std_logic_vector (7 downto 0);
        READ_WRITE : IN std_logic;
        ENABLE : IN std_logic;
        CLOCK : IN std_logic;
        ADDRESS : IN std_logic_vector (9 downto 0);
        OUTPUT : OUT std_logic_vector (7 downto 0)
);
END COMPONENT;

SIGNAL DATA : std_logic_vector (7 downto 0);
SIGNAL READ_WRITE : std_logic;
SIGNAL ENABLE : std_logic;
SIGNAL CLOCK : std_logic := '0';
SIGNAL OUTPUT : std_logic_vector (7 downto 0);
SIGNAL ADDRESS : std_logic_vector (9 downto 0);

BEGIN
```

```
DUT: memmod_ent
            port map( DATA, READ_WRITE, ENABLE, CLOCK, ADDRESS,
OUTPUT);

CLOCK <= NOT CLOCK AFTER CYCLE/2;

PROCESS
BEGIN

-- enable the design
ENABLE <= '0';

-- write data into registers
FOR i IN 0 TO 6 LOOP
        sig_read_write <= '0';
        sig_data <= "00000000" OR CONV_STD_LOGIC_VECTOR (i, 8);
        sig_address <= "0000000000" OR CONV_STD_LOGIC_VECTOR(i,10);
        wait for CYCLE;
        write_data (ADDRESS, DATA, READ_WRITE);
END LOOP;

-- read data from registers
FOR i IN 0 TO 6 LOOP
        sig_read_write <= '1';
        sig_address <= "0000000000" OR CONV_STD_LOGIC_VECTOR(i,10);
        wait for CYCLE;
        read_data (ADDRESS, DATA, READ_WRITE);
END LOOP;

END PROCESS;
END memmod_tb_arch;

CONFIGURATION memmod_tb_config OF memmod_tb_ent IS
  FOR memmod_tb_arch
    FOR ALL : memmod_ent
      USE ENTITY WORK.memmod_ent(memmod_arch);
    END FOR;
  END FOR;
END memmod_tb_config;
```

Figures 39 and 40 show the timing waveform from the simulation of the memory module using the testbench in Example 37.

The initial contents of memory array shown in Fig. 39 are invalid as they have not been initialized. When ***ENABLE*** and ***READ_WRITE*** are low, the memory module is enabled to operate in write mode. At every rising edge of ***CLOCK***, ***DATA*** is written into the memory array at a location indicated by ***ADDRESS***.

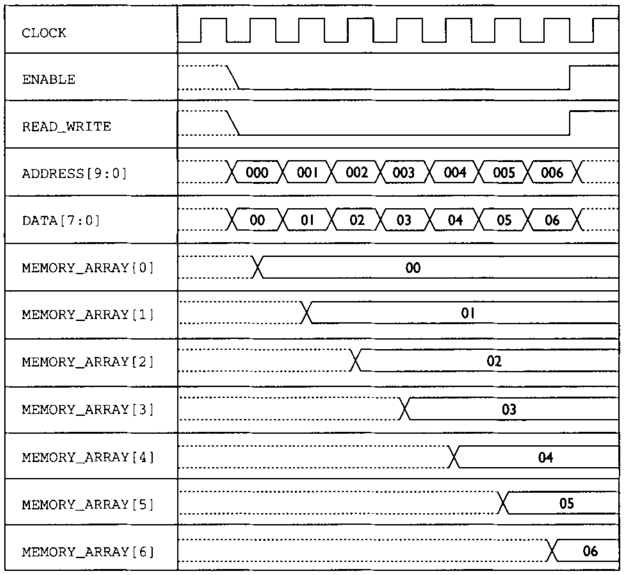


FIGURE 39 Timing Waveform for Memory Write to Memory Module Design.

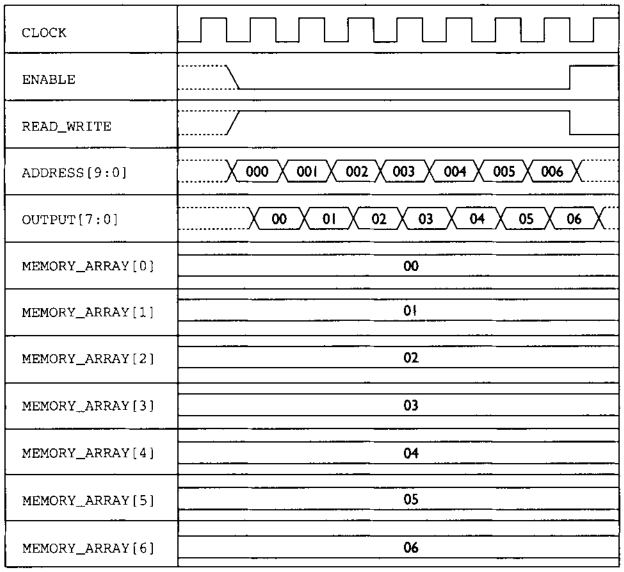


FIGURE 40 Timing Waveform for Memory Read from Memory Module Design.

From Fig. 40, ***ENABLE*** is asserted low while ***READ_WRITE*** is asserted high. This would enable the memory module in the read mode. At every rising edge of ***CLOCK***, data are read from the memory array at the location indicated by ***ADDRESS***. The data are driven out of the memory module through the output pins ***OUTPUT***.

5.4 CAR TRAFFIC CONTROLLER

State machines are very common in everyday design. They are easy to code and debug when functional errors are detected.

There are several different styles of coding a state machine. Some designers like to use different processes to represent the combinational logic and the state registers and others prefer to code the combinational logic and the state registers in the same process. No matter which style is used, the logic synthesized is similar to the same set of module-level functionality. However, a good coding style to maintain uses separate PROCESS for state registers and combinational logic. This gives the designer the flexibility to create separate levels of hierarchy in synthesis for the combinational logic and state registers.

Table 19 shows the interface pin description of a Car Traffic Controller module. The controller is built out of 4 inputs (***EVALUATE***, ***RED***, ***YELLOW*** and ***GREEN***). It is used in a car to control the automatic acceleration, deceleration, and brake. When the car approaches a traffic light, the traffic light will send 4 signals to the car. These signals are converted into electric signals by a sensor mounted on the top of the car. These electric signals are each connected to the controller inputs of ***EVALUATE***, ***RED***, ***YELLOW*** and ***GREEN***. Based on different combinations of these four inputs, the controller must decide the logical values to drive at the outputs ***BRAKE*** and ***SPEED***.

When the input ***RED*** is at logical '1', the car must stop. When ***YELLOW*** is at logical '1', the car must slow down. When ***GREEN*** is at logical '1', the car can accelerate. For any other combinations that involve more than either ***RED***, ***YELLOW*** or ***GREEN*** having a logical '1', the car must stop.

The car stops when ***BRAKE*** is at logical '1' and ***SPEED*** is at logical '0'. When ***BRAKE*** is at logical '0' and ***SPEED*** at logical '1', the car will accelerate. When both ***SPEED*** and ***BRAKE*** are at logical '0', the car will slow down prior to stopping.

The evaluation of the three inputs (***RED***, ***YELLOW*** and ***GREEN***) will occur at every rising edge of ***EVALUATE***. ***EVALUATE*** is a signal that oscillates whenever the car approaches a traffic light.

TABLE 19 Pin Description of a Car Traffic Controller Module

Pin	Input/Output	Description
EVALUATE	Input	At every rising edge of ***EVALUATE***, polling of input signals occurs
RED	Input	When at logical '1', the car must stop
YELLOW	Input	When at logical '1', with ***GREEN*** and ***RED*** at logical '0', the car must slow down
GREEN	Input	When at logical '1', with ***RED*** and ***YELLOW*** at logical '0', the car must accelerate
BRAKE	Output	Logical '1' means the car stops
SPEED	Output	Logical '1' means the car accelerates.

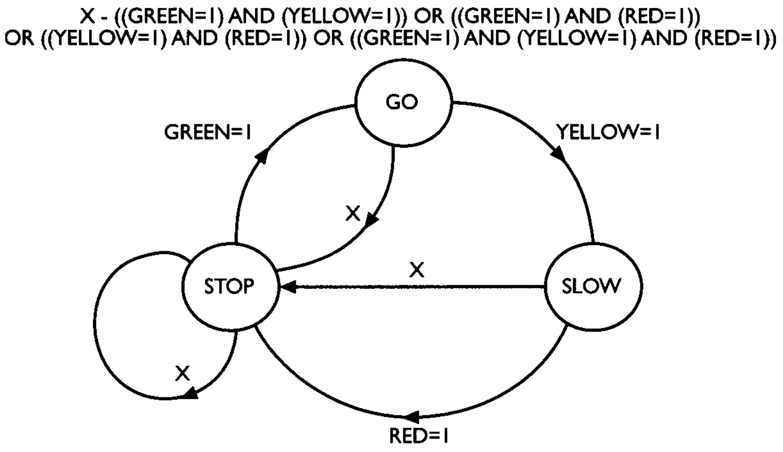

X - ((GREEN=1) AND (YELLOW=1)) OR ((GREEN=1) AND (RED=1))
OR ((YELLOW=1) AND (RED=1)) OR ((GREEN=1) AND (YELLOW=1) AND (RED=1))
GO
GREEN=1
YELLOW=1
X
STOP
X
SLOW
X
RED=1

FIGURE 41 State Diagram for Car Traffic Controller Module.

A state diagram (Fig. 41) is drawn to represent the functionality of the Car Traffic Controller module.

EXAMPLE 38 Example of Synthesizable VHDL for Car Traffic Controller Module

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY state_machine_ent IS
PORT (
        EVALUATE : IN std_logic;
        GREEN : IN std_logic;
        YELLOW : IN std_logic;
        RED : IN std_logic;
        SPEED : OUT std_logic;
        BRAKE : OUT std_logic
        );
END state_machine_ent;

ARCHITECTURE state_machine_arch OF state_machine_ent IS
TYPE state_type IS (GO, SLOW, STOP);
SIGNAL present_state, next_state : state_type := GO;
SIGNAL sig_all_light : std_logic_vector (2 downto 0);

BEGIN
        SPEED <= '1' WHEN ((sig_all_light = "100") AND
             ((present_state = GO) OR
             (present_state = STOP))) ELSE '0';
```

```
BRAKE <= '0' WHEN (((present_state = GO) AND
     ((sig_all_light = "100") OR (sig_all_light = "010")))
     OR ((present_state = SLOW) AND (sig_all_light =
     "010")) OR ((present_state = STOP) AND (sig_all_light
     = "100"))) ELSE '1';


PROCESS (GREEN, YELLOW, RED, present_state)
VARIABLE all_light : std_logic_vector (2 downto 0);
BEGIN
     all_light := GREEN & YELLOW & RED;
     sig_all_light <= all_light;
     CASE present_state IS
          WHEN GO =>
               IF (all_light = "100") THEN
                    next_state <= GO;
               ELSIF (all_light = "010") THEN
                    next_state <= SLOW;
               ELSE
                    next_state <= STOP;
               END IF;
          WHEN SLOW =>
               IF (all_light = "010") THEN
                    next_state <= SLOW;
               ELSIF (all_light = "001") THEN
                    next_state <= STOP;
               ELSE
                    next_state <= STOP;
               END IF;
          WHEN STOP =>
               IF (all_light = "001") THEN
                    next_state <= STOP;
               ELSIF (all_light = "100") THEN
                    next_state <= GO;
               ELSE
                    next_state <= STOP;
               END IF;
          WHEN OTHERS =>
               next_state <= STOP;
     END CASE;
END PROCESS;
```

Generation of next state based on present state of the state machine

```
PROCESS (EVALUATE, next_state)
BEGIN
     IF (EVALUATE = '1' AND EVALUATE'EVENT) THEN
          present_state <= next_state;
```

Assigning next state back to present state

```
            END IF;
        END PROCESS;

END state_machine_arch;
```

A testbench is written to exercise the functionality of the state machine.

EXAMPLE 39 Example of Testbench for Car Traffic Controller Module

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
```

-- PACKAGE for state_machine

```
PACKAGE state_machine_package IS

        CONSTANT CYCLE : TIME := 50 ns;
        SIGNAL sig_green : std_logic;
        SIGNAL sig_yellow : std_logic;
        SIGNAL sig_red : std_logic;

        PROCEDURE load_data (
            SIGNAL GREEN : out std_logic;
            SIGNAL YELLOW : out std_logic;
            SIGNAL RED : out std_logic);
END state_machine_package;

PACKAGE BODY state_machine_package IS

PROCEDURE load_data (
        SIGNAL GREEN : out std_logic;
        SIGNAL YELLOW : out std_logic;
        SIGNAL RED : out std_logic) IS
        BEGIN
            GREEN <= sig_green;
            YELLOW <= sig_yellow;
            RED <= sig_red;
               wait for CYCLE;
END load_data;
END state_machine_package;
```

-- Testbench for state_machine

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.state_machine_package.ALL;


ENTITY state_machine_tb_ent IS
END state_machine_tb_ent;


ARCHITECTURE state_machine_tb_arch OF state_machine_tb_ent IS
COMPONENT state_machine_ent
PORT (
        EVALUATE : IN std_logic;
        GREEN : IN std_logic;
        YELLOW : IN std_logic;
        RED : IN std_logic;
        SPEED : OUT std_logic;
        BRAKE : OUT std_logic
        );
END COMPONENT;
SIGNAL EVALUATE : std_logic := '0';
SIGNAL GREEN : std_logic;
SIGNAL YELLOW : std_logic;
SIGNAL RED : std_logic;
SIGNAL SPEED : std_logic;
SIGNAL BRAKE : std_logic;
BEGIN
        DUT: state_machine_ent PORT MAP (EVALUATE, GREEN, YELLOW, RED,
SPEED, BRAKE);


EVALUATE <= NOT EVALUATE AFTER CYCLE/2;
PROCESS
VARIABLE temp : std_logic_vector (2 downto 0);
BEGIN


FOR i IN 0 to 7 LOOP
        temp := CONV_STD_LOGIC_VECTOR (i, 3);
        sig_green <= temp (0);
        sig_yellow <= temp (1);
        sig_red <= temp (2);
        wait for CYCLE;
        load_data (GREEN, YELLOW, RED);
END LOOP;


END PROCESS;
END state_machine_tb_arch;
```

```
CONFIGURATION state_machine_tb_config OF state_machine_tb_ent IS
FOR state_machine_tb_arch
        FOR ALL: state_machine_ent
             USE ENTITY WORK.state_machine_ent(state_machine_arch);
        END FOR;
END FOR;
END state_machine_tb_config;
```

Figure 42 shows the simulation results from the testbench of Example 39. Output ***SPEED*** drives logical '1' only when input ***GREEN*** is at logical '1'. If more than one of the three inputs ***GREEN***, ***YELLOW***, and ***RED*** are at logical '1', output ***SPEED*** must always drive a value of logical '0' and ***BRAKE*** a value of logical '1' to stop the car. The car needs to stop if the traffic light that it is approaching is broken.

To synthesize this car traffic controller state machine using Synopsys's Design Compiler, first a set of design constraints must be declared on the design. This set of constraints includes input delays, output delays, and clock period information. Appendix D shows the full-scale synthesis of this state machine design which includes design, constraints and synthesis tweaks to obtain optimal synthesis results using Synopsys's Design Compiler. Appendix D also shows how Synopsys's FSM Compiler can be used to obtain optimal synthesis results for state machines.

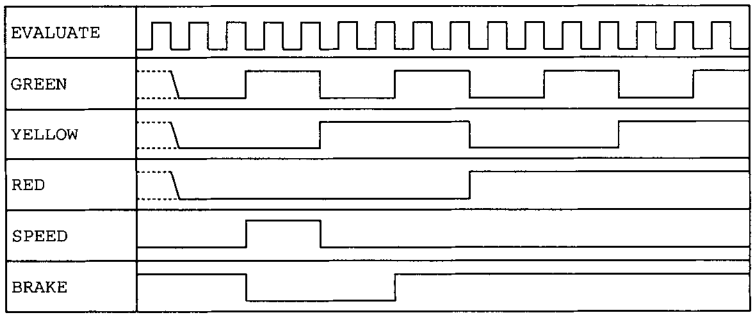


FIGURE 42 Timing Diagram Showing Simulation Results of a Car Traffic Controller.

6 PIPELINE MICROCONTROLLER SYNTHESIZABLE DESIGN

This chapter will bring the reader through the many steps needed in designing a full-scale design project. The example used in this chapter for the design project is a 3-stage pipeline microcontroller.

The contents of this chapter include:

- definition of instruction set for the microcontroller;
- architectural definition of the design using flow charts and truth tables;
- microarchitectural and module interface definitions;
- testbench to simulate the design; and
- timing waveforms of simulation results.

6.1 INSTRUCTION SET DEFINITION

The first thing that needs to be tackled prior to designing a microcontroller is to define an instruction set that the microcontroller can decode and execute.

For a barebones microcontroller, at least eight instructions are required. These instructions are basic operations that enable functionality of a microcontroller. Table 20 shows the instruction set with a description of each instruction.

TABLE 20 Description of Microcontroller Instruction Set

Instruction Set	Description
MOVE `<source1>`, `<destination>`	Move the contents of `<source1>` to `<destination>`
ADD `<source1>`, `<source2>`, `<destination>`	Add the contents of `<source1>` and contents of `<source2>` and put the result in `<destination>`

continued

TABLE 20 ***continued***

Instruction Set	Description
SUB `<source1>`, `<source2>`, `<destination>`	Subtract the contents of `<source1>` with contents of `<source2>` and put the results in `<destination>`
MUL `<source1>`, `<source2>`, `<destination>`	Multiply the contents of `<source1>` with the contents of `<source2>` and put the result in `<destination>`
CJE `<source1>`, `<source2>`, `<CODE>`	Compare contents of `<source1>` with the contents of `<source2>` and if they are equal, jump to the portion of the instruction that is associated with the label `<CODE>`
LOAD `<value>`, `<destination>`	Load the contents of `<value>` into `<destination>`
READ `<destination>`	Read the contents of `<destination>` and drive the data on the output pins
NOP	No instruction

Having eight instructions would physically translate to three ($2^3 = 8$) input pins. If additional instructions are required, the amount of bits to represent the instruction set can be expanded.

Note: For the instruction set, `<source1>`, `<source2>` and `<destination>` must be represented by either *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15* (which are internal registers of 32 bits each).

6.2 ARCHITECTURAL DEFINITION

With the set of instructions defined, we must now define the interface of the microcontroller.

To keep this task as simple as possible, several features that are normally found on a commercial microcontroller are not included. This simplicity is meant to help the reader focus on understanding the synthesizable code and not on learning pipeline designs.

Some of the features that have not been included are:

- no 'register scoreboarding',
- no access to external memory.

If access to external memory is required, the architecture can be expanded to consist of additional signals to the external memory module (and adding additional instructions to enable access to the external memory).

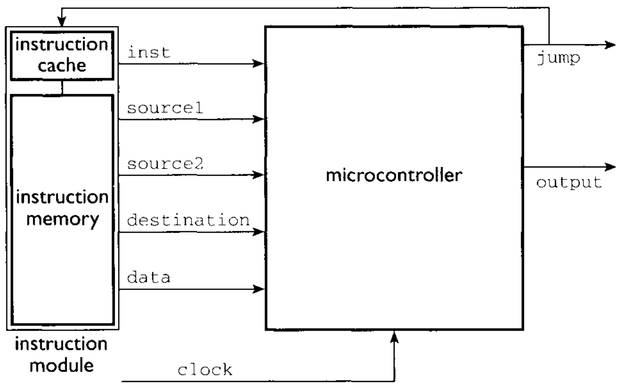


FIGURE 43 Diagram Showing Interface of Microcontroller.

If scoreboarding is required, the internal 16 registers that have 32 bits each can be expanded to 33 bits each, with one of the 33 bits for each register to be used as a scoreboard bit. of course, additional signals and circuitry will also be required to handle the scoreboarding.

Another assumption for the microcontroller design is that an external instruction cache and instruction memory supply the instructions and data into the microcontroller.

Figure 43 shows the microcontroller's interface with six inputs and two outputs. It communicates with an external *instruction module* that loads instructions (with data) to the microcontroller. The output ***jump*** is an input to the *instruction module* as a qualifier for the *instruction module* to branch to another portion of the code when a branch is taken during the ***CJE*** instruction.

Table 21 contains the description for all of the input and output signals on the interface of the microcontroller.

TABLE 21 Description of Microcontroller Interface Signals

Pins	Input/Output	Bit Size	Description
inst	Input	3	Instruction to be performed by the microcontroller 000 — ***MOVE*** 001 — ***ADD*** 010 — ***SUB*** 011 — ***MUL*** 100 — ***CJE*** 101 — ***LOAD*** 110 — ***READ*** 111 — ***NOP***
source1	input	4	*<source1>*

continued

TABLE 21 ***continued***

Pins	Input/Output	Bit Size	Description
source2	input	4	*<source2>*
destination	input	4	*<destination>*
data	input	32	Input *data* for ***LOAD*** instruction
clock	input	1	The microcontroller uses this clock input to operate its pipeline on a clock-by-clock cycle
jump	output	1	Asserted high when the ***CJE*** instruction compares *<source1>* and *<source2>* as having the same value
output	output	32	output of data for ***READ*** instruction

For ***source1***, ***source2*** and ***destination*** inputs, these are represented by 4 bits as the microcontroller has 16 internal registers. These registers are 32 bits each. All of the instructions that utilize *<source1>*, *<source2>*, and *<destination>* must be one of these 16 registers.

Table 22 shows the representation of the 16 registers by binary values of input signals ***source1***, ***source2***, and ***destination***.

TABLE 22 Representation of Sixteen Internal Registers for the Microcontroller Design

source1/source2/destination	Register	Register name
0000	register 0	*reg0*
0001	register 1	*reg1*
0010	register 2	*reg2*
0011	register 3	*reg3*
0100	register 4	*reg4*
0101	register 5	*reg5*
0110	register 6	*reg6*
0111	register 7	*reg7*
1000	register 8	*reg8*
1001	register 9	*reg9*
1010	register 10	*reg10*
1011	register 11	*reg11*
1100	register 12	*reg12*
1101	register 13	*reg13*
1110	register 14	*reg14*
1111	register 15	*reg15*

6.3 PIPELINE DEFINITION

With the architectural and interface signals definition completed, we will proceed to the actual definition of the design of a microcontroller.

The design is based on a pipeline. To gain more information on the advantages, disadvantages and the reasons for implementing pipeline designs, please refer to *Computer Architecture: A Quantitative Approach* by John L Hennessy and David A. Patterson (Morgan Kaufmann Publication) and *Computer Organization & Design: The Hardware/Software Interface* by David A. Patterson and John L. Hennessy (Morgan Kaufmann Publication).

By defining our microcontroller to be a 3-stage pipeline design, the microcontroller is separated into a *predecode* stage, *decode* stage, and an *execute* stage. See Fig. 44 for the diagram.

Predecode stage is the stage that interfaces with the external *instruction module*. The input signals for this stage receive the instructions and data from the *instruction module*.

The *decode* stage decodes the input stimulus from the external *instruction module* through the *predecode* stage. It also decodes the input stimulus. The output of this stage will pass all the necessary information to the *execute* stage whereby the instruction is executed.

As the microcontroller is a pipeline design, the instruction going in must pass each stage on a clock-by-clock basis, just like a pipeline. The design is of course much more complicated than it sounds because pipeline designs must come prepared with features that include execute bypassing. Execute bypassing is in general needed to bypass data through a module because the data might be "outdated." Bypassing is a mechanism often found in pipeline designs. The details of bypassing and how it can affect a design will be discussed in detail later in this chapter.

A pipeline design assumes we have a set of four instructions being passed from the instruction module to the microcontroller.

```
LOAD #FA, reg0
LOAD #1, reg1
ADD reg0, reg1, reg13
READ reg13
```

As these instructions pass through each stage of the pipeline on every clock cycle, a total of 6 clock cycles are required for the microcontroller to complete the execution of the set of 4 instructions.

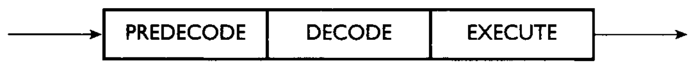


FIGURE 44 Diagram Showing the Pipeline Stage of the Microcontroller.

Time	Predecode stage	Decode stage	Execute stage
1	LOAD `#FA, reg0`		
2	LOAD `#1, reg1`	LOAD `#FA, reg0`	
3	ADD `reg0, reg1, reg13`	LOAD `#1, reg1`	LOAD `#FA, reg0`
4	READ `reg13`	ADD `reg0, reg1, reg13`	LOAD `#1, reg1`
5		READ `reg13`	ADD `reg0, reg1, reg13`
6			READ `reg13`

FIGURE 45 Instruction Execution in a Pipeline.

Figure 45 shows the instructions being executed in a pipeline manner. Execution of 4 instructions would require a total of 6 clock cycles whereby the first 2 clock cycles are required to pass the first instruction from *predecode* stage to *execute* stage.

6.4 MICROARCHITECTURE DEFINITION FOR THE PIPELINE MICROCONTROLLER

To design the three stages of the pipeline, the microarchitectural implementation of the design is considered. Functional partitioning and inter-functional blocks signaling interface must be taken into consideration. Functional partitioning is an important aspect of design. The different functions of a design can basically be grouped into different partitions with each partition having to perform a certain function. Good functional partitioning is important to achieve if the design is to have the optimum performance and area utilization for a given architecture.

We begin with defining the functional blocks required for the design. We split the function of the microcontroller into 4 different blocks — the *precode* block, *decode*

Note: This pipeline microcontroller can in actual fact be designed using only one block. Partitioning into different functional blocks is not needed as Synopsys's Design Compiler is capable of synthesizing the whole pipeline microcontroller (top-down synthesis approach). However, when dealing with much bigger designs that consist of many thousands of gates, it is important to partition the design into different functional blocks. Each block is individually coded and synthesized (bottoms-up synthesis approach). Chapter 8 discusses in detail a different synthesis approach (top-down and bottoms-up) on designs with different gate counts.

This example of pipeline microcontroller is partitioned into different functional blocks to illustrate to the reader how functional partitioning can be achieved.

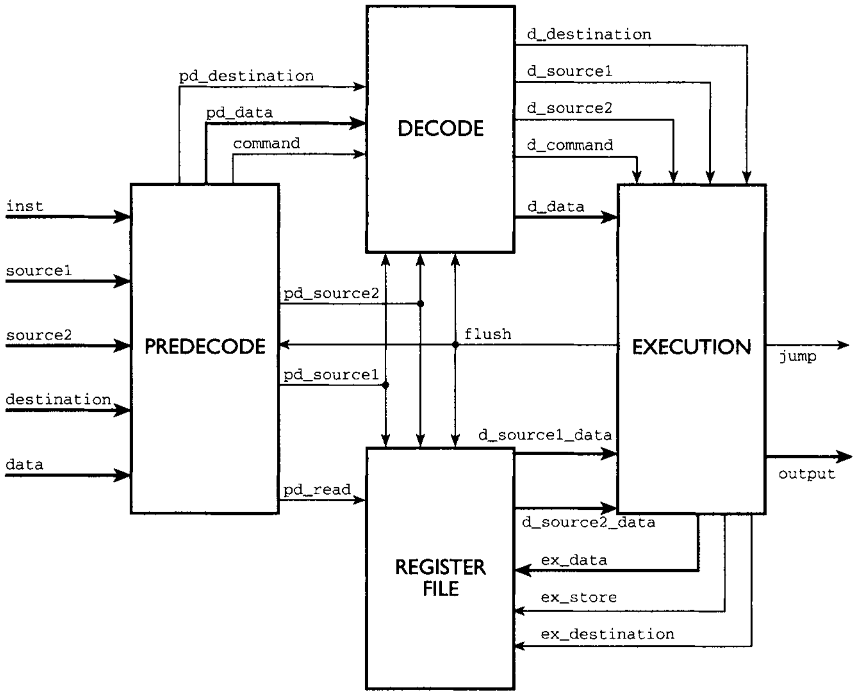


FIGURE 46 Microarchitectural Definition for Fullchip Microcontroller.

block, *register file* block, and *execute* block. Each of these blocks has a different functionality.

Figure 46 shows the fullchip block interconnect for the microcontroller. The clock signal is not shown in the diagram but it should be noted that the clock signal is routed as an input to every block in the microcontroller. Another important signal is the **`flush`** signal. It is generated from *execute* block and input to *predecode*, *decode* and *register file* block.

The signal **`flush`** is asserted high when a branch occurs. This informs other functional blocks that a branch has occurred. When a branch takes place, all current instructions in each block must be flushed. This would allow new instructions (where the branch is headed to) to be passed from the *instruction module* to the microcontroller. The **`jump`** signal, which is an output from *execute* block, is also an output from the microcontroller to the *instruction module*. It is used by the *instruction module* as an indicator that a branch has occured and the *instruction module* should send new instructions (where the branch heads to) to the microcontroller.

This would assume that the *instruction module* will have its own internal circuitry to keep track of the designation of the branch.

Before we proceed with the code for functional blocks, we will need to write a VHDL package file that is referenced by all the functional blocks.

As was done in to Example 2, a library has been defined and linked to directory ***WORK***.

1. Create a file <filename> in directory SOURCE and in this file type the contents below:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

PACKAGE pipeline_package IS
TYPE command_type IS (MOVE, ADD, SUB, MUL, CJE, LOAD,
READ, NOP);
TYPE register_type IS (reg0, reg1, reg2, reg3, reg4,
reg5, reg6, reg7, reg8, reg9, reg10, reg11, reg12, reg13,
reg14, reg15);
TYPE array_size IS array (0 to 15) of std_logic_vector(31
downto 0);
CONSTANT ZERO : std_logic_vector (31 downto 0) := (others
=> '0');
END pipeline_package;
```

2. Compile the file <filename> into the library link to directory ***WORK***.
3. When this is completed, you will have a VHDL library linked to the directory ***WORK*** containing the precompiled VHDL package *pipeline_package*.

6.4.1 Predecode Block

Predecode block handles the functionality of the *predecode* stage in the pipeline of the microcontroller.

This block interfaces mainly with the external *instruction module*. It accepts instruction and data from the *instruction module* and predecodes those instructions before sending them to *decode* block. It also has interface signals with *register file* block. These signals are used to request passing of data from the internal registers (*reg0* to *reg15*) of *register file* block to *execute* block. See Table 23 for a description and Fig. 47 for a diagram of predecode block interface signals.

TABLE 23 Description of Predecode Block Interface Signals

Signal name	I/O	Description
clock	Input	Clock input
inst	Input	A 3-bit instruction bus
		"000" = ***MOVE***
		"001" = ***ADD***

Signal name	I/O	Description
		"010" = ***SUB*** "011" = ***MUL*** "100" = ***CJE*** (compare and jump if equal) "101" = ***LOAD*** "110" = ***READ*** "111" = ***NOP*** (no operation)
source1	input	A 4-bit bus to specify register to be used as *<source1>* "0000" = *reg* "1000" = *reg8* "0001" = *reg1* "1001" = *reg9* "0010" = *reg2* "1010" = *reg10* "0011" = *reg3* "1011" = *reg11* "0100" = *reg4* "1100" = *reg12* "0101" = *reg5* "1101" = *reg13* "0110" = *reg6* "1110" = *reg14* "0111" = *reg7* "1111" = *reg15*
source2	input	A 4-bit bus to specify register to be used as *<source2>* "0000" = *reg0* "1000" = *reg8* "0001" = *reg1* "1001" = *reg9* "0010" = *reg2* "1010" = *reg10* "0011" = *reg3* "1011" = *reg11* "0100" = *reg4* "1100" = *reg12* "0101" = *reg5* "1101" = *reg13* "0110" = *reg6* "1110" = *reg14* "0111" = *reg7* "1111" = *reg15*
destination	input	A 4-bit bus to specify register to be used as *<destination>* "0000" = *reg0* "1000" = *reg8* "0001" = *reg1* "1001" = *reg9* "0010" = *reg2* "1010" = *reg10* "0011" = *reg3* "1011" = *reg11* "0100" = *reg4* "1100" = *reg12* "0101" = *reg5* "1101" = *reg13* "0110" = *reg6* "1110" = *reg14* "0111" = *reg7* "1111" = *reg15*
data	input	A 32-bit bus for data input during ***LOAD*** instruction
flush	input	asserted high when a branch occurs
command	output	Instruction to be passed to *decode* block. It will have value type of *command_type* whereby *command_type* is defined to be of type ***MOVE***, ***ADD***, ***SUB***, ***MUL***, ***CJE***, ***LOAD***, ***READ*** and ***NOP***.

TABLE 23 *continued*

Signal name	I/O	Description
pd_source1	output	Indication of <*source1*>, which is of type *register_type* and it can have a value of *reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, reg8, reg9, reg10, reg11, reg12, reg13, reg14, reg15*
pd_source2	output	Indication of <*source2*> which is of type *register_type* and it can have a value of *reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, reg8, reg9, reg10, reg11, reg12, reg13, reg14, reg15*
pd_destination	output	Indication of <*destination*>, which is of type *register_type* and it can have a value of *reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, reg8, reg9, reg10, reg11, reg12, reg13, reg14, reg15*
pd_data	output	A 32-bit bus for passing of data to *decode* block
pd_read	output	Single bit and asserted high to indicate to the *register file* to read its internal register <*pd_source1*> and <*pd_source2*>. Data read are put on the output as <*source1_data*> and <*source2_data*>.

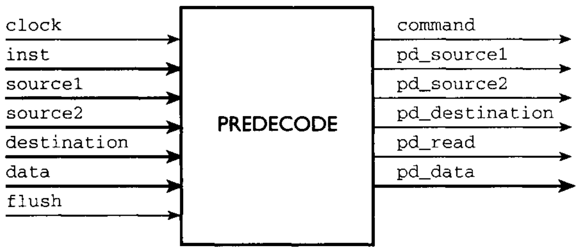


FIGURE 47 Diagram Showing the Predecode Block Interface Signals.

EXAMPLE 40 Example of Predecode Block Synthesizable VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY predecode_ent IS
PORT (
        clock : IN std_logic;
        inst : IN std_logic_vector (2 downto 0);
        source1 : IN std_logic_vector (3 downto 0);
        source2 : IN std_logic_vector (3 downto 0);
        destination : IN std_logic_vector (3 downto 0);
        data : IN std_logic_vector (31 downto 0);
        flush : IN std_logic;
        command : OUT command_type;
        pd_source1 : OUT register_type;
        pd_source2 : OUT register_type;
        pd_destination : OUT register_type;
        pd_data : OUT std_logic_vector (31 downto 0);
        pd_read : OUT std_logic
        );
END predecode_ent;


ARCHITECTURE predecode_arch OF predecode_ent IS
BEGIN
        PROCESS (clock, inst, source1, source2, destination, data,
                flush)
        VARIABLE internal_command : command_type := NOP;
        VARIABLE internal_source1 : register_type := reg0;
        VARIABLE internal_source2 : register_type := reg0;
        VARIABLE internal_destination : register_type := reg0;
        BEGIN
             IF (clock = '1' AND clock'EVENT) THEN
                  IF (flush = '0') THEN
                       CASE inst IS
                            WHEN "000" =>
                                 pd_read <= '1';
                                 internal_command := MOVE;
                            WHEN "001" =>
                                 pd_read <= '1';
                                 internal_command := ADD;
                            WHEN "010" =>
                                 pd_read <= '1';
                                 internal_command := SUB;
                            WHEN "011" =>
                                 pd_read <= '1';
                                 internal_command := MUL;
                            WHEN "100" =>
                                 pd_read <= '1';
                                 internal_command := CJE;
                            WHEN "101" =>
```

Decoding on ***inst*** input into *command_type* values.

pd_read is set to '1' when an instruction requires reading the values of internal registers.

```
                        pd_read <= '0';
                        internal_command := LOAD;
                WHEN "110" =>
                        pd_read <= '1';
                        internal_command := READ;
                WHEN "111" =>
                        pd_read <= '0';
                        internal_command := NOP;
                WHEN OTHERS =>
                        NULL;
        END CASE;
        CASE source1 IS
                WHEN "0000" =>
                        internal_source1 := reg0;
                WHEN "0001" =>
                        internal_source1 := reg1;
                WHEN "0010" =>
                        internal_source1 := reg2;
                WHEN "0011" =>
                        internal_source1 := reg3;
                WHEN "0100" =>
                        internal_source1 := reg4;
                WHEN "0101" =>
                        internal_source1 := reg5;
                WHEN "0110" =>
                        internal_source1 := reg6;
                WHEN "0111" =>
                        internal_source1 := reg7;
                WHEN "1000" =>
                        internal_source1 := reg8;
                WHEN "1001" =>
                        internal_source1 := reg9;
                WHEN "1010" =>
                        internal_source1 := reg10;
                WHEN "1011" =>
                        internal_source1 := reg11;
                WHEN "1100" =>
                        internal_source1 := reg12;
                WHEN "1101" =>
                        internal_source1 := reg13;
                WHEN "1110" =>
                        internal_source1 := reg14;
                WHEN "1111" =>
                        internal_source1 := reg15;
                WHEN OTHERS =>
                        NULL;
        END CASE;
```

Decoding of ***source1*** to determine register used for `<source1>`.

Usage of ***CASE*** statement to represent a huge multiplexer. ***IF*** statement is not used as it would generate priority encoder.

Decoding of **source2** to determine register used for *<source2>*.

```
CASE source2 IS
     WHEN "0000" =>
          internal_source2 := reg0;
     WHEN "0001" =>
          internal_source2 := reg1;
     WHEN "0010" =>
          internal_source2 := reg2;
     WHEN "0011" =>
          internal_source2 := reg3;
     WHEN "0100" =>
          internal_source2 := reg4;
     WHEN "0101" =>
          internal_source2 := reg5;
     WHEN "0110" =>
          internal_source2 := reg6;
     WHEN "0111" =>
          internal_source2 := reg7;
     WHEN "1000" =>
          internal_source2 := reg8;
     WHEN "1001" =>
          internal_source2 := reg9;
     WHEN "1010" =>
          internal_source2 := reg10;
     WHEN "1011" =>
          internal_source2 := reg11;
     WHEN "1100" =>
          internal_source2 := reg12;
     WHEN "1101" =>
          internal_source2 := reg13;
     WHEN "1110" =>
          internal_source2 := reg14;
     WHEN "1111" =>
          internal_source2 := reg15;
     WHEN OTHERS =>
          NULL;
END CASE;
CASE destination IS
     WHEN "0000" =>
          internal_destination := reg0;
     WHEN "0001" =>
          internal_destination := reg1;
     WHEN "0010" =>
          internal_destination := reg2;
     WHEN "0011" =>
          internal_destination := reg3;
          WHEN "0100" =>
          internal_destination := reg4;
```

Usage of ***CASE*** statement to represent a huge multiplexer. ***IF*** statement is not used as it would generate priority encoder.

Decoding of **destination** to determine register used for *<destination>*.

Usage of ***CASE*** statement to represent a huge multiplexer. ***IF*** statement is not used as it would generate priority encoder.

```
                                WHEN "0101" =>
                                     internal_destination := reg5;
                                WHEN "0110" =>
                                     internal_destination := reg6;
                                WHEN "0111" =>
                                     internal_destination := reg7;
                                WHEN "1000" =>
                                     internal_destination := reg8;
                                WHEN "1001" =>
                                     internal_destination := reg9;
                                WHEN "1010" =>
                                     internal_destination :=reg10;
                                WHEN "1011" =>
                                     internal_destination :=reg11;
                                WHEN "1100" =>
                                     internal_destination :=reg12;
                                WHEN "1101" =>
                                     internal_destination :=reg13;
                                WHEN "1110" =>
                                     internal_destination :=reg14;
                                WHEN "1111" =>
                                     internal_destination :=reg15;
                                WHEN OTHERS =>
                                     NULL;
                           END CASE;
```

Default ***pd_data*** to value of ***data*** when a ***LOAD*** instruction is decoded.

```
                           IF (internal_command = LOAD) THEN
                                pd_data <= data;
                           ELSE
                                pd_data <=
                                (others => '0');
                           END IF;
                      ELSE
```

Default to ***NOP*** when ***flush*** is detected '1'.

```
                           pd_data <= (others => '0');
                           internal_command := NOP;
                           pd_read <= '0';
                      END IF;
                      command <= internal_command;
                      pd_source1 <= internal_source1;
                      pd_source2 <= internal_source2;
                      pd_destination <= internal_destination;
                 END IF;
       END PROCESS;
END predecode_arch;
```

With the code in Example 40 for *predecode* block, a testbench is written to inject stimulus for simulation to check the functionality.

EXAMPLE 41 Example of VHDL Code for Testbench to Check for Correct Functionality

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY predecode_tb_ent IS
END predecode_tb_ent;

ARCHITECTURE predecode_tb_arch OF predecode_tb_ent IS
COMPONENT predecode_ent
PORT (
     clock : IN std_logic;
     inst : IN std_logic_vector (2 downto 0);
     source1 : IN std_logic_vector (3 downto 0);
     source2 : IN std_logic_vector (3 downto 0);
     destination : IN std_logic_vector (3 downto 0);
     data : IN std_logic_vector (31 downto 0);
     flush : IN std_logic;
     command : OUT command_type;
     pd_source1 : OUT register_type;
     pd_source2 : OUT register_type;
     pd_destination : OUT register_type;
     pd_data : OUT std_logic_vector (31 downto 0);
     pd_read : OUT std_logic
     );
END COMPONENT;

SIGNAL data : std_logic_vector (31 downto 0);
SIGNAL source1 : std_logic_vector (3 downto 0);
SIGNAL source2 : std_logic_vector (3 downto 0);
SIGNAL clock : std_logic := '0';
SIGNAL inst : std_logic_vector (2 downto 0);
SIGNAL destination : std_logic_vector (3 downto 0);
SIGNAL flush : std_logic;
SIGNAL command : command_type;
SIGNAL pd_source1 : register_type;
SIGNAL pd_source2 : register_type;
SIGNAL pd_destination : register_type;
SIGNAL pd_data : std_logic_vector (31 downto 0);
SIGNAL pd_read : std_logic;

CONSTANT CYCLE : TIME := 50 ns;
```

```
BEGIN

DUT: predecode_ent port map(clock, inst, source1, source2,
                            destination, data, flush, command,
                            pd_source1, pd_source2,
                            pd_destination, pd_data, pd_read);

clock <= NOT clock AFTER CYCLE/2;

PROCESS
BEGIN

-- default output set to 0
source1 <= "0000";
source2 <= "0000";
destination <= "0000";
data <= ZERO;
inst <= "111";

-- flush is 0, no flushing
flush <= '0';

-- load "4592fa83" into reg0 instruction
inst <= "101";
data <= "01000101100100101111101010000011";
destination <= "0000";
wait for CYCLE;

-- load "00000001" into reg15 instruction
inst <= "101";
data <= "00000000000000000000000000000001";
destination <= "1111";
wait for CYCLE;

-- mov reg0, reg8
inst <= "000";
source1 <= "0000";
destination <= "1000";
wait for CYCLE;

-- add reg0, reg15, reg1
inst <= "001";
source1 <= "0000";
source2 <= "1111";
destination <= "0001";
wait for CYCLE;

-- no operation
inst <= "111";
```

```
wait for CYCLE;
inst <= "111";
wait for CYCLE;

END PROCESS;
END predecode_tb_arch;

CONFIGURATION predecode_tb_config OF predecode_tb_ent IS
  FOR predecode_tb_arch
    FOR ALL : predecode_ent
      USE ENTITY WORK.predecode_ent(predecode_arch);
    END FOR;
  END FOR;
END predecode_tb_config;
```

The testbench of Example 41 simulates *predecode* block with the following instructions:

```
LOAD #4592fa83, reg0
LOAD #00000001, reg15
MOVE reg0, reg8
ADD reg0, reg15, reg1
NOP
```

Figure 48 shows the waveform from the simulation of the testbench.

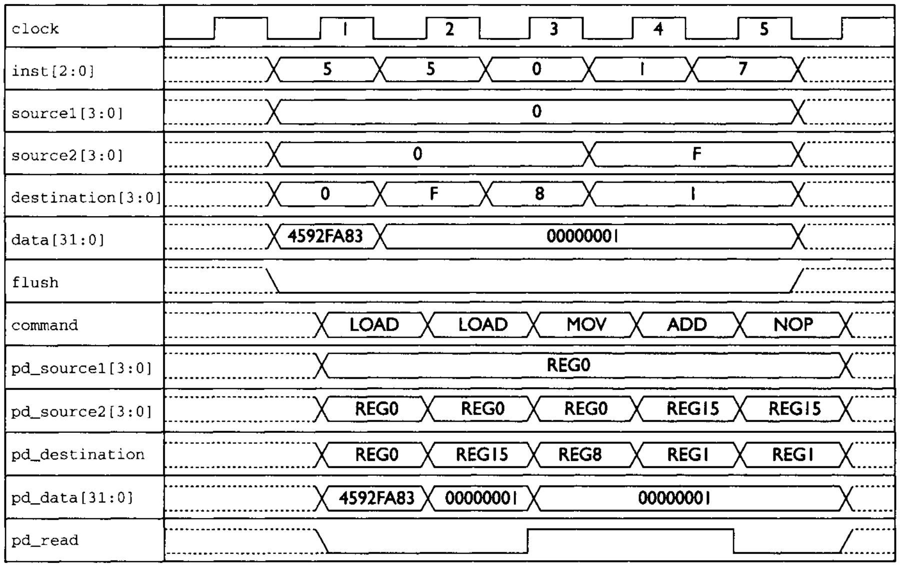


FIGURE 48 Timing Waveform for Predecode Testbench.

From Fig. 48:

1. At the rising edge of 1st clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **inst** is at a value of 5. *Predecode* block will translate to a **LOAD** instruction on output **COMMAND**.
   - Since the command is **LOAD**, **pd_read** is driven to a logical '0', indicating to *register file* block that no reading of any registers is necessary.
   - **source1** and **source2** show a value of 0. This is decoded to *reg0*. **pd_source1** and **pd_source2** drive the value *reg0* of type *register_type*.
   - **destination** has a value of 0. This is decoded to *reg0*. **pd_destination** drives the value *reg0* of type *register_type*.
   - **data** shows value of 4592FA83. This value is driven onto the **pd_data** output.
   - At the end of 1st clock, *predecode* block has decoded the instruction **LOAD** of value of 4592FA83 into register *reg0* as designated by **destination**.
2. At the rising edge of 2nd clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **command** drives **LOAD** instruction since **inst** is showing a value of 5.
   - **source1** and **source2** still remains at the value of 0. **pd_source1** and **pd_source2** continue to drive *reg0*.
   - **destination** changes to value F and this decodes to *reg15*. **pd_destination** changes from *reg0* to *reg15*.
   - **data** is at value 1. **pd_data** drives value 1.
   - At the end of the 2nd clock, an instruction **LOAD** value of data '00000001' into register *reg15* as designated by **destination** is decoded.
3. At the rising edge of 3rd clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **inst** is at value of 0. This decodes to a **MOVE** instruction. **command** drives **MOVE**.
   - **source1** and **source2** are still at value 0. **pd_source1** and **pd_source2** drive *reg0*.
   - **destination** has a value of 8. This decodes to *reg8*. **pd_destination** drives *reg8*.
   - **data** remains the same. But since the instruction driven out is of type **MOVE**, **pd_data** drives all zeros.
   - The instruction **MOVE** involves the reading of contents of certain registers. **pd_read** is asserted to logical '1' to indicate to *register file* block to read the contents of register designated by **pd_source1** and **pd_source2**.
   - At the end of the 3rd clock cycle, an instruction *MOVE* of contents *reg0* (as designated by **source1**) into *reg8* (which is designated by **destination**) is decoded.

4. At the rising edge of 4th clock,
   - ***flush*** is at logical '0'. No flushing should occur.
   - ***inst*** is now at a value of '1'. ***command*** drives ***ADD***.
   - ***source1*** is still 0. ***pd_source1*** continues to drive *reg0*. ***source2*** has a value of F that decodes to *reg15*. ***pd_source2*** drives *reg15*. ***destination*** has a value of 1 which decodes to *reg1*. ***pd_destination*** drives *reg1*.
   - ***pd_read*** asserted to a logical '1' to indicate to *register file* block to read the contents of register designated by ***pd_source1*** and ***pd_source2***.
   - ***pd_data*** drives all zeros since this command is ***ADD***.
   - At the end of the 4th clock cycle, the instruction ***ADD*** of contents of *reg0* with the contents of *reg15* is decoded. The result is to be stored in *reg1*.
5. At the rising edge of the 5th clock,
   - ***flush*** is at logical '0'. No flushing should occur.
   - ***inst*** has a value of 7. This is decoded into instruction ***NOP***. ***command*** drives ***NOP***.
   - ***pd_read*** de-asserts back to logical '0' since the instruction is ***NOP***. All other outputs remain the same but are invalid.
   - This cycle decodes the instruction ***NOP*** (no operation).

6.4.2 Decode Block

Decode block makes up part of the *decode* stage in the pipeline of the microcontroller. The other half of this stage is made up of *the register file* block.

Decode block decodes the inputs to this block and passes them out at the next clock cycle to the *execute* block. It is an additional block that takes up one clock cycle in the pipeline stage when instructions like ***ADD***, ***SUB***, ***MUL***, ***CJE***, or ***READ*** get into the pipeline. During these instructions, values from internal registers in *register file* block are to be passed to *execute* block in order to perform the operation.

Table 24 contains a description of the interface signals of *decode* block and Fig. 49 provides a diagram.

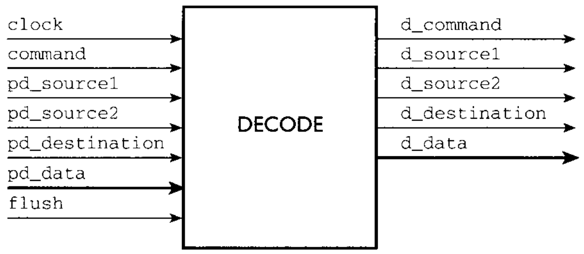


FIGURE 49 Diagram Showing the Interface Signal of Decode Block.

TABLE 24 Description of Decode Block Interface Signals

Signal Name	I/O	Description
clock	Input	Clock signal.
command	Input	Input of type *command_type* which can be any ***of MOVE***, ***ADD***, ***SUB***, ***MUL***, ***CJE***, ***LOAD***, ***READ*** and ***NOP***.
pd_source1	Input	Input of type *register_type*, which can be any of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*. It indicates the register to be used for <*source1*>.
pd_source2	Input	Input of type *register_type* which can be any of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*. It indicates the register to be used for <*source2*>.
pd_destination	Input	Input of type *register_type* which can be any of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*. It indicates the register to be used for <*destination*>.
pd_data	Input	32-bit bus, which is used to pass data between *predecode* block and *decode* block during a ***LOAD*** instruction.
flush	Input	When asserted high, *decode* block must go into flush mode. This happens when a branch occurs.
d_command	Output	Passing of instruction to *execute* block. It is of type *command_type*, which can be any of ***MOVE***, ***ADD***, ***SUB***, ***MUL***, ***CJE***, ***LOAD***, ***READ*** and ***NOP***.
d_destination	Output	Passes the information of ***pd_destination*** from *predecode* block to *execute* block.
d_source1	Output	Passes the information of ***pd_source1*** from *predecode* block to *execute* block.
d_source2	Output	Passes the information of ***pd_source2*** from *predecode* block to *execute* block.
d_data	Output	Passes the information of ***pd_data*** to *execute* block during a ***LOAD*** instruction.

EXAMPLE 42 Example of Decode Block Synthesizable VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY decode_ent IS
PORT (
        clock : IN std_logic;
        command : IN command_type;
        pd_source1 : IN register_type;
        pd_source2 : IN register_type;
        pd_destination : IN register_type;
        pd_data : IN std_logic_vector (31 downto 0);
        flush : IN std_logic;
        d_command : OUT command_type;
        d_destination : OUT register_type;
        d_source1 : OUT register_type;
        d_source2 : OUT register_type;
        d_data : OUT std_logic_vector (31 downto 0)
        );
END decode_ent;

ARCHITECTURE decode_arch OF decode_ent IS
BEGIN
        PROCESS (clock, command, pd_source1, pd_source2,
                pd_destination, flush, pd_data)
        BEGIN
            IF (clock = '1' AND clock'EVENT) THEN
                IF (flush = '0') THEN
                    CASE command IS
                        WHEN MOVE =>
```

Definition of actions to be taken for each instruction decoded.

```
                        -- MOVE <source1>, <destination>
                        -- <source2> is defaulted to reg0
                        -- d_data defaulted to all zero
                        -- since it is only used during
                        -- LOAD
                            d_source1 <= pd_source1;
                            d_source2 <= reg0;
                            d_destination <=
                            pd_destination;
                            d_data <= ZERO;
                        WHEN ADD =>
                -- ADD <source1>,<source2>,<destination>
                -- d_data default to all zeros
                            d_source1 <= pd_source1;
                            d_source2 <= pd_source2;
                            d_destination <=
                            pd_destination;
                            d_data <= ZERO;
                        WHEN SUB =>
                  -- SUB source1>,<source2>,<destination>
                  -- d_data default to ZERO
                            d_source1 <= pd_source1;
                            d_source2 <= pd_source2;
                            d_destination <=
                    pd_destination;
                    d_data <= ZERO;
                WHEN MUL =>
  -- MUL<source1>,<source2>,<destination>
  -- d_data again default to all zero
                    d_source1 <= pd_source1;
                    d_source2 <= pd_source2;
                    d_destination <=
                    pd_destination;
                    d_data <= ZERO;
                WHEN CJE =>
-- CJE <source1>,<source2>,<destination>
-- d_data default to all zero
                    d_source1 <= pd_source1;
                    d_source2 <= pd_source2;
                    d_destination <=
                    pd_destination;
                    d_data <= ZERO;
                WHEN LOAD =>
-- LOAD <value>, <destination>
-- d_data passes the data from predecode
-- block to execute block. <source1> and
-- <source2> are defaulted to reg0 they
-- are not used in this instruction.
                    d_data <= pd_data;
                    d_source1 <= reg0;
                    d_source2 <= reg0;
                    d_destination <=
                    pd_destination;
                WHEN READ =>
-- READ <destination>
--<source1> and <source2> defaults to
-- reg0 as they are not used.
-- d_data default to all zero.
                    d_source1 <= reg0;
                    d_source2 <= reg0;
                    d_destination <=
                    pd_destination;
                    d_data <= ZERO;
                WHEN NOP =>
-- no operation. all outputs to default
-- value.
                    d_source1 <= reg0;
                    d_source2 <= reg0;
                    d_destination <= reg0;
                    d_data <= ZERO;
```

```
                        WHEN OTHERS =>
                             NULL;
                   END CASE;
                   d_command <= command;
              ELSE
                   d_source1 <= reg0;
                   d_source2 <= reg0;
                   d_destination <= reg0;
                   d_data <= ZERO;
                   d_command <= NOP;
              END IF;
         END IF;
    END PROCESS;
END decode_arch;
```

When flush occurs, all operations default to ***NOP***.

Again, a testbench is written to simulate the design to ensure correct functionality.

EXAMPLE 43 Example of Decode Block Testbench

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY decode_tb_ent IS
END decode_tb_ent;

ARCHITECTURE decode_tb_arch OF decode_tb_ent IS
COMPONENT decode_ent
PORT (
       clock : IN std_logic;
       command : IN command_type;
       pd_source1 : IN register_type;
       pd_source2 : IN register_type;
       pd_destination : IN register_type;
       pd_data : IN std_logic_vector (31 downto 0);
       flush : IN std_logic;
       d_command : OUT command_type;
       d_destination : OUT register_type;
       d_source1 : OUT register_type;
       d_source2 : OUT register_type;
       d_data : OUT std_logic_vector (31 downto 0)
     );
END COMPONENT;

SIGNAL d_command : command_type;
SIGNAL d_destination : register_type;
SIGNAL d_source1 : register_type;
SIGNAL d_source2 : register_type;
SIGNAL pd_source1 : register_type;
SIGNAL pd_source2 : register_type;
SIGNAL clock : std_logic := '0';
SIGNAL pd_destination : register_type;
SIGNAL flush : std_logic;
SIGNAL command : command_type;
SIGNAL pd_data : std_logic_vector (31 downto 0);
SIGNAL d_data : std_logic_vector (31 downto 0);

CONSTANT CYCLE : TIME := 50 ns;

BEGIN

DUT: decode_ent
            port map(clock, command, pd_source1, pd_source2,
                    pd_destination, pd_data, flush, d_command,
                    d_destination, d_source1,
                    d_source2, d_data);

clock <= NOT clock AFTER CYCLE/2;

PROCESS
BEGIN

flush <= '0';
wait for CYCLE;

-- load, #10051, reg0
command <= LOAD;
pd_destination <= reg0;
pd_data <= "00000000000000010000000001010001";
wait for CYCLE;

-- load, #1, reg1
command <= LOAD;
pd_destination <= reg1;
pd_data <= "00000000000000000000000000000001";
wait for CYCLE;

-- add reg0, reg1, reg2
command <= ADD;
pd_source1 <= reg0;
pd_source2 <= reg1;
```

```
pd_destination <= reg2;
wait for CYCLE;

-- sub reg0, reg1, reg3
command <= SUB;
pd_source1 <= reg0;
pd_source2 <= reg1;
pd_destination <= reg3;
wait for CYCLE;

END PROCESS;
END decode_tb_arch;

CONFIGURATION decode_tb_config OF decode_tb_ent IS
  FOR decode_tb_arch
    FOR ALL : decode_ent
      USE ENTITY WORK.decode_ent(decode_arch);
    END FOR;
  END FOR;
END decode_tb_config;
```

Testbench of Example 43 simulates *decode* block with the following instructions:

```
LOAD, #10051, reg0
LOAD, #1, reg1
ADD reg0, reg1, reg2
SUB reg0, reg1, reg3
```

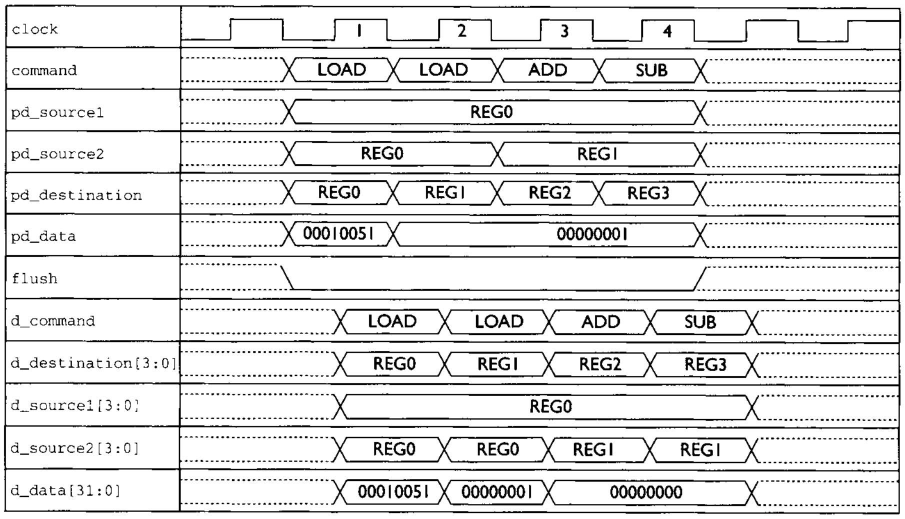


FIGURE 50 Timing Waveform for Decode Block Testbench.

From Fig. 50:

1. At the rising edge of 1st clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **d_command** drives **LOAD** since input command shows value **LOAD**.
   - **d_destination** drives value shown by **pd_destination** (*reg0*). **d_source1** drives value shown by **pd_source1** (*reg0*). **d_source2** drives value shown by **pd_source2** (*reg0*).
   - **d_data** drives value '00010051' since input bus **pd_data** shows value of '00010051'.
   - At the end of 1st clock, an instruction **LOAD** of value '00010051' into register *reg0* is decoded.
2. At the rising edge of 2nd clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **d_destination** drives value of *reg1* since input **pd_destination** changes to value *reg1*.
   - **d_data** changes to value '00000001' since input **pd_data** changes from value of '00010051' to '00000001'.
   - At the end of the 2nd clock, the instruction **LOAD** of data '00000001' into register *reg1* is decoded.
3. At the rising edge of 3th clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **d_command** drives **ADD** since input command is at value **ADD**.
   - **d_source1** drives value of *reg0* since input **pd_source1** has value of *reg0*.
   - **d_source2** drives value of *reg1* since input **pd_source2** has value of *reg1*.
   - Since this instruction is **ADD**, **d_data** drives value of '00000000'.
   - The instruction at this cycle decodes to **ADD** contents of *reg0* (designated by **pd_source1**) with the contents of *reg1* (as designated by **pd_source2**) and the result to be stored in *reg2* (designated by **pd_destination**).
4. At the rising edge of the 4th clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **d_command** drives value **SUB** (**command** has value of **SUB**).
   - **pd_source1** remains the same. **d_source1** shows no change.
   - **pd_source2** remains at *reg1*. **d_source2** also remains at *reg1*.
   - **d_destination** drives value of *reg3* since **pd_destination** changes to value *reg3*.
   - At the end of the 4th clock cycle, instruction decodes to subtraction of contents *reg1* (as designated by **pd_source2**) from contents of *reg0* (as designated by **pd_source1**) and the result stored in *reg3* (as designated by **pd_destination**).

6.4.3 Register File Block

This block is also part of the *decode* stage. It consists of sixteen 32-bit register designated `reg0` to `reg15`. *Register file* block interfaces with *predecode* block and *execute* block. It accepts signals from *predecode* block that indicate the need to read any of its registers and drive the data within those registers to *execute* block.

Register file block also receives signals from *execute* block when an instruction has been executed by *execute* block that needs to store data into any of the registers in *register file* block. See Table 25 and Fig. 51 for description and diagram, respectively.

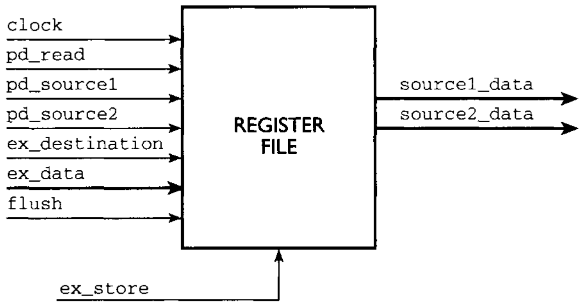


FIGURE 51 Diagram Showing the Interface Signals for the Register File Block.

TABLE 25 Description of Register File Block Interface Signals

Signal name	I/O	Description
clock	Input	Clock signal
flush	Input	When asserted to logical '1', *register file* block must default to ***NOP***.
pd_read	Input	When asserted to logical '1', the instruction being passed to *execute* block is either a ***MOVE***, ***ADD***, ***SUB***, ***MUL***, ***CJE*** or ***READ***. When this occurs, *register file* block must decode **pd_source1** and **pd_source2** inputs. The contents of the registers indicated by **pd_source1** and **pd_source2** are driven on **source1_data** and **source2_data** output.
pd_source1	Input	The contents of register designated by **pd_source1** are to be used as data for `<source1>`.
pd_source2	Input	The contents of register designated by **pd_source2** are to be used as data for `<source2>`.
ex_store	Input	When asserted to logical '1', *register file* block must store the data on **ex_data** bus into register indicated by **ex_destination**.
ex_data	Input	Transfer data from *execute* block to *register file* block for storage.

continued

TABLE 25 ***continued***

Signal name	I/O	Description
ex_destination	Input	Indication of register to be used in *register file* block as a destination for storing of data on ***ex_data*** bus when ***ex_store*** is asserted to logical '1'.
source1_data	Output	Output of contents of *<source1>*.
source2_data	Output	Output of contents of *<source2>*.

EXAMPLE 44 Example of Register File Block Synthesizable VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY register_file_ent IS
PORT (
        pd_read : IN std_logic;
        pd_source1 : IN register_type;
        pd_source2 : IN register_type;
        clock : IN std_logic;
        flush : IN std_logic;
        ex_store : IN std_logic;
        ex_destination : IN register_type;
        ex_data : IN std_logic_vector (31 downto 0);
        source1_data : OUT std_logic_vector (31 downto 0);
        source2_data : OUT std_logic_vector (31 downto 0)
        );
END register_file_ent;

ARCHITECTURE register_file_arch OF register_file_ent IS
SIGNAL reg : array_size;
BEGIN
        PROCESS (clock, flush, pd_read, pd_source1, pd_source2)
        BEGIN
             IF (clock = '1' AND clock'EVENT) THEN
                  IF (flush = '0') THEN
                       IF (pd_read = '1') THEN
                            IF (pd_source1 /= ex_destination)
                            THEN
                                 CASE pd_source1 IS
                                      WHEN reg0 =>
                                              source1_data <=
                                              reg(0);
                                      WHEN reg1 =>
```

```
                source1_data <=
                reg(1);
                WHEN reg2 =>
              source1_data <=
                reg(2);
      WHEN reg3 =>
              source1_data <=
                reg(3);
                WHEN reg4 =>
                source1_data <=
                reg(4);
      WHEN reg5 =>
                source1_data <=
                reg(5);
      WHEN reg6 =>
                source1_data <=
                reg(6);
      WHEN reg7 =>
                source1_data <=
                reg(7);
      WHEN reg8 =>
                source1_data <=
                reg(8);
      WHEN reg9 =>
                source1_data <=
                reg(9);
            WHEN reg10 =>
               source1_data <=
                reg(10);
      WHEN reg11 =>
              source1_data <=
                reg(11);
            WHEN reg12 =>
                source1_data <=
                reg(12);
      WHEN reg13 =>
                source1_data <=
                reg(13);
      WHEN reg14 =>
                source1_data <=
                reg(14);
      WHEN reg15 =>
                source1_data <=
                reg(15);
      WHEN OTHERS =>
                NULL;
END CASE;
```

Reading contents of register indicated by ***pd_source1***. Register contents are driven on ***source1_data*** output.

Reading values of register indicated by ***pd_source2***. Register contents are driven on ***source2_data*** output.

```
ELSE
     source1_data <= ex_data;
END IF;
IF (pd_source2 /= ex_destination)
THEN
     CASE pd_source2 IS
          WHEN reg0 =>
                 source2_data <=
                 reg(0);
          WHEN reg1 =>
                 source2_data <=
                 reg(1);
           WHEN reg2 =>
                source2_data <=
                 reg(2);
           WHEN reg3 =>
                source2_data <=
                 reg(3);
           WHEN reg4 =>
                 source2_data <=
                 reg(4);
          WHEN reg5 =>
                 source2_data <=
                 reg(5);
          WHEN reg6 =>
                 source2_data <=
                 reg(6);
          WHEN reg7 =>
                 source2_data <=
                 reg(7);
          WHEN reg8 =>
                 source2_data <=
                 reg(8);
          WHEN reg9 =>
                 source2_data <=
                 reg(9);
           WHEN reg10 =>
                source2_data <=
                 reg(10);
           WHEN reg11 =>
                source2_data <=
                 reg(11);
          WHEN reg12 =>
                 source2_data <=
                 reg(12);
          WHEN reg13 =>
```

```
                              source2_data <=
                              reg(13);
                       WHEN reg14 =>
                              source2_data <=
                              reg(14);
                       WHEN reg15 =>
                              source2_data <=
                              reg(15);
                       WHEN OTHERS =>
                              NULL;
                  END CASE;
             ELSE
                  source2_data <= ex_data;
             END IF;
        END IF;
        IF (ex_store = '1') THEN
             CASE ex_destination IS
                  WHEN reg0 =>
                       reg(0) <= ex_data;
                  WHEN reg1 =>
                       reg(1) <= ex_data;
                  WHEN reg2 =>
                       reg(2) <= ex_data;
                  WHEN reg3 =>
                       reg(3) <= ex_data;
                  WHEN reg4 =>
                       reg(4) <= ex_data;
                  WHEN reg5 =>
                       reg(5) <= ex_data;
                  WHEN reg6 =>
                       reg(6) <= ex_data;
                  WHEN reg7 =>
                       reg(7) <= ex_data;
                  WHEN reg8 =>
                       reg(8) <= ex_data;
                  WHEN reg9 =>
                       reg(9) <= ex_data;
                  WHEN reg10 =>
                       reg(10) <= ex_data;
                  WHEN reg11 =>
                       reg(11) <= ex_data;
                  WHEN reg12 =>
                       reg(12) <= ex_data;
                  WHEN reg13 =>
                       reg(13) <= ex_data;
                  WHEN reg14 =>
```

When **ex_store** is asserted to logical '1', data on **ex_data** bus are written into register indicated by **ex_destination**.

CASE statement is used to infer multiplexers. **IF** statement is not used as it would infer priority encoders.

```
                          reg(14) <= ex_data;
                     WHEN reg15 =>
                          reg(15) <= ex_data;
                     WHEN OTHERS =>
                          NULL;
                END CASE;
           END IF;
      ELSE
           source1_data <= ZERO;
           source2_data <= ZERO;
      END IF;
 END IF;
END PROCESS;

END register_file_arch;
```

source1_data and ***source2_data*** default to zero when flushing.

A testbench is written to simulate the code for the *register file* block.

EXAMPLE 45 Example of Register File Synthesizable VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY register_file_tb_ent IS
END register_file_tb_ent;

ARCHITECTURE register_file_tb_arch OF register_file_tb_ent IS
COMPONENT register_file_ent
PORT (
     pd_read : IN std_logic;
     pd_source1 : IN register_type;
     pd_source2 : IN register_type;
     clock : IN std_logic;
     flush : IN std_logic;
     ex_store : IN std_logic;
     ex_destination : IN register_type;
     ex_data : IN std_logic_vector (31 downto 0);
     source1_data : OUT std_logic_vector (31 downto 0);
     source2_data : OUT std_logic_vector (31 downto 0)
     );
END COMPONENT;

SIGNAL pd_read : std_logic;
SIGNAL pd_source1 : register_type;
SIGNAL pd_source2 : register_type;
SIGNAL clock : std_logic := '0';
SIGNAL flush : std_logic;
SIGNAL ex_store : std_logic;
SIGNAL ex_destination : register_type;
SIGNAL ex_data : std_logic_vector (31 downto 0);
SIGNAL source1_data : std_logic_vector (31 downto 0);
SIGNAL source2_data : std_logic_vector (31 downto 0);

CONSTANT CYCLE : TIME := 50 ns;

BEGIN

DUT: register_file_ent port map( pd_read, pd_source1, pd_source2,
                              clock, flush, ex_store,
                              ex_destination, ex_data,
                              source1_data, source2_data);

clock <= NOT clock AFTER CYCLE/2;

PROCESS
BEGIN

-- set default to 0
flush <= '0';
pd_read <= '0';
pd_source1 <= reg0;
pd_source2 <= reg0;
ex_store <= '0';
ex_destination <= reg0;
ex_data <= ZERO;
wait for CYCLE;

-- load, #10051, reg0
ex_store <= '1';
ex_destination <= reg0;
ex_data <= "00000000000000010000000001010001";
wait for CYCLE;

-- load, #1, reg1
ex_store <= '1';
ex_destination <= reg1;
ex_data <= "00000000000000000000000000000001";
wait for CYCLE;

-- add reg0, reg1, reg2
-- reading values of reg0 and reg1
```

```
ex_store <= '0';
pd_read <= '1';
pd_source1 <= reg0;
pd_source2 <= reg1;
wait for CYCLE;
```

-- sub reg0, reg1, reg3
-- reading values of reg0 and reg1

```
ex_store <= '0';
pd_read <= '1';
pd_source1 <= reg0;
pd_source2 <= reg1;
wait for CYCLE;

END PROCESS;
END register_file_tb_arch;

CONFIGURATION register_file_tb_config OF register_file_tb_ent IS
  FOR register_file_tb_arch
    FOR ALL : register_file_ent
      USE ENTITY WORK.register_file_ent(register_file_arch);
    END FOR;
  END FOR;
END register_file_tb_config;
```

The timing waveform for the simulation results of Example 45 is shown in Fig. 52.

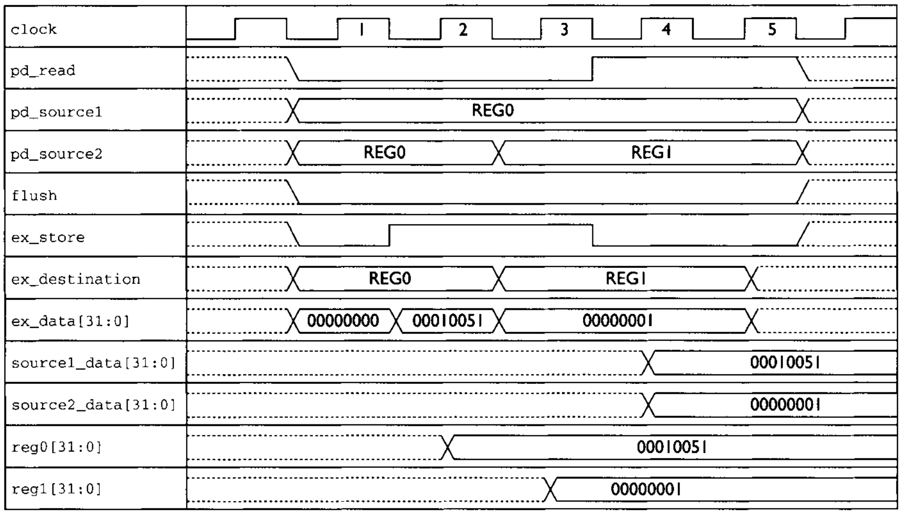


FIGURE 52 Timing Diagram Showing the Simulation for the Register File Testbench.

From Fig. 52:

1. At the rising edge of the 1st clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **pd_read** is at logical '0'. Reading of register contents is not required.
   - Input value of **pd_source1** and **pd_source2** is ignored since **pd_read** is at logical '0'.
   - **ex_store** is at logical '0'. Writing of data into registers is not required.
2. At the rising edge of the 2nd clock,
   - **flush** is at logical '0'. No flushing should occur.
   - **ex_store** is at logical '1'. **ex_data** shows value of '00010051'. **ex_destination** shows value of *reg0*. This is decoded to storing of data '00010051' from **ex_data** bus into register *reg0* as indicated by **ex_destination**.
   - At the next rising edge of clock, the content of *reg0* changes to '00010051'.
3. At the rising edge of the 3rd clock,
   - **ex_store** is at logical '1'. **ex_destination** shows value of *reg1*. **ex_data** shows value of '00000001'.
   - This is decoded to an event of storing data '00000001' from **ex_data** bus into register *reg1* as designated by **ex_destination**.
   - At the next rising edge of clock, the content of *reg1* changes to '00000001'.
4. At the rising edge of the 4th clock,
   - **ex_store** is at logical '0'.
   - **pd_read** is at logical '1'. **pd_source1** and **pd_source2** show value of *reg0* and *reg1*. This decodes to reading of *reg0* and *reg1* and driving the contents of those registers on **source1_data** and **source2_data** output.

6.4.4 Execute Block

This block represents the *execute* stage of the pipeline. This is also the block that executes the instructions. For example, the contents of *<source1>* and *<source2>* are added in this block for an ***ADD*** instruction. The results are sent to *register file* block and stored into the register designated by *<destination>*.

TABLE 26 Description of Execute Block Interface Signals

Signal Name	I/O	Description
d_command	Input	Command line input to indicate which instruction is to be executed. It is of type *command_type* and can be any of ***MOVE***, ***ADD***, ***SUB***, ***MUL***, ***CJE***, ***LOAD***, ***READ*** or ***NOP***.
d_source1_data	Input	32 bits input to pass data from *register file* block to *execute* block. The data on this bus are only valid for ***ADD***, ***SUB***, ***MUL*** and ***CJE*** where contents of registers in *register file* block are read and passed to *execute* block.

continued

TABLE 26 Description of Execute Block Interface Signals

Signal Name	I/O	Description
d_source2_data	Input	32 bits input to pass data from *register file* block to *execute* block. The data on this bus are only valid for ***ADD***, ***SUB***, ***MUL***, and ***CJE*** where contents of registers in *register file* block are read and passed to *execute* block.
d_destination	Input	This signal is of type *register_type* and can be any one of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*.
d_source1	Input	This signal is of type *register_type* and can be any one of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*.
d_source2	Input	This signal is of type *register_type* and can be any one of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*.
d_data	Input	32 bits data bus that passes data from *decode* block to *execute* block. It is only valid during ***LOAD*** instruction.
clock	Input	Clock signal.
flush	Output	Asserted to logical '1' when a branch occurs. When other blocks see this signal going at logical '1', they must flush all existing instructions in the pipeline
jump	Output	Asserted to logical '1' when a branch occurs. It is used as an indicator to the external *instruction module* that a branch is to occur.
ex_data	Output	32-bit-data bus that passes results of the arithmetic operation performed by *execute* block to *register file* block for storage.
ex_store	Output	Asserted to logical '1' for indication to *register file* block to store data on ***ex_data*** bus into register designated by ***ex_destination***.
ex_destination	Output	This signal is of type *register_type* and can be any of *reg0*, *reg1*, *reg2*, *reg3*, *reg4*, *reg5*, *reg6*, *reg7*, *reg8*, *reg9*, *reg10*, *reg11*, *reg12*, *reg13*, *reg14* or *reg15*. It indicates to *register file* block on register to use for storage of data from *ex_data* bus.
output	Output	This is the output of *execute* block. During a ***READ*** *<destination>* instruction, the contents of register designated by *<destination>* are driven on this output bus.

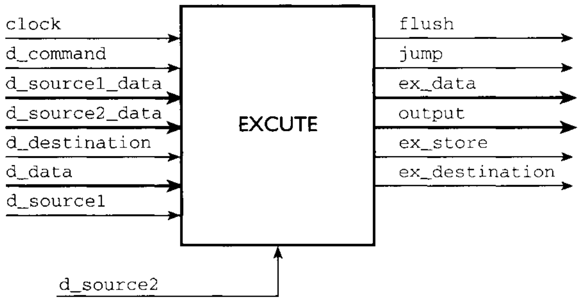


FIGURE 53 Diagram Showing the Interface Signal of Execute Block

EXAMPLE 46 Example of Execute Block Synthesizable VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.pipeline_package.ALL;

ENTITY execute_ent IS
PORT (
       d_command : IN command_type;
       d_source1_data : IN std_logic_vector (31 downto 0);
       d_source2_data : IN std_logic_vector (31 downto 0);
       d_destination : IN register_type;
       d_source1 : IN register_type;
       d_source2 : IN register_type;
       d_data : IN std_logic_vector (31 downto 0);
       clock : IN std_logic;
       flush : OUT std_logic;
       jump : OUT std_logic;
       ex_data : OUT std_logic_vector (31 downto 0);
       ex_destination : OUT register_type;
       ex_store : OUT std_logic;
       output : OUT std_logic_vector (31 downto 0)
       );
END execute_ent;

ARCHITECTURE execute_arch OF execute_ent IS

SIGNAL int_ex_destination : register_type := reg0;
SIGNAL int_ex_data : std_logic_vector (31 downto 0);
SIGNAL int_d_source1_data : std_logic_vector (31 downto 0);
SIGNAL int_d_source2_data : std_logic_vector (31 downto 0);
```

Refer to Note on p. 126

```
BEGIN
    PROCESS (d_command, int_ex_destination, d_source1,
            d_source2, int_d_source1_data,
            int_d_source2_data, d_source1_data, d_source2_data,
            int_ex_data)
    BEGIN
        IF (d_command = LOAD OR d_command = MOVE OR d_command
            = NOP) THEN
            int_d_source1_data <= d_source1_data;
            int_d_source2_data <= d_source2_data;
        ELSE
            IF (int_ex_destination = d_source1) THEN
                int_d_source1_data <= int_ex_data;
                int_d_source2_data <= d_source2_data;
            ELSIF (int_ex_destination = d_source2) THEN
                int_d_source2_data <= int_ex_data;
                int_d_source1_data <= d_source1_data;
            ELSE
                int_d_source1_data <= d_source1_data;
                int_d_source2_data <= d_source2_data;
            END IF;
        END IF;
    END PROCESS;

    PROCESS (clock, d_command, int_d_source1_data,
            int_d_source2_data, d_destination, d_source1,
            d_source2, int_ex_data, d_data)
    BEGIN
        IF (clock = '1' AND clock'EVENT) THEN
            CASE d_command IS
                WHEN MOVE =>
                    int_ex_data <= int_d_source1_data;
                    int_ex_destination <=
                    d_destination;
                    ex_store <= '1';
                    jump <= '0';
                    output <= ZERO;
                    flush <= '0';
                WHEN ADD =>
                    -- both MSB of src1 and src2 cannot
                    -- be '1'. if it is then overflow.
                    int_ex_data <=
                    signed(int_d_source1_data) +
                    signed(int_d_source2_data);
                    int_ex_destination <=
                    d_destination;
```

```
          ex_store <= '1';
          jump <= '0';
          flush <= '0';
          output <= ZERO;
     WHEN SUB =>
          int_ex_data <= signed(int_d_source1_data)-
          signed(int_d_source2_data);
          int_ex_destination <=
          d_destination;
          ex_store <= '1';
          jump <= '0';
          flush <= '0';
          output <= ZERO;
     WHEN MUL =>
          -- both src1 and src2 must be max
          -- 16 bits long
          -- if not will overflow
          -- excess of 16 bits is truncated
          int_ex_data <=
          signed(int_d_source1_data ( 15
                    downto 0)) *
          signed(int_d_source2_data ( 15
                    downto 0));
          int_ex_destination <=
          d_destination;
          ex_store <= '1';
          jump <= '0';
          flush <= '0';
          output <= ZERO;
     WHEN CJE =>
          IF (int_d_source1_data =
          int_d_source2_data) THEN
               jump <= '1';
               flush <= '1';
          ELSE
               jump <= '0';
               flush <= '0';
          END IF;
          ex_store <= '0';
          output <= ZERO;
     WHEN LOAD =>
          int_ex_destination <=
          d_destination;
          int_ex_data <= d_data;
          ex_store <= '1';
          output <= ZERO;
```

Signal assignment for each instruction

```
                    jump <= '0';
                    flush <= '0';
                WHEN READ =>
                    output <= int_d_source1_data;
                    ex_store <= '0';
                    flush <= '0';
                    jump <= '0';
                WHEN NOP =>
                    output <= ZERO;
                    flush <= '0';
                    jump <= '0';
                    ex_store <= '0';
                WHEN OTHERS =>
                    NULL;
            END CASE;
        END IF;
    END PROCESS;

    ex_data <= int_ex_data;
    ex_destination <= int_ex_destination;

END execute_arch;
```

> ***Note:*** From *execute* block code, the first process checks for instruction signal ***d_command*** to ensure that it is of type ***READ***, ***ADD***, ***SUB***, ***MUL*** or ***CJE***. This checking is done for implementation of a feature called '***execute bypassing***' that is common to most pipeline designs.
>
> If the instruction going into *execute* block is ***READ***, ***ADD***, ***SUB***, ***MUL*** or ***CJE***, the inputs to the multiplier/adder/subtractor/comparator must be bypassed. In other words, the output of *execute* block must be used for the input and not the output of *register file* block.

Execute bypassing is a pipeline solution when two consecutive instructions executed by a pipeline design are dependent on each other.

For example, let us say we have the following instructions:

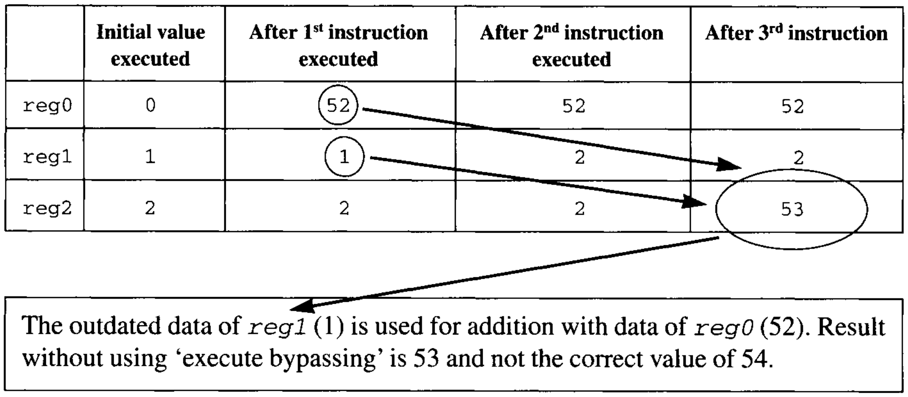
Both instructions having dependence on register *reg1*.

```
Inst (1) LOAD #52, reg0
Inst (2) LOAD #02, reg1
Inst (3) ADD reg0, reg1, reg2
```

At the end of instruction (2), before *execute* block can complete this instruction and write the data #02 into `reg1` in *register file* block, the data needed for the execution of instruction (2) have already been passed to *execute* block. Therefore the data of `reg1` passed to *execute* block on the third instruction are the contents of `reg1` before *execute* block is able to store the value of #02 into `reg1` register. In other words, the third instruction will perform an addition on the wrong contents of `reg1`.

The solution to this problem is to use a concept called 'execute bypassing'. In this concept, *execute* block is 'informed' in advance that there is dependence on the second and third instruction. It will therefore use the contents of output **ex_data** and not the 'outdated' data from *register file* block.

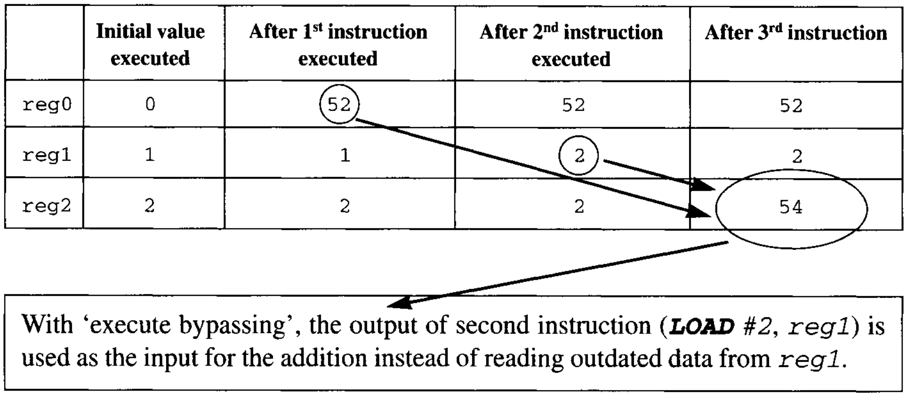
Assuming the initial contents of *reg0* = 0, *reg1* = 1 and *reg2* = 2 (without 'execute bypassing'):



	Initial value executed	After 1st instruction executed	After 2nd instruction executed	After 3rd instruction
reg0	0	52	52	52
reg1	1	1	2	2
reg2	2	2	2	53

The outdated data of `reg1` (1) is used for addition with data of `reg0` (52). Result without using 'execute bypassing' is 53 and not the correct value of 54.

Now the same instructions go through the pipeline but this time with 'execute bypassing':



	Initial value executed	After 1st instruction executed	After 2nd instruction executed	After 3rd instruction
reg0	0	52	52	52
reg1	1	1	2	2
reg2	2	2	2	54

With 'execute bypassing', the output of second instruction (***LOAD*** #2, `reg1`) is used as the input for the addition instead of reading outdated data from `reg1`.

The first process of Example 46 describes the functionality required to perform 'execute bypassing'.

Figure 54 is a block diagram showing the interface signals for *execute* block.

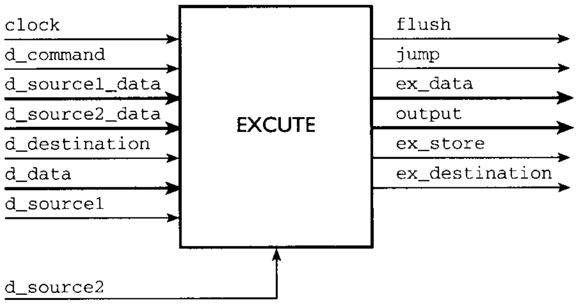


FIGURE 54 Diagram Showing Interface Signals for Execute Block.

A testbench is written to check the functionality of the code for *execute* block as shown in Example 46.

EXAMPLE 47 Example of Execute Block Testbench for Functionality Check

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY execute_tb_ent IS
END execute_tb_ent;

ARCHITECTURE execute_tb_arch OF execute_tb_ent IS
COMPONENT execute_ent
PORT (
     d_command : IN command_type;
     d_source1_data : IN std_logic_vector (31 downto 0);
     d_source2_data : IN std_logic_vector (31 downto 0);
     d_destination : IN register_type;
     d_source1 : IN register_type;
     d_source2 : IN register_type;
     d_data : IN std_logic_vector (31 downto 0);
     clock : IN std_logic;
     flush : OUT std_logic;
     jump : OUT std_logic;
     ex_data : OUT std_logic_vector (31 downto 0);
     ex_destination : OUT register_type;
     ex_store : OUT std_logic;
     output : OUT std_logic_vector (31 downto 0)
     );
END COMPONENT;
```

```
SIGNAL ex_data : std_logic_vector (31 downto 0);
SIGNAL ex_destination : register_type;
SIGNAL ex_store : std_logic;
SIGNAL d_command : command_type;
SIGNAL d_source1_data : std_logic_vector (31 downto 0);
SIGNAL d_source2_data : std_logic_vector (31 downto 0);
SIGNAL d_destination : register_type;
SIGNAL d_source1 : register_type;
SIGNAL d_source2 : register_type;
SIGNAL clock : std_logic := '0';
SIGNAL flush : std_logic;
SIGNAL jump : std_logic;
SIGNAL output : std_logic_vector (31 downto 0);
SIGNAL d_data : std_logic_vector (31 downto 0);

CONSTANT CYCLE : TIME := 50 ns;

BEGIN

DUT: execute_ent port map(d_command, d_source1_data,
                         d_source2_data, d_destination,
                         d_source1, d_source2, d_data, clock,
                         flush, jump, ex_data, ex_destination,
                         ex_store, output);

clock <= NOT clock AFTER CYCLE/2;

PROCESS
BEGIN

wait for CYCLE;

-- load, #51, reg0

d_command <= LOAD;
d_destination <= reg0;
d_data <= "00000000000000000000000000110011";
wait for CYCLE;

-- load, #1, reg1
d_command <= LOAD;
d_destination <= reg1;
d_data <= "00000000000000000000000000000001";
wait for CYCLE;

-- add reg0, reg1, reg2
```

```
d_command <= ADD;
d_source1 <= reg0;
d_source2 <= reg1;
d_destination <= reg2;
d_source1_data <= "00000000000000000000000001010001";
d_source2_data <= "00000000000000000000000000000001";
wait for CYCLE;

-- sub reg0, reg1, reg3
d_command <= SUB;
d_source1 <= reg0;
d_source2 <= reg1;
d_destination <= reg3;
d_source1_data <= "00000000000000000000000001010001";
d_source2_data <= "00000000000000000000000000000001";
wait for CYCLE;

END PROCESS;
END execute_tb_arch;

CONFIGURATION execute_tb_config OF execute_tb_ent IS
  FOR execute_tb_arch
    FOR ALL : execute_ent
      USE ENTITY WORK.execute_ent(execute_arch);
    END FOR;
  END FOR;
END execute_tb_config;
```

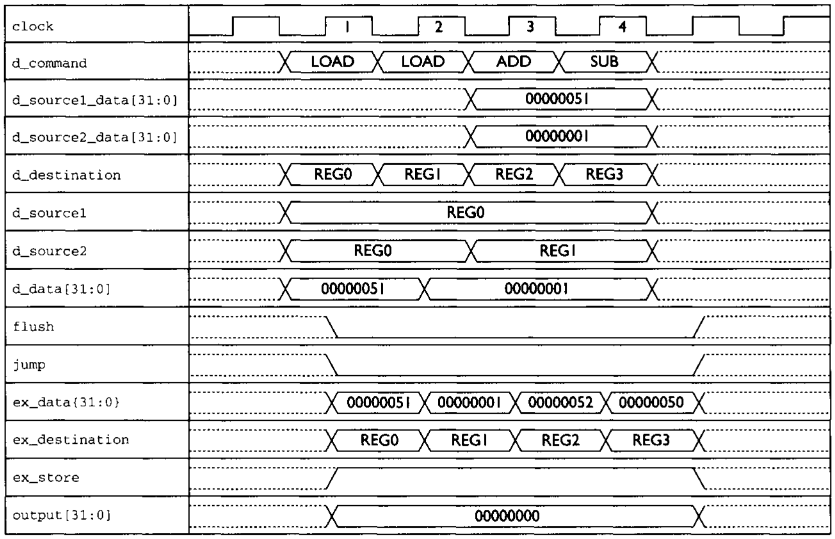


FIGURE 55 Timing Waveform Showing the Testbench for Execute Block.

From Fig. 55:

1. At the rising edge of the 1st clock,
   - ***flush*** is asserted to logical `'0'`. No flushing should occur.
   - ***jump*** is asserted to logical `'0'` as the instruction at ***d_command*** is ***LOAD***.
   - ***output*** is driven with value `'00000000'` as the instruction being executed is ***LOAD*** and not ***READ***.
   - ***d_destination*** has a value of *reg0*. Since this is a ***LOAD*** command, ***ex_store*** is asserted to logical `'1'` and ***ex_data*** is driven with value from input bus ***d_data*** (`'00000051'`).
   - ***ex_destination*** is asserted with value *reg0* because ***d_destination*** shows value of *reg0*. *Register file* block will use the value on ***ex_destination*** to determine in which register to store ***ex_data***.
   - This decodes to storing of data `'00000051'` (indicated by ***d_data***) into *reg0* as designated by ***d_destination***.
2. At the rising edge of the 2nd clock,
   - ***flush*** is asserted to logical `'0'`. No flushing should occur.
   - ***jump*** is asserted to logical `'0'` since instruction at ***d_command*** is ***LOAD***.
   - ***ex_store*** is asserted to logical `'1'`. ***ex_data*** asserts value of '00000001' which is the value at input ***d_data***.
   - ***ex_destination*** asserts value *reg1* because ***d_destination*** is showing value *reg1*.
   - This decodes to ***LOAD*** instruction of `'00000001'` (as indicated by ***d_data*** bus) into `reg1` (as designated by ***d_destination***).
3. At the rising edge of the 3rd clock,
   - ***flush*** is asserted to logical `'0'`. No flushing should occur.
   - ***d_command*** shows value ***ADD***. ***d_source1*** shows *reg0* and ***d_source2*** shows *reg1* while ***d_destination*** shows *reg2*. This decodes to an addition of *reg0* and *reg1*, with the result stored into *reg2*.
   - ***ex_store*** is asserted to logical `'1'`. ***ex_data*** drives value of `'00000052'` (`'00000051'` + `'00000001'`) on ***ex_data*** bus.
   - ***ex_destination*** drives value *reg2* to indicate that the result is to be stored in register *reg2*.
4. At the rising edge of the 4th clock,
   - ***flush*** is asserted to logical `'0'`. No flushing should occur.
   - ***d_command*** shows value of ***SUB***, which indicates that a ***SUB*** command is to be executed.
   - ***d_source1*** shows value of *reg0* and ***d_source2*** shows value of *reg1* while ***d_destination*** shows value of *reg3*.
   - This decodes to subtraction of *reg1* (as indicated by ***d_source2***) from *reg0* (as indicated by ***d_source1***) and the results of the operation to be stored in *reg3*.
   - ***ex_store*** is asserted to logical `'1'` while ***ex_data*** drives `'00000050'` (`'00000052'` – `'00000001'`) on ***ex_data*** bus.
   - ***ex_destination*** drives value *reg3*, which indicates to *register file* block that the data on ***ex_data*** bus is to be stored into register *reg3*.

6.4.5 Fullchip Microcontroller

In order to obtain fullchip for the pipeline microcontroller, the four different blocks are instantiated and connected as shown in Fig. 46.

EXAMPLE 48 Synthesizable Code of Fullchip Microcontroller

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.pipeline_package.ALL;

ENTITY microc_ent IS
PORT (
        clock : IN std_logic;
        inst : IN std_logic_vector (2 downto 0);
        source1 : IN std_logic_vector (3 downto 0);
        source2 : IN std_logic_vector (3 downto 0);
        destination : IN std_logic_vector (3 downto 0);
        data : IN std_logic_vector (31 downto 0);
        jump : OUT std_logic;
        output : OUT std_logic_vector (31 downto 0)
        );
END microc_ent;

ARCHITECTURE microc_arch OF microc_ent IS

COMPONENT execute_ent
PORT (
        d_command : IN command_type;
        d_source1_data : IN std_logic_vector (31 downto 0);
        d_source2_data : IN std_logic_vector (31 downto 0);
        d_destination : IN register_type;
        d_source1 : IN register_type;
        d_source2 : IN register_type;
        d_data : IN std_logic_vector (31 downto 0);
        clock : IN std_logic;
        flush : OUT std_logic;
        jump : OUT std_logic;
        ex_data : OUT std_logic_vector (31 downto 0);
        ex_destination : OUT register_type;
        ex_store : OUT std_logic;
        output : OUT std_logic_vector (31 downto 0)
        );
END COMPONENT;
```

```
COMPONENT register_file_ent
PORT (
        pd_read : IN std_logic;
        pd_source1 : IN register_type;
        pd_source2 : IN register_type;
        clock : IN std_logic;
        flush : IN std_logic;
        ex_store : IN std_logic;
        ex_destination : IN register_type;
        ex_data : IN std_logic_vector (31 downto 0);
        source1_data : OUT std_logic_vector (31 downto 0);
        source2_data : OUT std_logic_vector (31 downto 0)
        );
END COMPONENT;

COMPONENT decode_ent
PORT (
        clock : IN std_logic;
        command : IN command_type;
        pd_source1 : IN register_type;
        pd_source2 : IN register_type;
        pd_destination : IN register_type;
        pd_data : IN std_logic_vector (31 downto 0);
        flush : IN std_logic;
        d_command : OUT command_type;
        d_destination : OUT register_type;
        d_source1 : OUT register_type;
        d_source2 : OUT register_type;
        d_data : OUT std_logic_vector (31 downto 0)
        );
END COMPONENT;

COMPONENT predecode_ent
PORT (
        clock : IN std_logic;
        inst : IN std_logic_vector (2 downto 0);
        source1 : IN std_logic_vector (3 downto 0);
        source2 : IN std_logic_vector (3 downto 0);
        destination : IN std_logic_vector (3 downto 0);
        data : IN std_logic_vector (31 downto 0);
        flush : IN std_logic;
        command : OUT command_type;
        pd_source1 : OUT register_type;
        pd_source2 : OUT register_type;
        pd_destination : OUT register_type;
        pd_data : OUT std_logic_vector (31 downto 0);
        pd_read : OUT std_logic
        );
END COMPONENT;

SIGNAL sig_clock : std_logic;
SIGNAL sig_inst : std_logic_vector (2 downto 0);
SIGNAL sig_source1 : std_logic_vector (3 downto 0);
SIGNAL sig_source2 : std_logic_vector (3 downto 0);
SIGNAL sig_destination : std_logic_vector (3 downto 0);
SIGNAL sig_data : std_logic_vector (31 downto 0);
SIGNAL sig_jump : std_logic;
SIGNAL sig_output : std_logic_vector (31 downto 0);
SIGNAL sig_flush : std_logic;
SIGNAL sig_command : command_type;
SIGNAL sig_pd_source1 : register_type;
SIGNAL sig_pd_source2 : register_type;
SIGNAL sig_pd_destination : register_type;
SIGNAL sig_pd_data : std_logic_vector (31 downto 0);
SIGNAL sig_ex_data : std_logic_vector (31 downto 0);
SIGNAL sig_ex_destination : register_type;
SIGNAL sig_ex_store : std_logic;
SIGNAL sig_d_command : command_type;
SIGNAL sig_d_source1_data : std_logic_vector (31 downto 0);
SIGNAL sig_d_source2_data : std_logic_vector (31 downto 0);
SIGNAL sig_d_destination : register_type;
SIGNAL sig_d_source1 : register_type;
SIGNAL sig_d_source2 : register_type;
SIGNAL sig_pd_read : std_logic;
SIGNAL sig_d_data : std_logic_vector (31 downto 0);
BEGIN

DUT_predecode: predecode_ent PORT MAP  (
                           clock => sig_clock,
                           inst => sig_inst,
                           source1 => sig_source1,
                           source2 => sig_source2,
                           destination => sig_destination,
                            data => sig_data,
                           flush => sig_flush,
                           command => sig_command,
                           pd_source1 => sig_pd_source1,
                           pd_source2 => sig_pd_source2,
                            pd_destination =>
                           sig_pd_destination,
                           pd_data => sig_pd_data,
                           pd_read => sig_pd_read);
```

```
DUT_decode: decode_ent PORT MAP (
                           clock => sig_clock,
                             command => sig_command,
                           pd_source1 => sig_pd_source1,
                           pd_source2 => sig_pd_source2,
                           pd_destination =>
                           sig_pd_destination,
                           pd_data => sig_pd_data,
                           flush => sig_flush,
                           d_command => sig_d_command,
                           d_destination => sig_d_destination,
                           d_source1 => sig_d_source1,
                           d_source2 => sig_d_source2,
                           d_data => sig_d_data);

DUT_register_file: register_file_ent PORT MAP (
                           pd_read => sig_pd_read,
                           pd_source1 => sig_pd_source1,
                           pd_source2 => sig_pd_source2,
                           clock => sig_clock,
                            flush => sig_flush,
                           ex_store => sig_ex_store,
                           ex_destination =>
                           sig_ex_destination,
                           ex_data => sig_ex_data,
                           source1_data => sig_d_source1_data,
                           source2_data =>
                           sig_d_source2_data);

DUT_execute: execute_ent PORT MAP (
                           d_command => sig_d_command,
                             d_source1_data =>
                           sig_d_source1_data,
                           d_source2_data =>
                           sig_d_source2_data,
                           d_destination => sig_d_destination,
                           d_source1 => sig_d_source1,
                           d_source2 => sig_d_source2,
                           d_data => sig_d_data,
                           clock => sig_clock,
                           flush => sig_flush,
                           jump => sig_jump,
                           ex_data => sig_ex_data,
                           ex_destination =>
                           sig_ex_destination,
                           ex_store => sig_ex_store,
                           output => sig_output);
```

```
sig_clock <= clock;
sig_inst <= inst;
sig_source1 <= source1;
sig_source2 <= source2;
sig_destination <= destination;
sig_data <= data;
jump <= sig_jump;
output <= sig_output;

END microc_arch;

CONFIGURATION microc_config OF microc_ent IS
FOR microc_arch
        FOR ALL: predecode_ent
             USE ENTITY WORK.predecode_ent(predecode_arch);
        END FOR;
        FOR ALL: decode_ent
             USE ENTITY WORK.decode_ent(decode_arch);
        END FOR;
        FOR ALL: register_file_ent
             USE ENTITY
WORK.register_file_ent(register_file_arch);
        END FOR;
        FOR ALL: execute_ent
             USE ENTITY WORK.execute_ent(execute_arch);
        END FOR;
END FOR;
END microc_config;
```

A testbench is written to check the functionality of the microcontroller.

EXAMPLE 49 Example of Microcontroller Testbench

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY microc_tb_ent IS
PORT (
        clock : IN std_logic;
        inst : IN std_logic_vector (2 downto 0);
        source1 : IN std_logic_vector (3 downto 0);
        source2 : IN std_logic_vector (3 downto 0);
        destination : IN std_logic_vector (3 downto 0);
        data : IN std_logic_vector (31 downto 0);
        jump : OUT std_logic;
        output : OUT std_logic_vector (31 downto 0)
        );
END microc_tb_ent;

ARCHITECTURE microc_tb_arch OF microc_tb_ent IS

COMPONENT microc_ent
PORT (
        clock : IN std_logic;
        inst : IN std_logic_vector (2 downto 0);
        source1 : IN std_logic_vector (3 downto 0);
        source2 : IN std_logic_vector (3 downto 0);
        destination : IN std_logic_vector (3 downto 0);
        data : IN std_logic_vector (31 downto 0);
        jump : OUT std_logic;
        output : OUT std_logic_vector (31 downto 0)
        );
END COMPONENT;

SIGNAL sig_clock : std_logic := '0';
SIGNAL sig_inst : std_logic_vector (2 downto 0);
SIGNAL sig_source1 : std_logic_vector (3 downto 0);
SIGNAL sig_source2 : std_logic_vector (3 downto 0);
SIGNAL sig_destination : std_logic_vector (3 downto 0);
SIGNAL sig_data : std_logic_vector (31 downto 0);
SIGNAL sig_jump : std_logic;
SIGNAL sig_output : std_logic_vector (31 downto 0);

CONSTANT CYCLE : TIME := 50 ns;
CONSTANT ZERO : std_logic_vector (31 downto 0) :=
"00000000000000000000000000000000";

BEGIN

sig_clock <= NOT sig_clock AFTER CYCLE/2;

DUT_microc: microc_ent PORT MAP (clock => sig_clock,
                        inst => sig_inst,
                        source1 => sig_source1,
                        source2 => sig_source2,
                        destination => sig_destination,
                        data => sig_data,
                        jump => sig_jump,
                        output => sig_output);

PROCESS
```

```
BEGIN

-- load #51, reg0
sig_inst <= "101";
sig_source1 <= "0000";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= "00000000000000000000000001010001";
wait for 2*CYCLE;

-- load #02, reg1
sig_inst <= "101";
sig_source1 <= "0000";
sig_source2 <= "0000";
sig_destination <= "0001";
sig_data <= "00000000000000000000000000000010";
wait for CYCLE;

-- add reg0, reg1, reg2
sig_inst <= "001";
sig_source1 <= "0000";
sig_source2 <= "0001";
sig_destination <= "0010";
sig_data <= ZERO;
wait for CYCLE;

-- sub reg0, reg1, reg3
sig_inst <= "010";
sig_source1 <= "0000";
sig_source2 <= "0001";
sig_destination <= "0011";
sig_data <= ZERO;
wait for CYCLE;

-- mul reg0, reg1, reg4
sig_inst <= "011";
sig_source1 <= "0000";
sig_source2 <= "0001";
sig_destination <= "0100";
sig_data <= ZERO;
wait for CYCLE;

-- mov reg1, reg5
sig_inst <= "000";
sig_source1 <= "0001";
sig_source2 <= "0000";
```

```
sig_destination <= "0101";
sig_data <= ZERO;
wait for CYCLE;

-- mov reg1, reg6
sig_inst <= "000";
sig_source1 <= "0001";
sig_source2 <= "0000";
sig_destination <= "0110";
sig_data <= ZERO;
wait for CYCLE;

-- cmp reg0, reg5
-- compare and not equal, no jump
sig_inst <= "100";
sig_source1 <= "0000";
sig_source2 <= "0101";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- cmp reg5, reg6
-- compare and equal, jump
sig_inst <= "100";
sig_source1 <= "0101";
sig_source2 <= "0110";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- penalty 2 clocks
wait for CYCLE;
wait for CYCLE;

-- read reg0
sig_inst <= "110";
sig_source1 <= "0000";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- read reg1
sig_inst <= "110";
sig_source1 <= "0001";
sig_source2 <= "0000";
```

```
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- read reg2
sig_inst <= "110";
sig_source1 <= "0010";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- read reg3
sig_inst <= "110";
sig_source1 <= "0011";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- read reg4
sig_inst <= "110";
sig_source1 <= "0100";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- read reg5
sig_inst <= "110";
sig_source1 <= "0101";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

-- read reg6
sig_inst <= "110";
sig_source1 <= "0110";
sig_source2 <= "0000";
sig_destination <= "0000";
sig_data <= ZERO;
wait for CYCLE;

END PROCESS;
END microc_tb_arch;
```

```
CONFIGURATION microc_tb_config OF microc_tb_ent IS
        FOR microc_tb_arch
             FOR ALL : microc_ent
                  USE ENTITY WORK.microc_ent(microc_arch);
             END FOR;
        END FOR;
END microc_tb_config;
```

The input stimulus from the testbench of Example 49 is a set of instructions as shown:

```
(1) LOAD 51, reg0
(2) LOAD 2, reg1
(3) ADD reg0, reg1, reg2
(4) SUB reg0, reg1, reg3
(5) MUL reg0, reg1, reg4
(6) MOVE reg1, reg5
(7) MOVE reg1, reg6
(8) CJE reg0, reg5
(9) CJE reg5, reg6
(10) CJE reg5, reg6
(11) CJE reg5, reg6
(12) READ reg0
(13) READ reg1
(14) READ reg2
(15) READ reg3
(16) READ reg4
(17) READ reg5
(18) READ reg6
```

These instructions will be passed through the pipeline of the microcontroller as shown:

Input Stimulus	PREDECODE	DECODE	EXECUTE
LOAD 51, reg0	**LOAD** 51, reg0		
LOAD 2, reg1	**LOAD** 2, reg1		
ADD reg0, reg1, reg2	**ADD** reg0, reg1, reg2		
SUB reg0, reg1, reg3	**SUB** reg0, reg1, reg3	**ADD** reg0, reg1, reg2	
MUL reg0, reg1, reg4	**MUL** reg0, reg1, reg4	**SUB** reg0, reg1, reg3	**ADD** reg0, reg1, reg2
MOVE reg1, reg5	**MOVE** reg1, reg5	**MUL** reg0, reg1, reg4	**SUB** reg0, reg1, reg3
MOVE reg1, reg6	**MOVE** reg1, reg6	**MOVE** reg1, reg5	**MUL** reg0, reg1, reg4
CJE reg0, reg5	**CJE** reg0, reg5	**MOVE** reg1, reg6	**MOVE** reg1, reg5

continued

Input Stimulus	PREDECODE	DECODE	EXECUTE
CJE reg5, reg6	**CJE** reg5, reg6	**CJE** reg0, reg5	**MOVE** reg1, reg6
CJE reg5, reg6	**CJE** reg5, reg6	**CJE** reg5, reg6	**CJE** reg0, reg5
CJE reg5, reg6	**CJE** reg5, reg6	**CJE** reg5, reg6	**CJE** reg5, reg6
READ reg0	**NOP**	**NOP**	**NOP**
READ reg1	**NOP**	**NOP**	**NOP**
READ reg2	**READ** reg2	**NOP**	**NOP**
READ reg3	**READ** reg3	**READ** reg2	**NOP**
READ reg4	**READ** reg4	**READ** reg3	**READ** reg2
READ reg5	**READ** reg5	**READ** reg4	**READ** reg3
READ reg6	**READ** reg6	**READ** reg5	**READ** reg4
		READ reg6	**READ** reg5
			READ reg6

The input stimulus for ***READ*** `reg0` and ***READ*** `reg1` will not be executed by the microcontroller. During the instruction of ***CJE*** `reg5`, `reg6` is executed by *execute* block and both registers `reg5` and `reg6` have the same contents. When this occurs, output signal **`jump`** will be asserted to logical `'1'`. The output signal **`flush`** will also be asserted to logical `'1'`. This will cause *predecode* block, *decode* block, and *register file* block to be flushed.

It takes two cycles for the instruction ***NOP*** to travel from *predecode* block to *execute* block. Therefore the two instructions shown in shade (***READ*** `reg0` and ***READ*** `reg1`) will not be executed by the microcontroller.

The timing waveform simulation results are shown in Fig. 56. The output signal from the microcontroller ***output*** drives values of *53, 4F, A2, 2* and *2*. These values are the contents of registers `reg2`, `reg3`, `reg4`, `reg5` and `reg6`.

From Fig. 56, **`jump`** is asserted to logical `'1'` at clock 11 and clock 12. When **`jump`** de-asserts back to logical `'0'` at clock 13, the microcontroller takes three clock cycles to read the contents of register `reg2` and drives this value on **`output`**. This results in the microcontroller getting a penalty hit of 2 clock cycles.

This is the reason why **`output`** drives only contents of `reg2`, `reg3`, `reg4`, `reg5` and `reg6` but not `reg0` and `reg1`. The first two instructions after the branch (***READ*** `reg0` and ***READ*** `reg1`) were flushed away from the pipeline when the branch occurred.

Appendix E shows the full synthesis of this pipeline microcontroller design. Top-level design constraints are used as inputs to Design Compiler to synthesize the design. Synthesis tweaks are also included in Appendix E to show the reader how the synthesized results of the microcontroller are tweaked to obtain optimal performance.

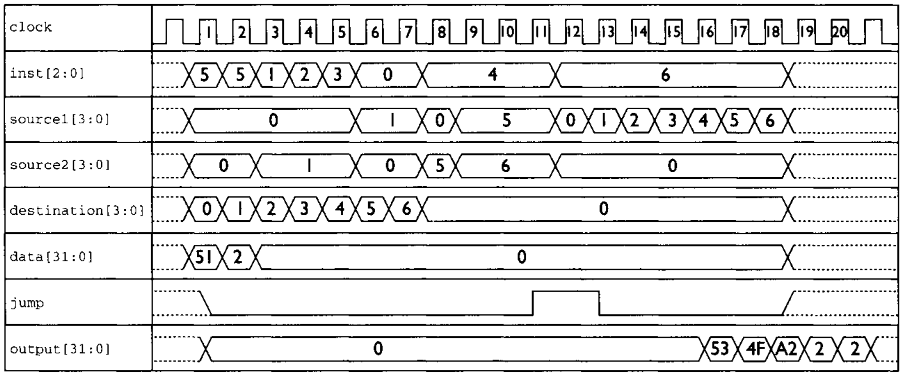


FIGURE 56 Timing Waveform For Microcontroller Testbench.

This Page Intentionally Left Blank

PART 2

LOGIC SYNTHESIS WITH SYNOPSYS

7 TIMING CONSIDERATIONS IN DESIGN

The examples that are shown in Chapters 2 through 6 do not consider timing issues. Those examples consider only design functionality. However, in VHDL synthesis the timing and functionality of a design must always be considered together. When a designer has verified that his/her design is functionally correct through simulations, he/she will proceed with synthesis.

In synthesis, VHDL code is mapped into hardware logic gates for a specific technology library. During this phase of synthesis the designer will also input design constraints into the synthesis tool. This would allow the tool to map more efficiently to logic gates. Logic optimization is performed, if necessary, in order to obtain the best possible synthesis result for a given VHDL code and design constraint. If the synthesized result meets all timing criteria, the designer can move forward to layout. However if timing is not met, the designer will have to analyze the design to fix the timing violations.

In general there are two kinds of timing violations that exist in a design — setup timing violation and hold timing violation.

7.1 SETUP TIMING VIOLATION

Setup timing violation is usually encountered when a design is exposed to design constraints that are "too tight." In other words, the design is over-constrained.

For example, the architecture of a certain design is only able to perform up to a speed of 100 MHz using a 0.5–μ technology. When the designer tries to synthesize the design to perform at a speed of 133 MHz, the synthesized result will show countless setup violations. The synthesized result will have combinational logic that has delays greater than clock period.

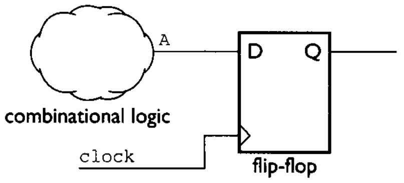


FIGURE 57 Diagram Showing Combinational Logic Driving Signal A as Input to Flip-Flop.

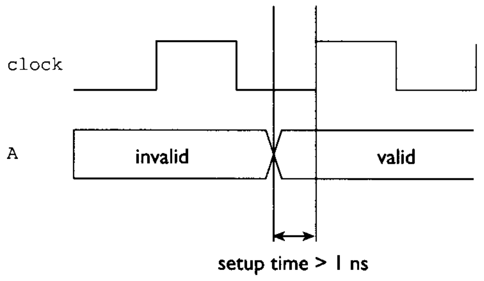


FIGURE 58 Timing Diagram Showing Setup Time on Signal A.

In other words, setup violations are timing violations that cannot be met as the propagation delay for a logic path is longer than the required time.

In design terms, a signal/bus is said to have an X amount of setup time requirement. This would mean that the signal/bus must be valid X unit time before the rise of clock for a design that consists of positive edge triggered flip-flop. Similarly for a design that consists of negative edge triggered flip-flop, the signal/bus must be valid X unit time before the falling edge of clock.

In general, for a logic component with an X setup time requirement, it simply means that the input to that logic component must be valid X unit time prior to an evaluation of the condition of the input signals by that logic component.

Figure 57 shows a combinational logic as an input to a positive edge triggered flip-flop. The signal **A**, which is the output of the combinational logic, must meet the setup time requirement of the flip-flop. If the flip-flop has a setup time requirement of 1 ns, and **clock** has a period of 10 ns, signal **A** must be valid at least 1 ns before the rising edge of clock.

Figure 58 shows signal **A** valid at or before 1 ns from the rising edge of clock.

7.2 HOLD TIMING VIOLATION

Hold time violation is of a different nature than setup violation. For every sequential component, there is a hold time requirement in which the input signal must be held valid for the entire hold time. If the input signal is invalidated during this hold time, a hold time violation is generated.

Hold time violation often occurs in designs that are "too fast." For example, if the input to a positive edge triggered flip-flop changes too fast (the input changes before the flip-flop is able to capture that input), a hold time violation is generated.

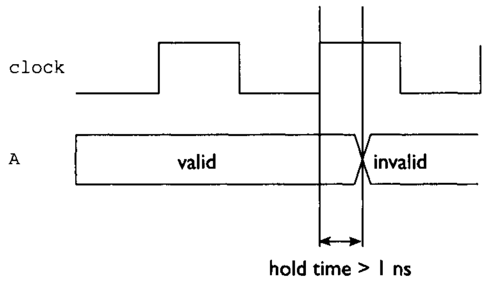


FIGURE 59 Timing Diagram Showing Hold Time on Signal A.

As can be seen from the logic diagram of Fig. 57, Fig. 59 shows that signal **A** must be held by the combinational logic for an X amount of time. The value of this X amount of time would depend on the hold time requirement of the flip-flop. If the flip-flop has a hold time requirement of 1 ns with a **clock** period of 10 ns, signal **A** must be held valid for at least 1 ns after the rising edge of clock.

7.3 SETUP/HOLD TIMING CONSIDERATIONS IN SYNTHESIS

Upon synthesis, the designer must run static timing analysis on the synthesized result to check for setup and hold time violations. Figure 60 shows a diagram for a synthesized circuit.

To ensure that there is no setup violation, the equation that follows must hold true:

$$t_{prop} + t_{delay} < t_{clock} - t_{setup}$$

Here, t_{prop} is the propagation delay from input clock to output Q of flip-flop; t_{delay} is the propagation delay across combinational logic; t_{setup} is the setup time requirement of flip-flop; and t_{clock} is the clock period.

To ensure there is no hold time violation, the equation that follows must hold true:

$$t_{delay} + t_{prop} > t_{hold}$$

Here, t_{prop} is propagation delay from input clock to output Q of flip-flop; t_{delay} is the propagation delay across combinational logic; and t_{hold} is the hold time requirement of flip-flop.

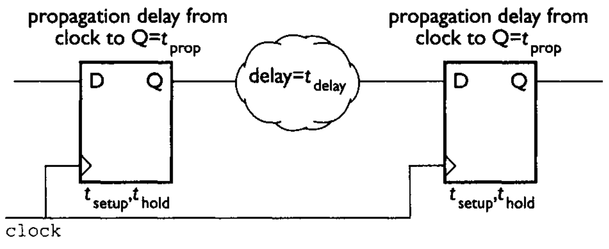

propagation delay from clock to Q=t_prop
propagation delay from clock to Q=t_prop
D
Q
delay=t_delay
D
Q
t_setup,t_hold
t_setup,t_hold
clock

FIGURE 60 Synthesized Circuit for Static Timing Analysis.

When timing violations are encountered in synthesis, several options can be taken by the designer to attempt to fix these violations.

Synthesis optimization using synthesis tool. This is the simplest and easiest method for fixing timing violations. Most synthesis tools have inbuilt algorithms that allow the designer to make tweaks to the synthesized result in order to obtain an optimized result. However, if a design has overly severe timing violations, it will not be possible for the synthesis tool to fix them. A good rule of thumb to remember is that synthesis tools can frequently obtain around 10%-performance improvement. If the synthesized results have violations that exceed 20%-possible timing improvement, the designer must resort to a more manual approach.

Microarchitecural tweaks are manual compared to using a synthesis tool for fixing timing violations. In this method, the designer must have a strong understanding of the microarchitectural implementation of the design. The designer must also understand the transformation of the VHDL code of that design into hardware logic gates. Once the designer grasps these concepts, he/she can tweak the VHDL code to change the microarchitectural implementation to the desired implementation and thus fix the timing violation.

Architectural changes is the last option that a designer has to fix the timing violations. This approach is not recommended as it would mean that whatever design is being worked on must change architecturally. This should be the last resort for a designer and be used only if synthesis tool optimization and microarchitectural tweaks cannot fix the timing violations.

7.4 MICROARCHITECTURAL TWEAKS FOR FIXING SETUP TIME VIOLATIONS

From the design perspective, setup violations mean there are performance problems on the synthesized design. In other words, the synthesized design utilizes more logic levels than allowed. The more logic levels a design has, the greater the propagation delay of the design.

When a design has setup violations that cannot be fixed by using synthesis tool optimization algorithms, the designer must resort to making either tweaks in the VHDL code or microarchitectural implementation changes.

There are several ideas that can be used to perform microarchitectural tweaks when a desired design is not performing as required. They include:

- logic duplication to generate independent paths;
- logic duplication prior to selection of later arriving signal;
- balancing of logic between flip-flops; and
- ripple decoding vs. multiple stage decoding.

7.4.1 Logic Duplication to Generate Independent Paths

From Fig. 61, assuming a critical path exists from ***A*** to ***Q2***, logic optimization on combinational logic ***X***, ***Y***, and ***Z*** would be difficult because ***X*** is shared with ***Y*** and ***Z***. The optimization that can be performed on ***X*** might not be optimal due to the sharing

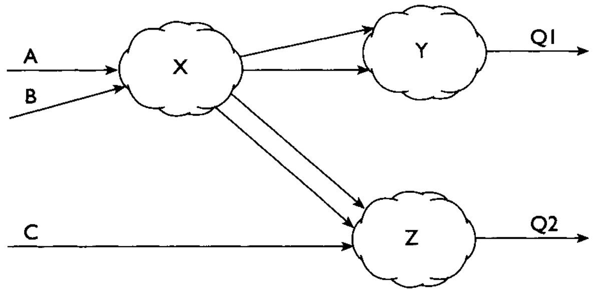


FIGURE 61 Diagram Showing Generic Combinational Logic.

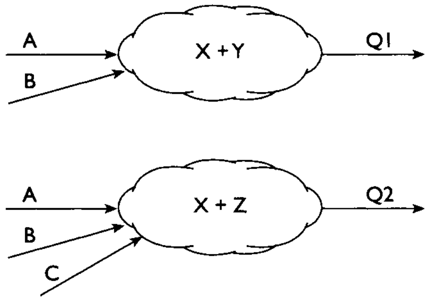


FIGURE 62 Diagram Showing Generic Combinational Logic with "Logic Duplication"

of ***X***. To overcome this problem, the designer can duplicate ***X*** and build two independent paths for ***Q1*** and ***Q2*** (as shown in Fig. 62).

With logic duplication to generate two independent paths of ***'X+Y'*** and ***'X+Z'***, each can be independently optimized for ***Q1*** and ***Q2***. This method improves timing but it increases the die area as now more logic gates are utilized.

7.4.2 Logic Duplication Prior to Selection of Later Arriving Signal

Figure 63 shows a generic logic circuit whereby signal ***Q*** has a setup violation. Signal ***sel*** is a late arriving signal compared to signals ***A*** and ***B***. Combinational logic ***C*** is assumed to be optimized and therefore cannot be further optimized.

To fix the setup violation of signal ***Q***, the earlier arriving signals of ***A*** and ***B*** with respect to signal ***sel*** can be decoded in advance. The designer can duplicate

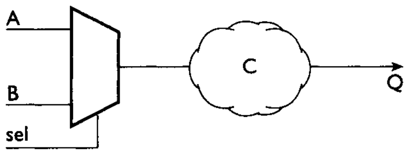


FIGURE 63 Diagram Showing a Generic Logic Circuit.

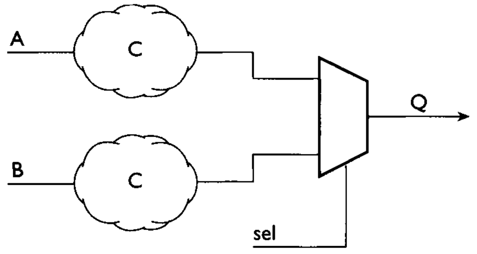


FIGURE 64 Diagram Showing a Generic Logic Circuit with Logic Duplication for Decoding of Early Arriving Signals.

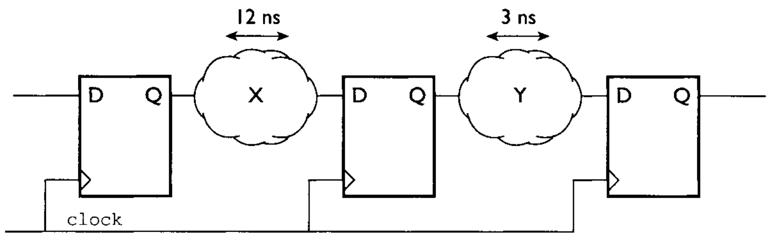


FIGURE 65 Diagram Showing Generic Design Utilizing Different Stages of Decoding.

combinational logic **C** and bring it forward to the inputs of the multiplexer (See Fig. 64).

Since **A** and **B** are early arriving signals compared to signal **sel**, the combinational logic **C** is brought forward to enable the decoding of signals **A** and **B** prior to selection of either of these using the multiplexer.

This method can improve timing but it will increase the die area because now more logic gates are utilized (needs two combinational logic C instead of just one).

7.4.3 Balancing of Logic between Flip-Flops

A common method to fix setup violations in designs that utilize different stages of decoding is to balance the logic between each stage. For example, Fig. 65 shows a generic design that consists of two sets of combinational logic **X** and **Y** between three flip-flops.

Combinational logic **X** has a delay of 12 ns while combinational logic **Y** has a delay of 3 ns. If **clock** has a 10-ns period, the output from combinational logic **X** would have setup violation. The delay of **X** is greater than the **clock** period itself.

Logic **X** will show a setup violation of 2 ns while **Y** still has plenty of time to meet the setup requirement. The extra time from **Y** can be used to fix the setup violation that **X** has.

At this point, the designer can balance the logic between **X** and **Y** by moving part of the logic from **X** to **Y**. This way a more efficient and balanced logic is achieved between the two combinational logic **X** and **Y**.

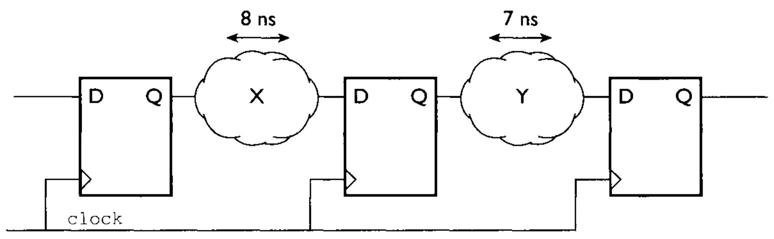


FIGURE 66 Diagram Showing Generic Design Utilizing Different Stages of Decoding with Logic Balancing.

With part of the logic of *X* moved to *Y*, the delay that *X* now has is only 8 ns (see Fig. 66). The additional delay of 4 ns from *X* is added into *Y*. This brings the delay of *Y* to 7 ns.

By moving and balancing the logic between *X* and *Y*, both *X* and *Y* can now meet the setup requirements needed for a 10-ns-**clock** period.

7.4.4 Priority Decoding Versus Multiplex Decoding

Priority decoding is a good design method to use when the designer knows for certain that an input signal is arriving late. The earlier arriving signals can be decoded before the late signal arrives.

For example, a Boolean equation with eight inputs,

$$Q = A.B.C.D.E.F.G.H$$

can be designed using 7 AND gates. Figure 67 shows the design of the Boolean equation using priority decoding while Figure 68 shows the design using multiplex decoding.

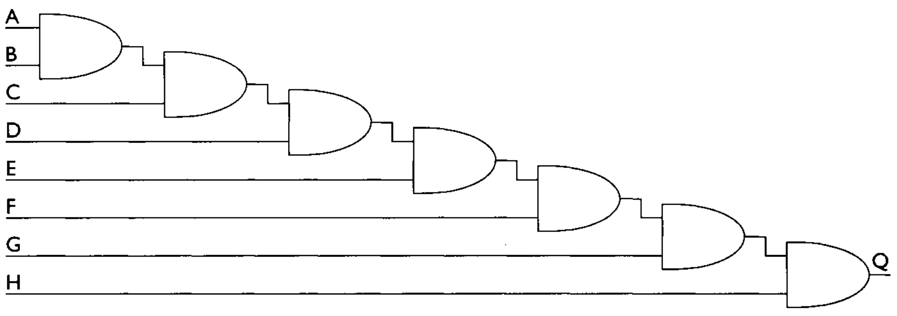


FIGURE 67 Diagram Showing a Priority Decoding Design.

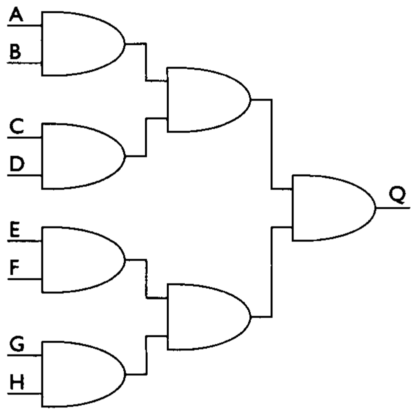


FIGURE 68 Diagram Showing a Multiplex Decoding Design.

Multiplex decoding is most suitable when neither of the input signals arrives later than the other. In general, if there are no late arriving signals, multiplex decoding is much faster than priority decoding.

From Fig. 67, there are 7 levels of logic from input signal *A* to output signal *Q*. If each of the AND gates has a propagation delay of 1 ns, output *Q* is only valid 7 ns after input *A* is valid.

From Fig. 68, there are only 3 levels of logic from input signal *A* to output signal *Q*. Therefore, output *Q* is valid 3 ns after input *A* is valid.

The designer must keep in mind that when input signals arrive at or about the same time, it is always more efficient to use multiplex decoding when designing. Priority decoding is suitable only when either one of the input signals arrives late.

7.5 MICROARCHITECTURAL TWEAKS FOR FIXING HOLD TIME VIOLATIONS

Hold time violations occur when a signal changes too fast or becomes invalid too fast. A simple way to fix hold time violations is to use buffers. Any path that has hold time violations can be fixed by adding buffers in that path. These buffers will add delay to the path and ultimately slow it down.

Figure 69 shows a design whereby signal *B*, which is an input to the flip-flop, shows a hold time violation. To fix this violation, the designer can add buffers to signal *B* before it reaches the flip-flop.

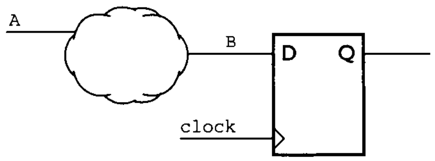


FIGURE 69 Diagram Showing a Generic Design with Hold Time Violation.

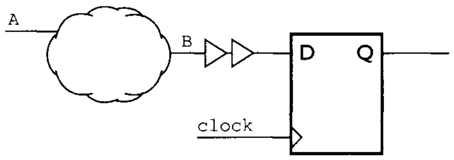


FIGURE 70 Diagram Showing a Generic Design with Hold Time Violation Fixed.

Figure 70 shows a hold time violation fix by using two buffers back-to-back. These additional buffers add the overall delay of signal *B* to input of the flip-flop.

7.6 ASYNCHRONOUS/FALSE PATHS

Designers need to be aware of whether any portions of a design contain paths that are asynchronous. It is good practice for a designer always to document these asynchronous paths.

When a designer puts his/her VHDL code into synthesis, it is encouraged that the designer "inform" the synthesis tool of existing asynchronous paths. This would allow the synthesis tool to understand that an asynchronous path that is having setup violation is in actual fact showing a false violation.

7.7 MULTICYCLE PATHS

Multicycle paths are paths that have delays over more than 1-clock cycle. Again it is good practice for a designer always to document these multicycle paths.

From Fig. 71, the combinational logic between the two rising edge triggered flip-flop has a delay of 35 ns while the clock period is only 10 ns. The combinational logic is a multicycle logic that requires 4-clock period. This means the output from the first flip-flop to the input of the second flip-flop is a multicycle path.

In synthesis, it is encouraged that the designer "inform" the synthesis tool of any multicycle paths. This would allow the synthesis tool to more efficiently optimize the other logic paths that are not meeting setup requirements rather than to attempt to optimize the multicycle path.

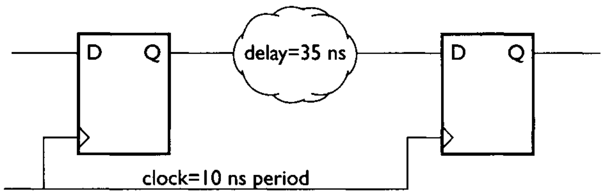


FIGURE 71 Diagram Showing a Design with Multicycle Path.

This Page Intentionally Left Blank

8 VHDL SYNTHESIS WITH TIMING CONSTRAINTS

The most commonly used synthesis tool in the ASIC industry is ***Synopsys's Design Compiler.*** The following VHDL examples are synthesized using ***Design Compiler.*** Design constraints and synthesis tweaks are based on commands and synthesis options from ***Design Compiler***. For more information on synthesis options and commands, please refer to ***Synopsys's Design Compiler Manual.***

The examples are synthesized using `'class.db'` synthesis technology library. This library is used by Synopsys for synthesis training classes.

When a design is synthesized with a set of given constraints, there is no guarantee that the synthesized result would be able to meet performance and area criteria the first time it is synthesized. In fact, most of the time, a design would not be able to meet performance or area requirements upon the initial synthesis run. When situations such as these occur, the designer must then use the synthesis tool to tweak the design to yield an optimum design either from a performance perspective, an area perspective, or both.

8.1 INTRODUCTION TO DESIGN COMPILER

Synopsys's synthesis tool is divided into two sections, ***Design Analyzer*** and ***Design Compiler.*** The former is the graphical interface to Synopsys's synthesis tool and the latter is the command shell interface to the same synthesis tool.

When using Synopsys's synthesis tool, there is a startup file that you must have in your current working directory from which you invoke the synthesis tool. (This assumes that you have set up the environment variables to point to the right path for Synopsys's ***Design Analyzer*** and ***Design Compiler***.) This startup file is ***`.synopsys_dc.setup`*** file, and there should be two of them. One would be in the root directory from which Synopsys is installed, and the other would be a local startup file in your current working directory. This local startup file should be used to specify your individual design specifications.

In these local startup files, four important parameters must be set before you can perform any synthesis.

- ***search_path***
  This parameter is used to specify to the synthesis tool all the paths that it should search when looking for a synthesis technology library to reference during synthesis.
- ***target_library***
  The file pointed to by this parameter is the library that contains all the logic cells for mapping during synthesis.
- ***symbol_library***
  This parameter points to the library that contains "visual" information on the logic cells in the synthesis technology library. All logic cells have a symbolic representation and information about the symbols is stored in this library.
- ***link_library***
  This parameter points to the library that contains information on the logic gates in the synthesis technology library.

An example on use of these four variables from a ***.synopsys_dc.setup*** file:

```
search_path = ". /synopsys/libraries/syn
                /cell_library/libraries/syn"
target_library = class.db
link_library = class.db
symbol.library = class.sdb
```

Once you have these variables set up correctly, you are ready to invoke the synthesis tool. To invoke the graphical interface:

```
unix_prompt> design_analyzer
```

To invoke the command line shell interface:

```
unix_prompt> dc_shell
```

8.2 USING DESIGN COMPILER FOR SYNTHESIS

Design Compiler has many options that allow the designer a great deal of flexibility when synthesizing a design. Chapter 8.3 presents different possible options to use when a synthesized design that does not meet the required performance criteria is encountered. Chapter 8.2 presents some general commands that are often used during synthesis. These include, for example, reading a design, setting constraints on the design, creating clocks, handling clock skews, and others.

- ***Reading a Design***
  Once the environment has been set up (as shown in Chapter 8.1), a design must be read into ***Design Compiler.*** The design can be represented in many different formats such as VHDL, Verilog, DB, State Table, EDIF, Equation, LSI, Mentor, XNF and PLA.

```
dc_shell> read -format <format_type> <filename>
```

  For example, to read in a VHDL file example.vhd:

```
dc_shell> read -format vhdl example.vhd
```

  Multiple files can also be read in using one single command line:

```
dc_shell> read -format vhdl {example.vhd
example1.vhd example2.vhd example3.vhd}
```

- ***Initial Checking of a Design***
  After a design has been read into ***Design Compiler***, the command `check_design` can be used by the designer to check for minor design problems. Command `check_design` will check for shorts, opens, nonconnection, multiple connection and multiple instantiations.

```
dc_shell> check_design
```

- ***Creating a Clock***
  When a designer wishes to constrain a design that has been read into ***Design Compiler,*** he/she must first create a clock. This clock is associated with the clock pin of the design. If the design does not have a clock pin, then this clock is created as a virtual clock. A clock must always be created as a reference for timing analysis.

```
dc_shell> create_clock -name <clock_name> -
period <clock_period> <design_clock_pin_name>
```

  If a design has a clock pin with the name `clockA` and is to operate with a 10-ns clock period of 50% duty cycle:

```
dc_shell> create_clock -name clock -period 10
clockA
```

  However, if a design does not have a clock pin, then a virtual clock is created:

```
dc_shell> create_clock -name clock -period 10
```

If a designer wishes to create a clock but with a duty cycle other than 50%:

```
dc_shell> create_clock -name clock -period 10 -
waveform {2 10}
```

This command will create a virtual clock of period 10 ns with the rising edge at 2 ns and falling edge at 10 ns.

- *Setting Clock Skew*
  Synopsys during synthesis is not able to synthesize any clock trees. Clock trees rely heavily on the placement of cells on the layout portion of a design. Therefore, during synthesis, ***Design Compiler*** cannot synthesize a clock tree. To overcome this problem of clock-tree synthesis, the designer can use the command ***set_clock_skew*** to apply a clock skew to the created clock. This clock skew is a method of modeling propagation delay that exists in a clock tree.

```
dc_shell> set_clock_skew -rise_delay
<rising_clock_skew> -fall_delay
<falling_clock_skew> <clock_name>
```

This command is useful in applying an estimated clock skew onto the clock tree. However, if there is information being back-annotated into ***Design Compiler*** from layout, then the actual clock delay can be calculated along the clock tree.

```
dc_shell> set_clock_skew -propagated
<clock_name>
```

- **Setting Input and Output Delays on a Design**
  When constraining a design, it is important for the designer to specify to ***Design Compiler*** the input and output delays of a design. By using these input and output delay values and the clock period, ***Design Compiler*** is then able to analyze the required timing for a certain path.

```
dc_shell> set_input_delay -clock <clock_name>
<input_delay> <input_port>
```

For example, to specify an input port ***INPUTA*** to have an input delay of 2 ns with reference to clock ***CLOCKA***.

```
dc_shell> set_input_delay 2 -clock CLOCKA
INPUTA
```

For this command, because only one number (the value of 2) is used, ***Design Compiler*** will use the same number for minimum and maximum for rise and fall input time for the input port ***INPUTA***.

If the designer knows for certain the minimum and maximum input time for the input port, the designer can use the -max and -min option.

```
dc_shell> set_input_delay -clock CLOCKA -max
<delay_value> INPUTA
dc_shell> set_input_delay -clock CLOCKA -min
<delay_value> INPUTA
```

Output delay is specified in the same way as that for input delay.

```
dc_shell> set_output_delay -clock <clock_name>
<delay_value> <output_port>
```

If the designer wishes to specify for output delay using minimum and maximum time for the output port:

```
dc_shell> set_output_delay -clock <clock_name>
-max <delay_value> <output_port>
dc_shell> set_output_delay -clock <clock_name>
-min <delay_value> <output_port>
```

- *Synthesizing a Design*
  Once the constraints have been set on the design, the designer can synthesize the design using the command compile. There are many other commands that can be used together with the compile command in order to achieve optimum synthesis results. These commands are discussed in detail in Chapter 8.3.

```
dc_shell> compile -map_effort medium
```

- *Saving a Design*
  Design Compiler can save a database in different formats. A common format in which to save a synthesized database is the Synopsys DB format.

```
dc_shell> write -format db -output <filename.db>
```

8.3 PERFORMANCE TWEAKS

There are many ways in which a designer can tweak his/her design to obtain optimum performance results. Several of the more general often used tweaks for performance optimization are as follows:

1. compilation with `map_effort` high option;
2. group critical paths together and giving them weight factor;
3. flattening a design;

4. characterizing submodules;
5. register balancing;
6. use of FSM Compiler to optimize finite state machine designs;
7. choosing high-speed implementation for high-level functional modules; and
8. balancing of logic trees with heavy loading.

8.3.1 Compilation with 'map_effort high' Option

Generally, during synthesis, it is advisable for the designer to run a quick synthesis on his/her design using a ***map_effort medium*** option when employing design constraints. This would allow the designer to have a feel for the timing violations if any exist. Using a ***map_effort high*** option during the first synthesis run is not advisable as the run-time for a ***map_effort high*** option is significantly longer than that for a ***map_effort medium***.

In general, a rule of thumb to remember is that moving from ***map_effort medium*** to ***map_effort high*** compilation can improve design performance by about 10%.

Example 50 uses component inference to obtain a 32-bit adder; ***inputA*** and ***inputB*** are 32 bits each and the output ***Sum*** is 33 bits wide. The 33rd bit is the ***CARRY*** bit.

EXAMPLE 50 Synthesizable VHDL Code for a 32-Bit Adder

filename: adder_timing.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY adder_timing_ent IS
PORT (
     inputA : IN std_logic_vector (31 downto 0);
     inputB : IN std_logic_vector (31 downto 0);
     Sum : OUT std_logic_vector (32 downto 0)
     );
END adder_timing_ent;

ARCHITECTURE adder_timing_arch OF adder_timing_ent IS
BEGIN
     Sum <= signed('0'&inputA) + signed('0'&inputB);
END adder_timing_arch;
```

Although the design in Example 50 does not have a clock, in synthesis with timing constraints a virtual clock must be declared. This is required as input and output delay declarations are referenced to a clock.

By assuming a virtual clock of 10-ns cycle, the inputs to the adder are assumed to receive only valid data 3 ns after the rise of a clock. Also, the output of the adder is assumed to be driven 3 ns before the rise of the clock.

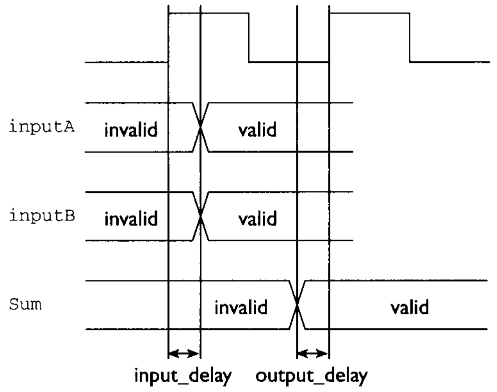


FIGURE 72 Timing Diagram Showing Input Delay and Output Delay with Reference to a Virtual Clock.

To place the constraint upon the adder, the following steps are taken.

1. To invoke ***Design Compiler*** without the Graphical Unit Interface (GUI).

```
unix_prompt> dc_shell
dc_shell>
```

2. To read the VHDL code for the design ***into Design Compiler***.

```
dc_shell> read -format vhdl adder_timing.vhd
```

3. ***Design Compiler*** will analyze and elaborate the VHDL file. When this is completed, set `current_design` to the 32-bit adder.

```
dc_shell> current_design = adder_timing_ent
```

4. Set the design constraint on the adder.

```
dc_shell> create_clock -name clock -period 10.0
dc_shell> set_input_delay 3.0 -clock clock
inputA*
dc_shell> set_input_delay 3.0 -clock clock
inputB*
dc_shell> set_output_delay 3.0 -clock clock Sum
```

5. Compile the design using ***`map_effort medium`*** option

```
dc_shell> compile -map_effort medium
```

6. When compilation is completed, do a ***report_timing*** on the synthesized result to obtain a report of the timing violations.

```
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : adder_timing_ent
Version: 1998.02-1
Date   : Mon Mar 15 22:39:29 1999
****************************************
Operating Conditions:
Wire Loading Model Mode: top

Design            Wire Loading Model          Library
-----------------------------------------------------
adder_timing_ent       05x05                    class

  Startpoint: inputA[5] (input port clocked by clock)
  Endpoint: Sum[11] (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                   Incr      Path
-----------------------------------------------------
  clock clock (rise edge)                 0.00      0.00
  clock network delay (ideal)             0.00      0.00
  input external delay                    3.00      3.00 r
  inputA[5] (in)                          0.00      3.00 r
  add_15/plus/A[5]
 (adder_timing_ent_DW01_addsub_33_1)      0.00      3.00 r
  add_15/plus/U53/Z (IVI)                 0.12      3.12 f
  add_15/plus/U280/Z (ND2I)               0.29      3.41 r
  add_15/plus/U25/Z (AO3P)                0.50      3.91 f
  add_15/plus/U249/Z (NR2I)               0.76      4.67 r
  add_15/plus/U501/Z (ND2I)               0.12      4.79 f
  add_15/plus/U214/Z (AN2I)               0.67      5.46 f
  add_15/plus/U57/Z (ND2I)                0.30      5.75 r
  add_15/plus/U297/Z (ND2I)               0.12      5.87 f
  add_15/plus/U298/Z (ND2I)               0.30      6.17 r
  Point                                         Incr      Path
------------------------------------------------------------------
  add_15/plus/U410/Z (ND2I)                     0.12      6.29 f
  add_15/plus/U411/Z (ND2I)                     0.30      6.59 r
  add_15/plus/U412/Z (ND2I)                     0.12      6.71 f
  add_15/plus/U413/Z (ND2I)                     0.25      6.96 r
  add_15/plus/U24/Z (ENI)                       0.38      7.34 f
  add_15/plus/SUM[11]
(adder_timing_ent_DW01_addsub_33_1)             0.00      7.34 f
  Sum[11] (out)                                 0.00      7.34 f
  data arrival time                                       7.34

  clock clock (rise edge)                      10.00     10.00
  clock network delay (ideal)                   0.00     10.00
  output external delay                                  -3.00
  data required time                                      7.00
------------------------------------------------------------------
  data required time                                      7.00
  data arrival time                                      -7.34
------------------------------------------------------------------
  slack (VIOLATED)                                       -0.34
```

7. The violation shown by ***Design Compiler*** is –0.34 ns. An incremental compilation is performed but with ***map_effort high*** option. Also, note that ***incremental_mapping*** is used. This option permits an incremental compilation rather than recompiling the whole design.

```
dc_shell> compile -map_effort high -
incremental_mapping
```

8. When compilation is completed, ***report_timing*** is executed to obtain data on timing violation.

```
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : adder_timing_ent
Version: 1998.02-1
Date   : Mon Mar 15 22:44:16 1999
****************************************
```

```
Operating Conditions:
Wire Loading Model Mode: top

Design             Wire Loading Model           Library
-----------------------------------------------------------
adder_timing_ent        05x05                    class
  Startpoint: inputB[16] (input port clocked by clock)

  Endpoint: Sum[30] (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                   Incr      Path
-----------------------------------------------------------
  clock clock (rise edge)                 0.00      0.00
  clock network delay (ideal)             0.00      0.00
  input external delay                    3.00      3.00 f
  inputB[16] (in)                         0.00      3.00 f
  add_15/plus/B[16]
 (adder_timing_ent_DW01_addsub_33_1)      0.00      3.00 f
  add_15/plus/U342/Z (ND2I)               0.29      3.29 r
  add_15/plus/U344/Z (AO3P)               0.55      3.84 f
  add_15/plus/U345/Z (ND4P)               0.81      4.65 r
  add_15/plus/U168/Z (ND2I)               0.12      4.77 f
  add_15/plus/U435/Z (ND2I)               0.25      5.03 r
  add_15/plus/U196/Z (ND2I)               0.12      5.15 f
  add_15/plus/U511/Z (IVI)                0.29      5.44 r
  add_15/plus/U513/Z (ND2I)               0.28      5.72 f
  add_15/plus/U197/Z (ND2I)               0.25      5.97 r
  add_15/plus/U258/Z (ND2I)               0.12      6.09 f
  add_15/plus/U84/Z (NR2I)                0.57      6.66 r
  add_15/plus/U95/Z (ENI)                 0.34      7.00 f
  add_15/plus/SUM[30]
 (adder_timing_ent_DW01_addsub_33_1)      0.00      7.00 f
  Sum[30] (out)                           0.00      7.00 f
  data arrival time                                 7.00

  clock clock (rise edge)                10.00     10.00
  clock network delay (ideal)             0.00     10.00
  output external delay                  -3.00      7.00
  data required time                                7.00
-----------------------------------------------------------
  data required time                                7.00
  data arrival time                                -7.00
-----------------------------------------------------------
  slack (MET)                                       0.00
```

9. From the timing report, ***Design Compiler*** has optimized away the critical path with a setup violation of –0.34 ns by using a `map_effort high` option with `incremental_mapping`.

8.3.2 Group Critical Paths Together and Give Them a Weight Factor

For designs that still cannot meet timing requirements even with a ***map_effort high*** compilation option, the designer can use the ***group_path*** command to group timing critical paths and set a weight factor on these critical paths. The larger the value of the weight, the more effort will ***Design Compiler*** use to try to optimize that path. This command allows a designer to prioritize critical paths for optimization.

EXAMPLE 51 Example of a 16-Bit Subtractor

Filename: subtractor_timing.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY subtractor_ent IS
PORT (
      inputA : IN std_logic_vector (15 downto 0);
      inputB : IN std_logic_vector (15 downto 0);
      outputC : OUT std_logic_vector (15 downto 0)
      );
END subtractor_ent;

ARCHITECTURE subtractor_arch OF subtractor_ent IS
BEGIN
      outputC <= signed(inputA) - signed(inputB);
END subtractor_arch;
```

The design constraints for Example 51:

```
dc_shell> read -format vhdl
subtractor_timing.vhd
dc_shell> Create_clock -name clock -period 5
dc_shell> set_input_delay 1 -clock clock
inputA*
dc_shell> set_input_delay 1 -clock clock
inputB*
dc_shell> set_output_delay 1 -clock clock
outputC*
```

1. The subtractor design is compiled with a ***map_effort medium*** option.

```
dc_shell> current_design =
subtractor_timing_ent
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : subtractor_ent
Version: 1998.02-1
Date   : Tue Mar 16 17:47:02 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design           Wire Loading Model          Library
-----------------------------------------------------
subtractor_ent        05x05                   class

  Startpoint: inputA[8] (input port clocked by clock)
  Endpoint: outputC[11]
           (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                   Incr     Path
-----------------------------------------------------
  clock clock (rise edge)                 0.00     0.00
  clock network delay (ideal)             0.00     0.00
  input external delay                    1.00     1.00 r
  inputA[8] (in)                          0.00     1.00 r
  sub_15/minus/A[8]
(subtractor_ent_DW01_sub_16_1)            0.00     1.00 r
  sub_15/minus/U44/Z (IVI)                0.12     1.12 f
  sub_15/minus/U68/Z (ND2I)               0.25     1.38 r
  sub_15/minus/U32/Z (AN2I)               0.34     1.72 r
  sub_15/minus/U97/Z (ND2I)               0.28     1.99 f
  sub_15/minus/U66/Z (NR2I)               0.57     2.56 r
  sub_15/minus/U230/Z (ND2I)              0.19     2.75 f
  Point                                     Incr      Path
  -----------------------------------------------------------
  sub_15/minus/U222/Z (AO3P)                0.68      3.43 r
  sub_15/minus/U180/Z (ND2I)                0.12      3.55 f
  sub_15/minus/U181/Z (ND2I)                0.25      3.81 r
  sub_15/minus/U80/Z (ENI)                  0.38      4.18 f
  sub_15/minus/DIFF[11]
(subtractor_ent_DW01_sub_16_1)              0.00      4.18 f
  outputC[11] (out)                         0.00      4.18 f
  data arrival time                                   4.18

  clock clock (rise edge)                   5.00      5.00
  clock network delay (ideal)               0.00      5.00
  output external delay                    -1.00      4.00
  data required time                                  4.00
-------------------------------------------------------------
  data required time                                  4.00
  data arrival time                                  -4.18
-------------------------------------------------------------
 slack (VIOLATED)                                    -0.18
```

2. The subtractor design is having a setup violation of 0.18 ns. A `group_path` command is used to group the critical path and give it a weight factor of 5.

```
dc_shell> group_path -name critical1 -from
inputA[8] -to outputC[11] -weight 5
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : subtractor_ent
Version: 1998.02-1
Date   : Tue Mar 16 17:50:55 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top
```

```
Design           Wire Loading Model          Library
-----------------------------------------------------
subtractor_ent          05x05                  class

  Startpoint: inputA[8] (input port clocked by clock)
  Endpoint: outputC[11]
            (output port clocked by clock)
  Path Group: critical1
  Path Type: max

  Point                                Incr     Path
-----------------------------------------------------
  clock clock (rise edge)              0.00     0.00
  clock network delay (ideal)          0.00     0.00
  input external delay                 1.00     1.00 f
  inputA[8] (in)                       0.00     1.00 f
  sub_15/minus/A[8]
(subtractor_ent_DW01_sub_16_1)         0.00     1.00 f
  sub_15/minus/U373/Z (IVI)            0.24     1.24 r
  sub_15/minus/U376/Z (ND2I)           0.12     1.36 f
  sub_15/minus/U372/Z (AN2I)           0.64     2.00 f
  sub_15/minus/U366/Z (ND2I)           0.25     2.25 r
  sub_15/minus/U387/Z (IVI)            0.18     2.43 f
  sub_15/minus/U321/Z (ND2I)           0.25     2.69 r
  sub_15/minus/U239/Z (IVI)            0.12     2.81 f
  sub_15/minus/U237/Z (ND2I)           0.25     3.06 r
  sub_15/minus/U236/Z (ND2I)           0.12     3.18 f
  sub_15/minus/U235/Z (ND2I)           0.25     3.44 r
  sub_15/minus/U181/Z (ND2I)           0.12     3.56 f
  sub_15/minus/U80/Z (ENI)             0.38     3.93 f
  sub_15/minus/DIFF[11]
(subtractor_ent_DW01_sub_16_1)         0.00     3.93 f
  outputC[11] (out)                    0.00     3.93 f
  data arrival time                             3.93

  clock clock (rise edge)              5.00     5.00
  clock network delay (ideal)          0.00     5.00
  output external delay               -1.00     4.00
  data required time                            4.00
-----------------------------------------------------
  data required time                            4.00
  data arrival time                            -3.93
-----------------------------------------------------
  slack (MET)                                   0.07
  Startpoint: inputA[2] (input port clocked by clock)
  Endpoint: outputC[15]
           (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                 Incr     Path
-----------------------------------------------------
  clock clock (rise edge)               0.00     0.00
  clock network delay (ideal)           0.00     0.00
  input external delay                  1.00     1.00 r
  inputA[2] (in)                        0.00     1.00 r
  sub_15/minus/A[2]
(subtractor_ent_DW01_sub_16_1)          0.00     1.00 r
  sub_15/minus/U84/Z (IVI)              0.18     1.18 f
  sub_15/minus/U144/Z (ND2I)            0.34     1.53 r
  sub_15/minus/U121/Z (ND2I)            0.12     1.65 f
  sub_15/minus/U40/Z (IVI)              0.24     1.89 r
  sub_15/minus/U39/Z (ND2I)             0.28     2.17 f
  sub_15/minus/U146/Z (ND2I)            0.25     2.42 r
  sub_15/minus/U202/Z (ND2I)            0.20     2.62 f
  sub_15/minus/U131/Z (ND2I)            0.25     2.87 r
  sub_15/minus/U246/Z (IVI)             0.12     2.99 f
  sub_15/minus/U244/Z (ND2I)            0.25     3.24 r
  sub_15/minus/U243/Z (ND2I)            0.12     3.36 f
  sub_15/minus/U242/Z (ND2I)            0.25     3.62 r
  sub_15/minus/U117/Z (ENI)             0.38     3.99 f
  sub_15/minus/DIFF[15]
(subtractor_ent_DW01_sub_16_1)          0.00     3.99 f
  outputC[15] (out)                     0.00     3.99 f
  data arrival time                              3.99

  clock clock (rise edge)               5.00     5.00
  clock network delay (ideal)           0.00     5.00
  output external delay                -1.00     4.00
  data required time                             4.00
-----------------------------------------------------
  data required time                             4.00
  data arrival time                             -3.99
-----------------------------------------------------
  slack (MET)                           0.01
```

3. A setup violation of 0.18 ns is fixed by using **`group_path`** and `map_effort high` option.

8.3.3 Logical Flattening of a Design

Logical flattening of a design can be used to break the hierarchy of a design. All logic gates for that particular design will be at the same level of hierarchy. This would allow ***Design Compiler*** to try to optimize those logic gates to gain better performance and area utilization. ***Design Compiler*** during optimization must maintain the integrity of block interface/ports and therefore is not able to optimize across the hierachical boundary.

This option of logical flattening is used for hierarchical designs. However, this option is not suitable for usage if the hierarchical design is large. Too huge a design will take up considerable computing resources (for example, a long time to compile), thus preventing Design Compiler from performing a good optimization.

EXAMPLE 52 Example of a 4-Bit Multiplier

Filename: multiplier_timing.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY multiplier_timing_ent IS
PORT (
      inputA : IN std_logic_vector (3 downto 0);
      inputB : IN std_logic_vector (3 downto 0);
      outputC : OUT std_logic_vector (7 downto 0)
      );
END multiplier_timing_ent;

ARCHITECTURE multiplier_timing_arch OF multiplier_timing_ent IS
BEGIN
      outputC <= signed(inputA) * signed(inputB);
END multiplier_timing_arch;
```

Note: Example 52 is a simple example of a 4-bit multipler. When synthesized, it contains only one level of hierarchy consisting of a multipler from DesignWare library. Therefore, when this design is logically flattened to break the hierarchy, there is a minimum of optimization that can be achieved. Logical flattening of a design can obtain better results if a design contains more than one level of hierarchy. Better optimization results can also be obtained if a design consists of one level of hierarchy but with logic gates external to the level of hierarchy.

The following synthesis scripts to synthesize Example 52 are meant to show the reader how logical flattening can be achieved.

1. To set the design constraint for the multipler:

```
unit_prompt> dc_shell
dc_shell> read -format vhdl
multiplier_timing.vhd
dc_shell> read -format vhdl
multiplier_timing.vhd
dc_shell> current_design =
multiplier_timing_ent
dc_shell> create_clock -name clock -period 10.0
dc_shell> set_input_delay 3 -clock clock
inputA*
dc_shell> set_input_delay 3 -clock clock
inputB*
dc_shell> set_output_delay 2.7 -clock clock
outputC*
```

2. The multiplier is compiled with a **map_effort medium** option.

```
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : multiplier_timing_ent
Version: 1998.02-1
Date   : Tue Mar 16 16:28:09 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design             Wire Loading Model          Library
------------------------------------------------------
multiplier_timing_ent  05x05                     class

  Startpoint: inputA[1] (input port clocked by clock)
  Endpoint: outputC[7] (output port clocked by clock)
  Path Group: clock
  Path Type: max
  Point                                            Incr     Path
---------------------------------------------------------------
  clock clock (rise edge)                          0.00     0.00
  clock network delay (ideal)                      0.00     0.00
  input external delay                             3.00     3.00 f
  inputA[1] (in)                                   0.00     3.00 f
  mul_15/mult/B[1]
(multiplier_timing_ent_DW02_mult_4_4_0)            0.00     3.00 f
  mul_15/mult/U20/Z (ND4P)                         0.72     3.72 r
  mul_15/mult/U59/Z (ND2I)                         0.12     3.84 f
  mul_15/mult/U98/Z (ND2I)                         0.25     4.10 r
  mul_15/mult/U99/Z (IVI)                          0.17     4.27 f
  mul_15/mult/U100/Z (AO3P)                        0.68     4.95 r
  mul_15/mult/U114/Z (ENI)                         0.48     5.43 f
  mul_15/mult/U108/Z (ND2I)                        0.25     5.68 r
  mul_15/mult/U41/Z (ND2I)                         0.26     5.95 f
  mul_15/mult/FS/B[2]
(multiplier_timing_ent_DW01_add_6_0)               0.005.95 f
  mul_15/FS/mult/U6/Z (AO3P)                       0.68     6.63 r
  mul_15/FS/mult/U20/Z (ND2I)                      0.20     6.83 f
  mul_15/FS/mult/U25/Z (ND2I)                      0.25     7.08 r
  mul_15/FS/mult/U26/Z (ND2I)                      0.12     7.20 f
  mul_15/FS/mult/U11/Z (ENI)                       0.38     7.58 f
  mul_15/mult/FS/SUM[5]
(multiplier_timing_ent_DW01_add_6_0)               0.00     7.58 f
  mul_15/mult/PRODUCT[7]
(multiplier_timing_ent_DW02_mult_4_4_0)            0.00     7.58 f
  outputC[7] (out                                  0.00     7.58 f
  data arrival time                                         7.58

  clock clock (rise edge)                         10.00    10.00
  clock network delay (ideal)                      0.00    10.00
  output external delay                           -2.70     7.30
  data required time                                        7.30
---------------------------------------------------------------
  data required time                                        7.30
  data arrival time                                        -7.58
---------------------------------------------------------------
  slack (VIOLATED)                                         -0.28
```

3. With a violation of 0.28 ns, all hierarchies in the design are torn down to create a flat design.

```
dc_shell> ungroup -all -flatten
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : multiplier_timing_ent
Version: 1998.02-1
Date   : Tue Mar 16 16:28:23 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design              Wire Loading Model          Library
-------------------------------------------------------
multiplier_timing_ent  05x05                     class

  Startpoint: inputA[1] (input port clocked by clock)
  Endpoint: outputC[7] (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                  Incr      Path
-------------------------------------------------------
  clock clock (rise edge)                0.00      0.00
  clock network delay (ideal)            0.00      0.00
  input external delay                   3.00      3.00 f
  inputA[1] (in)                         0.00      3.00 f
  mul_15/mult/U19/Z (AN2I)               0.63      3.63 f
  mul_15/mult/U18/Z (ENI)                0.42      4.05 f
  mul_15/mult/U109/Z (ND2I)              0.30      4.35 r
  mul_15/mult/U110/Z (ND2I)              0.12      4.47 f
  mul_15/mult/U29/Z (ENI)                0.42      4.89 f
  U32/Z (ENI)                            0.48      5.37 f
  U69/Z (ND2I)                           0.25      5.62 r
  U68/Z (ND2I)                           0.12      5.74 f
```

```
U77/Z (ND2I)                      0.25    6.00 r
U76/Z (IVI)                       0.12    6.12 f
U75/Z (ND2I)                      0.30    6.42 r
U54/Z (ND2I)                      0.12    6.54 f
mul_15/FS/mult/U25/Z (ND2I)       0.25    6.79 r
mul_15/FS/mult/U26/Z (ND2I)       0.12    6.91 f
mul_15/FS/mult/U11/Z (ENI)        0.38    7.29 f
outputC[7] (out)                  0.00    7.29 f
data arrival time                         7.29

clock clock (rise edge)          10.00   10.00
clock network delay (ideal)       0.00   10.00
output external delay            -2.70    7.30
data required time                        7.30
------------------------------------------------
data required time                        7.30
data arrival time                        -7.29
------------------------------------------------
slack (MET)                               0.01
```

The negative slack of –0.28 ns is fixed with logical flattening of the design and recompiling with `map_effort high` and `incremental_mapping` option.

8.3.4 Characterizing Submodules

`Characterize` is a very useful ***Design Compiler*** command for optimizing hierarchical designs. When a module is synthesized with constraints independently, it is able to meet timing requirements. When this same module is instantiated in a higher-level hierarchy, it may no longer be able to meet the design constraints of the higher-level hierarchy.

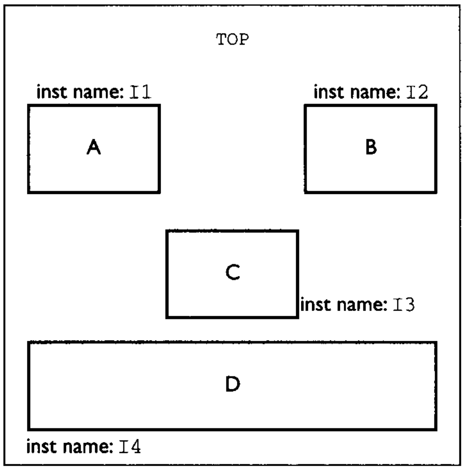


FIGURE 73 Diagram Showing Multiple Submodules on Module TOP.

From Fig. 73, submodule ***A, B*** and ***C*** are synthesized independently and all are able to meet their design constraints. However, when all three submodules are instantiated in a higher-level hierarchy of module ***TOP***, all three submodules may not necessarily meet timing requirements. This is especially true if the design constraints set on module ***TOP*** are tighter than those set on each submodule independently. Furthermore, there may be some glue logic on module ***TOP*** to glue the three submodules ***A, B*** and ***C*** together, which will also change the timing requirements for each of these submodules.

To overcome this problem of submodule usage in a higher-level hierarchy, the command ***characterize*** can come in handy. This command captures the boundary conditions of a submobile based on the environment of the higher-level hierarchy. The designer can then compile the submodule independently once this boundary condition of the submodule is captured.

```
dc_shell> current_design = TOP
dc_shell> characterize I1
dc_shell> current_design = I1
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> current_design = TOP
dc_shell> characterize I2
dc_shell> current_design = I2
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> current_design = TOP
dc_shell> characterize I3
dc_shell> current_design = I3
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> current_design = TOP
```

Appendix D shows the constraint synthesis of the pipeline microcontroller example of Chapter 6. Optimization of the microcontroller makes use of the command ***characterize***.

8.3.5 Register Balancing

Register balancing is a very useful command when it comes to optimizing designs that are made up of pipelines. The concept here is to allow ***Design Compiler*** to move logic from one stage of the pipeline to another. This would allow ***Design Compiler*** the flexibility to move logic away from pipeline stages that are overly constrained to pipeline stages that have additional timing.

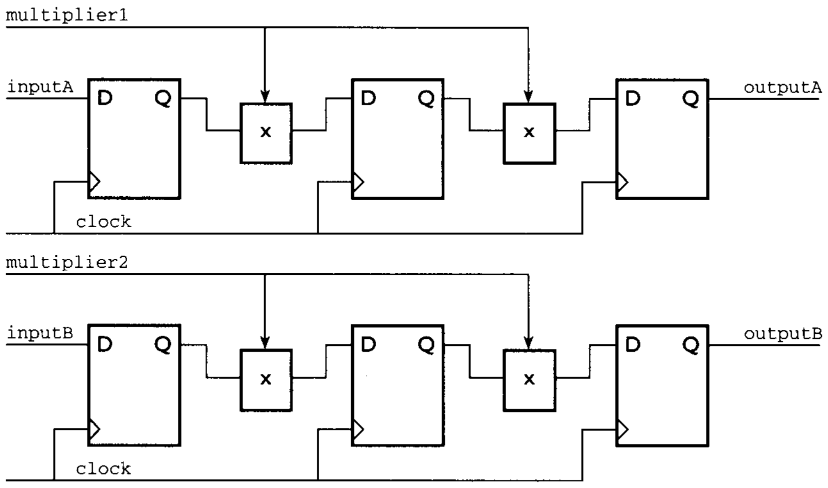


FIGURE 74 Diagram Showing a Pipeline Design.

Figure 74 shows a design with two pipes. Each pipe has three flip-flops and between each of these there is a multiplier. The first multiplier for each of these two pipes is a 4-bit multiplier while the second multiplier is an 8-bit multiplier. Therefore the combinational logic involved between the second and third flip-flop is twice that of the combinational logic between the first and second flip-flop.

EXAMPLE 53 Example of VHDL Code for a 2-Pipe Design

Filename: balance_buf.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY balance_reg_ent IS
PORT (
      clock : IN std_logic;
      multiplier1 : IN std_logic_vector(3 downto 0);
      multiplier2 : IN std_logic_vector (7 downto 0);
      inputA : IN std_logic_vector (3 downto 0);
      inputB : IN std_logic_vector (3 downto 0);
      outputA : OUT std_logic_vector (15 downto 0);
      outputB : OUT std_logic_vector (15 downto 0)
      );
END balance_reg_ent;

ARCHITECTURE balance_reg_arch OF balance_reg_ent IS
SIGNAL level1_inputA, level1_inputB:std_logic_vector(3 downto 0);
SIGNAL end_level1_inputA, end_level1_inputB : std_logic_vector (7
                                                       downto 0);
SIGNAL level2_inputA, level2_inputB:std_logic_vector(7 downto 0);
SIGNAL end_level2_inputA, end_level2_inputB : std_logic_vector
                                                  (15 downto 0);
BEGIN
      PROCESS (clock)
      BEGIN
            IF (clock = '1' AND clock'EVENT) THEN
                 level1_inputA <= inputA;
                 level1_inputB <= inputB;
                 level2_inputA <= end_level1_inputA;
                 level2_inputB <= end_level1_inputB;
                 outputA <= end_level2_inputA;
                 outputB <= end_level2_inputB;
            END IF;
      END PROCESS;

      end_level1_inputA <= signed(level1_inputA) *
                           signed(multiplier1);
      end_level1_inputB <= signed(level1_inputB) *
                           signed(multiplier1);
      end_level2_inputA <= signed(level2_inputA) *
                           signed(multiplier2);
      end_level2_inputB <= signed(level2_inputB) *
                           signed(multiplier2);

END balance_reg_arch;
```

1. Read in the VHDL file.

```
dc_shell> read -format vhdl balance_reg.vhd
```

2. Set the design constraint..

```
dc_shell> current_design = balance_reg_ent
dc_shell> create_clock -name clock -period 12
clock
dc_shell> set_input_delay 2 -clock clock
inputA*
dc_shell> set_input_delay 2 -clock clock
inputB*
dc_shell> set_input_delay 2 -clock clock
multiplier1
dc_shell> set_input_delay 2 -clock clock
multiplier2
```

3. Perform a ***map_effort medium*** compilation on the design.

```
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

4. The synthesized result shows a setup violation of 3.11 ns.

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : balance_reg_ent
Version: 1998.02-1
Date   : Wed Mar 17 09:55:51 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top
Wire Loading Model Mode: top

Design              Wire Loading Model          Library
-----------------------------------------------------------
balance_reg_ent         20x20                    class

  Startpoint: multiplier2[0]
              (input port clocked by clock)
  Endpoint: outputB_reg[15]
            (rising edge-triggered flip-flop clocked
             by clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr      Path
-----------------------------------------------------------
  clock clock (rise edge)                   0.00      0.00
  clock network delay (ideal)               0.00      0.00
  input external delay                      2.00      2.00 f
  multiplier2[0] (in)                       0.00      2.00 f
  mul_38/mult/A[0]
 (balance_reg_ent_DW02_mult_8_8_0)          0.00      2.00 f
  mul_38/mult/U188/Z (NR2I)                 0.65      2.65 r
  mul_38/mult/U52/Z (AN2I)                  0.43      3.08 r
  mul_38/mult/U454/Z (ND2I)                 0.22      3.29 f
  mul_38/mult/U300/Z (AO3P)                 0.75      4.04 r
 Point                                          Incr      Path
--------------------------------------------------------------
  mul_38/mult/U104/Z (ENI)                      0.51      4.55 f
  mul_38/mult/U491/Z (IVI)                      0.33      4.88 r
  mul_38/mult/U460/Z (ND2I)                     0.22      5.10 f
  mul_38/mult/U17/Z (AO3P)                      0.75      5.85 r
  mul_38/mult/U19/Z (IVI)                       0.15      5.99 f
  mul_38/mult/U50/Z (ENI)                       0.56      6.55 f
  mul_38/mult/U528/Z (MUX21L)                   0.88      7.44 r
  mul_38/mult/U172/Z (ENI)                      0.51      7.95 f
  mul_38/mult/U342/Z (ND2I)                     0.27      8.22 r
  mul_38/mult/U27/Z (AN2I)                      0.41      8.63 r
  mul_38/mult/U26/Z (MUX21LP)                   0.47      9.10 f
  mul_38/mult/U458/Z (ND2I)                     0.27      9.37 r
  mul_38/mult/U237/Z (ND2I)                     0.15      9.52 f
  mul_38/mult/U95/Z (NR2I)                      0.89     10.41 r
  mul_38/mult/FS/B9
(balance_reg_ent_DW01_add_14_0)                 0.00     10.41 r
  mul_38/mult/FS/U51/Z (IVI)                    0.23     10.64 f
  mul_38/mult/FS/U116/Z (ND2I)                  0.27     10.91 r
  mul_38/mult/FS/U25/Z (AN2I)                   0.36     11.27 r
  mul_38/mult/FS/U68/Z (ND2I)                   0.15     11.43 f
  mul_38/mult/FS/U69/Z (NR2I)                   0.65     12.07 r
  mul_38/mult/FS/U113/Z (ND2I)                  0.15     12.22 f
  mul_38/mult/FS/U72/Z (AN2I)                   0.62     12.84 f
  mul_38/mult/FS/U101/Z (ND2I)                  0.39     13.23 r
  mul_38/mult/FS/U79/Z (ND2I)                   0.15     13.38 f
  mul_38/mult/FS/U21/Z (ND2I)                   0.27     13.65 r
  mul_38/mult/FS/U19/Z (NR2I)                   0.22     13.87 f
  mul_38/mult/FS/U42/Z (ENI)                    0.44     14.31 f
  mul_38/mult/FS/SUM13
(balance_reg_ent_DW01_add_14_0)                 0.00     14.31 f
  mul_38/mult/PRODUCT15
(balance_reg_ent_DW02_mult_8_8_0)               0.00     14.31 f
  outputB_reg[15]/D (FD1)                       0.00     14.31 f
  data arrival time                            14.31
clock clock (rise edge)                        12.00     12.00
  clock network delay (ideal)                   0.00     12.00
  outputB_reg[15]/CP (FD1)                      0.00     12.00 r
  library setup time                           -0.80     11.20
  data required time                                     11.20
---------------------------------------------------------------
  data required time                                     11.20
  data arrival time                                     -14.31
---------------------------------------------------------------
  slack (VIOLATED)                                       -3.11
```

5. Execute the ***balance_registers*** command.

```
dc_shell> balance_registers
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : balance_reg_ent
Version: 1998.02-1
Date   : Wed Mar 17 09:59:39 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top

Design           Wire Loading Model          Library
-----------------------------------------------------
balance_reg_ent        20x20                   class

  Startpoint: balance_reg_ent_REG125_S1
             (rising edge-triggered flip-flop clocked
              by clock)
  Endpoint: balance_reg_ent_REG45_S1
           (rising edge-triggered flip-flop clocked
            by clock)
  Path Group: clock
  Path Type: max

  Point                                 Incr      Path
-----------------------------------------------------
  clock clock (rise edge)               0.00      0.00
  clock network delay (ideal)           0.00      0.00
  balance_reg_ent_REG125_S1/CP (FD1)    0.00      0.00 r
  balance_reg_ent_REG125_S1/Q (FD1)     2.63      2.63 r
  mul_35/mult/U29/Z (ND2I)              0.25      2.89 f
  mul_35/mult/U55/Z (IVI)               0.26      3.15 r
  mul_35/mult/U56/Z (ND2I)              0.42      3.57 f
  mul_35/mult/U31/Z (ND2I)              0.33      3.90 r
  mul_35/mult/U61/Z (IVI)               0.15      4.05 f
  mul_35/mult/U47/Z (ND2I)              0.33      4.37 r
  mul_35/mult/U70/Z (ND2I)              0.15      4.53 f
```

```
 Point                                    Incr     Path
------------------------------------------------------------
 mul_35/mult/U17/Z (ENI)                  0.51     5.04 f
 mul_35/mult/U89/Z (IVI)                  0.26     5.30 r
 mul_35/mult/U37/Z (ND2I)                 0.32     5.62 f
 mul_35/mult/U12/Z (NR2I)                 1.14     6.76 r
 mul_35/mult/FS/U10/Z (IVI)               0.15     6.90 f
 mul_35/mult/FS/U16/Z (ND2I)              0.33     7.23 r
 mul_35/mult/FS/U17/Z (IVI)               0.15     7.38 f
 mul_35/mult/FS/U15/Z (NR2I)              0.65     8.03 r
 mul_35/mult/FS/U18/Z (ND2I)              0.15     8.18 f
 mul_35/mult/FS/U12/Z (ND2I)              0.42     8.60 r
 mul_35/mult/FS/U20/Z (IVI)               0.15     8.75 f
 mul_35/mult/FS/U21/Z (ND2I)              0.27     9.02 r
 mul_35/mult/FS/U22/Z (ND2I)              0.15     9.17 f
 mul_35/mult/FS/U24/Z (ND2I)              0.27     9.44 r
 balance_reg_ent_REG45_S1/D (FD1)         0.00     9.44 r
 data arrival time                                 9.44

 clock clock (rise edge)                 12.00    12.00
 clock network delay (ideal)              0.00    12.00
 balance_reg_ent_REG45_S1/CP (FD1)        0.00    12.00 r
 library setup time                      -0.80    11.20
 data required time                               11.20
------------------------------------------------------------
 data required time                               11.20
 data arrival time                                -9.44
------------------------------------------------------------
 slack (MET)                                       1.76
```

6. Timing now improves from negative 3.11 ns to positive slack of 1.76 ns. ***Design Compiler*** has successfully moved logic from the second multiplier to the first multiplier by using the **balance_registers** command.

7. Appendix D, which shows Top-Down Synthesis of the pipeline microcontroller design example of Chapter 6, makes use of the command **balance_registers** as well.

8.3.6 Usage of FSM Compiler to Optimize Finite State Machine

FSM Compiler is an option within ***Design compiler*** that allows unique optimizations for finite state machines. ***FSM Compiler*** also gives a designer the flexibility to optimize a state machine design for speed performance or small area optimization.

In general, it is a good design practice for a designer to always partition a design that has state machines to be in an independent module. However, combining random logic with state machines is not a good partitioning practice.

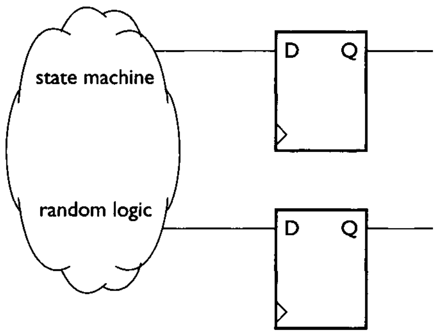


FIGURE 75 Diagram Showing a Nonpartitioned Design.

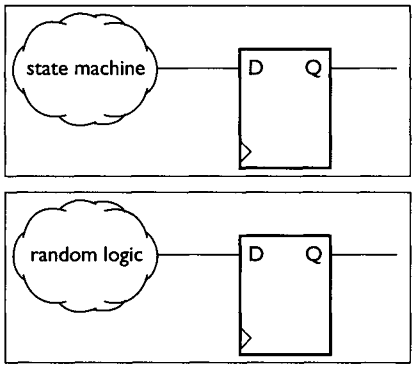


FIGURE 76 Diagram Showing a Well-Partitioned State Machine Logic and Random Logic.

By isolating a state machine from the rest of random logic, ***FSM compiler*** can be used to compile the state machine. This will allow the designer flexibility in choosing different forms of encoding on the state machine.

Figure 75 shows a good example of a badly partitioned design. In this diagram, state machine logic and random logic are clumped together in one piece. As a result, ***Design Compiler*** is unable to independently optimize random logic and state machine logic.

Figure 76 is a good example of good partitioning whereby a state machine is partitioned in a submodule separate from random logic. In this case, ***Design Compiler*** can be used to optimize the submodule containing the random logic and state machine logic independently.

To prepare a state machine design to be compiled using FSM, take the following steps:

1. Read in the VHDL file.

```
dc_shell> read -format vhdl <filename.vhd>
```

2. Map the design.

```
dc_shell> compile -map_effort medium
```

3. Group the logic of the state machine. This includes grouping of state vector flip-flops and their respective logic using ***group*** command. This step is not needed if your design is well partitioned to consist only of state machine logic.

```
dc_shell> set_fsm_state_vector {<flip-flop
name>, <flip-flop name>, ........}
dc_shell> group -fsm -design_name
<fsm_design_name>
```

4. Extract the finite state machine. This will extract the state machine from a netlist format into a state table format. The order of the flip-flops must be specified in the same order of state vector bits.

```
dc_shell> set_fsm_state_vector {<flip-flop
name>, <flip-flop name>, ........}
dc_shell> set_fsm_encoding {"S0=0", "S1=1",
"S2=2", .......}
dc_shell> extract
```

5. Write the design in FSM format

```
dc_shell> write -format st -output
my_state_machine.st
```

6. If you already have a state machine design in FSM format, you can directly read in the design

```
dc_shell> read -format st my_state_machine.st
```

7. Define the state ordering

```
dc_shell> set_fsm_order {S0, S1, S2, ....}
```

8. Define the encoding style.

```
dc_shell> set_fsm_encoding_style
<encoding_style>
```

<encoding_style> can be ***gray, binary, one-hot,*** or ***auto***. If ***auto*** is selected, ***Design Compiler*** will automatically generate any unassigned state vector encodings.

9. Remove any redundant states. This is optional.

```
dc_shell> set_fsm_minimize true
```

10. Compile the state machine.

```
dc_shell> compile -map_effort high
```

The following example shows the synthesis of the traffic controller state machine example of Chapter 5.4 with design constraints using FSM Compiler.

Filename: state_machine.vhd

```
dc_shell> read -format vhdl state_machine.vhd
dc_shell> current_design = state_machine_ent
dc_shell> create_clock EVALUATE -name clock -
period 5
dc_shell> set_input_delay 2 GREEN -clock clock
dc_shell> set_input_delay 2 RED -clock clock
dc_shell> set_input_delay 2 YELLOW -clock clock
dc_shell> set_output_delay 2 BRAKE -clock clock
dc_shell> set_output_delay 2 SPEED -clock clock
```

Compile the design with a ***map_effort medium*** option

```
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : state_machine_ent
Version: 1998.02-1
Date   : Tue Mar 16 15:40:45 1999
****************************************
```

```
Operating Conditions:
Wire Loading Model Mode: top

Design              Wire Loading Model            Library
-----------------------------------------------------------
state_machine_ent         05x05                    class

  Startpoint: RED (input port clocked by clock)
  Endpoint: BRAKE (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr      Path
-----------------------------------------------------------
  clock clock (rise edge)                   0.00      0.00
  clock network delay (ideal)               0.00      0.00
  input external delay                      2.00      2.00 f
  RED (in)                                  0.00      2.00 f
  U/Z (NR2I)                                0.57      2.57 r
  U38/Z (ND2I)                              0.19      2.76 f
  U53/Z (MUX21L)                            0.45      3.21 r
  U54/Z (IVI)                               0.07      3.28 f
  BRAKE (out)                               0.00      3.28 f
  data arrival time                         3.28

  clock clock (rise edge)                   5.00      5.00
  clock network delay (ideal)               0.00      5.00
  output external delay                    -2.00      3.00
  data required time                        3.00
-----------------------------------------------------------
  data required time                                  3.00
  data arrival time                                  -3.28
-----------------------------------------------------------
  slack (VIOLATED)                                   -0.28
```

With a setup violation of 0.28 ns, the design is recompiled with FSM compiler.

```
dc_shell> extract
dc_shell> set_fsm_encoding_style one_hot
dc_shell> set_fsm_order { S0 S1 S2 S3 }
dc_shell> set_fsm_encoding { "S0=2#1000"
"S1=2#0100"  "S2=2#0010"  "S3=2#0001" }
dc_shell> set_fsm_minimize true
dc_shell> compile  -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

The encoding style is set to one-hot to optimize the state machine to obtain the most optimal synthesis results in terms of speed. However, area utilization will increase with one-hot encoding.

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : state_machine_ent
Version: 1998.02-1
Date   : Tue Mar 16 15:40:52 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design           Wire Loading Model          Library
-----------------------------------------------------
state_machine_ent      05x05                  class

  Startpoint: GREEN (input port clocked by clock)
  Endpoint: BRAKE (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                               Incr      Path
-----------------------------------------------------
  clock clock (rise edge)             0.00      0.00
  clock network delay (ideal)         0.00      0.00
  input external delay                2.00      2.00 r
  GREEN (in)                          0.00      2.00 r
  U65/Z (IVI)                         0.12      2.12 f
  U73/Z (ND2I)                        0.29      2.41 r
  U64/Z (MUX21LP)                     0.45      2.86 f
  U71/Z (ND2I)                        0.21      3.07 r
  BRAKE (out)                         0.00      3.07 r
  data arrival time                   3.07

  clock clock (rise edge)             5.00      5.00
  clock network delay (ideal)         0.00      5.00
  output external delay              -2.00      3.00
  data required time                            3.00
-----------------------------------------------------
  data required time                            3.00
  data arrival time                            -3.07
-----------------------------------------------------
  slack (VIOLATED)                             -0.07
```

A slack of –0.28 ns is reduced to –0.07 ns by using ***FSM Compiler*** to recompile the state machine using one-hot encoding.

8.3.7 Choosing High-Speed Implementation for High-level Functional Module

When coding VHDL, the designer can always use reserved symbols/keywords to infer high-level functional modules. For example, to infer an adder, the designer need not write the code for it. The symbol '+' will allow ***Design Compiler*** to infer an adder. However, there are four different implementations for an adder.

TABLE 27 Description of Adder-Implementation Type

Implementation type	*Description*
rpl	Ripple carry
cla	Carry look ahead
clf	Fast carry look ahead
sim	Simulation model

The implementation type ***sim*** is only for simulation. Implementation type ***rpl, cla*** and ***clf*** are for synthesis; ***clf*** is the fastest implementation followed by ***cla***; the slowest is ***rpl***.

During synthesis, if compilation of ***map_effort low*** is used, implementation selection of the adder will not change from current choice. However, if compilation of ***map_effort medium*** and above is used, implementation selection is based on the optimization algorithm. Therefore, a designer should always try to use ***map_effort medium*** and above in order to allow ***Design Compiler*** to choose different implementations based on the optimizations of the design.

If compilation of ***map_effort low*** is used, the designer can still manually change the implementation selection by setting the variable ***set_implementation***.

```
dc_shell> set_implementation
<implementation_type> <cell_list>
```

8.3.8 Balancing of Logic Trees with Heavy Loading

The concept of balancing of logic trees is very useful in designs that have very heavy loadings. This is especially true if the design consists of any net that has a huge fanout.

EXAMPLE 54 VHDL Code of an Inverter

Filename: balance_buf.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY balance_buf_ent IS
PORT (
      inputA : IN std_logic;
      outputA1 : OUT std_logic;
      outputA2 : OUT std_logic
      );
END balance_buf_ent;

ARCHITECTURE balance_buf_arch OF balance_buf_ent IS
BEGIN
      outputA1 <= not inputA;
      outputA2 <= not inputA;
END balance_buf_arch;
```

From Example 54, **outputA1** and **outputA2** are both inverted signals of inputA.

```
c_shell> read -format vhdl balance_buf.vhd
dc_shell> current_design = balance_buf_ent
dc_shell> create_clock -name clock -period 5
dc_shell> set_input_delay 1.5 -clock clock
inputA*
dc_shell> set_output_delay 2.0 -clock clock
output*
dc_shell> set_wire_load 20x20
dc_shell> compile -map_effort medium
```

Synthesis results are shown in Fig. 77.

Use the command **report_net** to obtain a report on the net for this design.

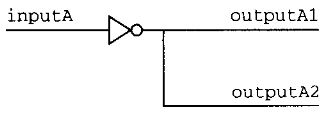


FIGURE 77 Diagram Showing Synthesized Circuit for **balance_buf_ent**.

```
dc_shell> report_net
```

```
Information: Updating design information... (UID-85)
****************************************
Report : net
Design : balance_buf_ent
Version: 1998.02-1
Date   : Tue Mar 23 17:23:01 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design             Wire Loading Model           Library
------------------------------------------------------------
balance_buf_ent        20x20                      class

Net           Fanout  Fanin   Load  Resistance  Pins
Attributes
------------------------------------------------------------
inputA             1      1   1.86        0.00     2
outputA2           2      1   1.41        0.00     3
------------------------------------------------------------
Total 2 nets       3      2   3.27        0.00     5
Maximum            2      1   1.86        0.00     3
Average         1.50   1.00   1.63        0.00  2.50
```

The loading on the output net is not too critical. But now let us assume the loading on net ***outputA2*** to be a large number, probably due to a large fanout or a long interconnect layout routing.

```
dc_shell> set_load 100 outputA2
```

The inverter synthesized by ***Design Compiler*** as shown in Fig. 77 will not be able to drive such a large loading, which has now been set on the net ***outputA2***. The delay through this net would be large due to the heavy loading.

A simple solution that ***Design Compiler*** offers to fix problems such as these are through the use of ***balance_buffer*** command. This command 'informs' ***Design Compiler*** that a net tree needs to be balanced. Then ***Design Compiler*** will create buffer trees to drive the large loading. Figure 78 shows the synthesized results after ***balance_buffer*** command is executed.

```
dc_shell> balance_buffer -depth 2 -to outputA2
dc_shell> report_net
```

```
Information: Updating design information... (UID-85)
****************************************
Report : net
Design : balance_buf_ent
Version: 1998.02-1
Date   : Tue Mar 23 17:23:01 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design              Wire Loading Model          Library
-----------------------------------------------------------
balance_buf_ent            20x20                  class

Net            Fanout  Fanin      Load  Resistance   Pins
Attributes
-----------------------------------------------------------
inputA              1      1      1.86        0.00      2
n3                  2      1      5.41        0.00      3
outputA1            1      1    100.86        0.00      2
outputA2            1      1    100.86        0.00      2
-----------------------------------------------------------
Total 4 nets        5      4    208.99        0.00      9
Maximum             2      1    100.86        0.00      3
Average          1.25   1.00     52.25        0.00   2.25
```

Figure 78 is different from Fig. 77 in the sense that two additional levels of inverter are added to the output of the first inverter. This would allow **inputA** to be able to drive the heavy loading on net ***outputA1*** and ***outputA2***.

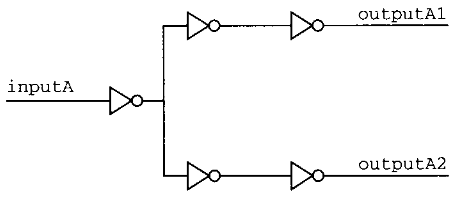


FIGURE 78 Diagram Showing Synthesized Circuit for **balance_buf_ent** After **balance_buffer**.

8.4 AREA OPTIMIZATION IN SYNTHESIS TWEAKS

Area optimization is obtained in synthesis through logic sharing. ***Design Compiler*** will always optimize a design with timing requirements as the highest priority followed by area. Several guidelines to obtain synthesis results with optimized area are as follows:

- do not use combinational logic as individual blocks;
- do not use glue logic between modules; and
- ***set_max_area*** attribute

8.4.1 Do Not Use Combinational Logic as Individual Blocks

Figure 79 shows ***module B*** as an independent block and separated from ***module A*** and ***module C***. This is not a good method to partition your design ***as Design Compiler*** is not able to optimize ***logic Y*** in ***module B*** with ***logic X*** in ***module A*** or ***logic Z*** in ***module C***. ***Design Compiler*** does not optimize, remove or add interface ports to a design module. Therefore, it is unable to automatically break the hierarchy of ***modules A, B and C*** to optimize ***logic X, Y and Z***. Figure 80 shows a better approach to partition your design. The partitioning in Fig. 80 will enable ***Design Compiler*** to optimize ***logic X, Y and Z***.

A better-partitioned design in Fig. 80 removes ***module B*** and combines ***logic X, Y and Z*** into a single combinational logic in ***module A***. With this partitioning, combined logic of ***X + Y + Z*** can be optimized by ***Design Compiler*** to create a faster and smaller design.

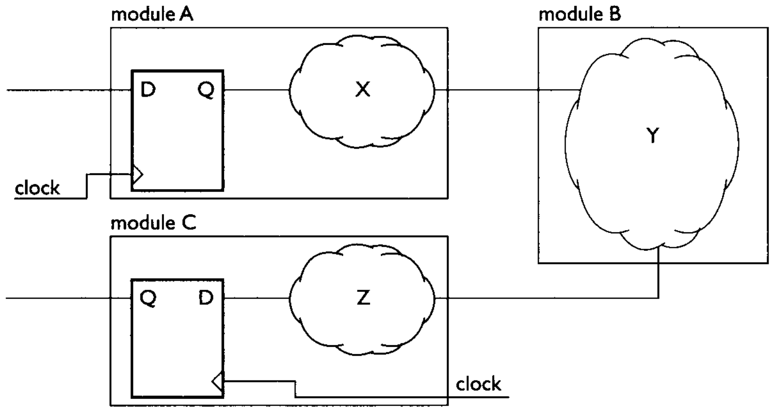


FIGURE 79 Diagram Showing an Independent Combinational Logic Block.

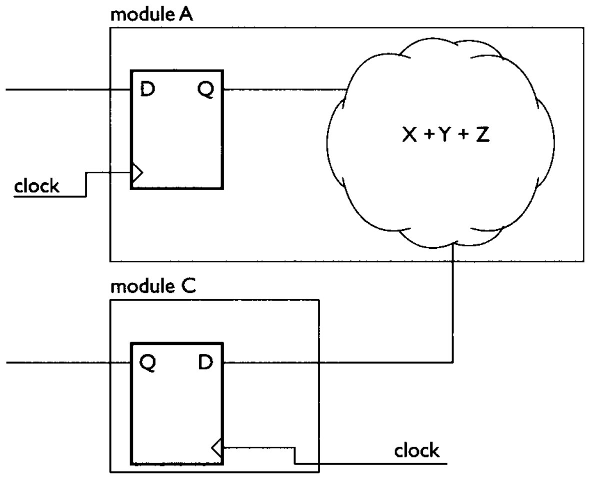


FIGURE 80 Diagram Showing Combinational Logic X, Y and Z Combined.

8.4.2 Do Not Use Glue Logic Between Modules

As seen in Fig. 79, if ***module B*** is removed and combinational ***logic Y*** is replaced with a logic gate, this logic gate is referred to as ***glue logic***. Its name comes from the fact that this logic gate 'glues' ***module A*** and ***module C*** together.

The glue logic between ***module A*** and ***module C*** cannot be optimized into combinational ***logic X*** or combinational ***logic Z***. The boundary of ***module A*** and ***module C*** prevents ***Design Compiler*** from optimizing the glue logic into either ***logic X*** or ***logic Z***. To overcome this problem, the designer can use ***group*** command to create a new block that consists of ***module A*** and the glue logic or ***module C*** and the glue logic.

To group the ***glue logic*** with ***module A***:

```
dc_shell> group {I1 I2}  -design_name module_B -
cell_name I4
```

This will create a new module called ***module_B***, which consists of ***module A*** and the ***glue logic***.

To group the ***glue logic*** with ***module C***:

```
dc_shell> group {I2 I3}  -design_name module_B -
cell_name I4
```

This will create a new module called ***module_B***, which consists of ***module C*** and the ***glue logic***.

Once the ***glue logic*** is combined with either ***module A*** or ***module C***, an incremental compilation can be performed to allow ***Design Compiler*** to try to optimize away the glue logic.

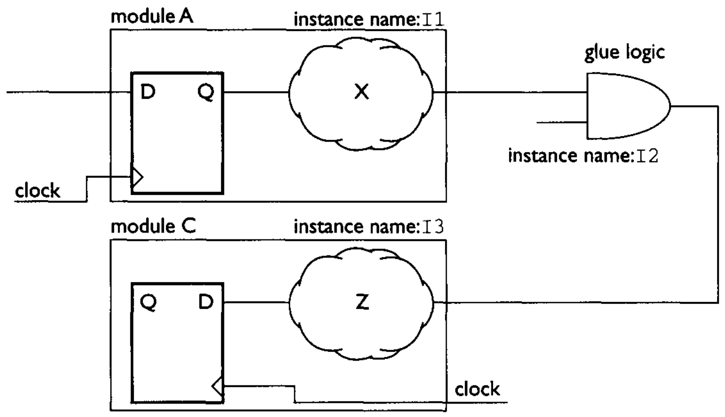


FIGURE 81 Diagram Showing Glue Logic Between Module A and Module C.

8.4.3 *set_max_area* Attribute

The ***set_max_area*** attribute is used by ***Design Compiler*** during synthesis optimization to obtain minimum area utilization. However, ***Design Compiler*** will not optimize the area involving paths that have negative slack. ***Design Compiler*** gives timing requirement the highest priority. Only when timing is met can ***Design Compiler*** optimize for area.

```
dc_shell> set_max_area 0.0
dc_shell> compile -map-effort high
```

8.5 FIXING HOLD-TIME VIOLATIONS IN SYNOPSYS

If a synthesized design has hold violations, the designer can set the attribute ***set_fix_hold*** to have ***Design Compiler*** fix the hold violations.

Fixing hold violations is a lot simpler in ***Design Compiler*** compared to fixing setup violations. Synthesis tweaks and sometimes microarchitectural implementation tweaks are needed to fix setup violations but hold violations, are fixed automatically by Synopsys by setting the ***set_fix_hold*** attribute.

```
dc_shell> set_fix_hold <clock_name>
dc_shell> compile -map_effort high -
incremental_mapping
```

During optimization, ***Design Compiler*** will insert delay at registers that are fed by ***<clock_name>***.

8.6 MISC SYNTHESIS COMMANDS GENERALLY USED

There are many synthesis commands for Synopsys's ***Design Compiler***. This chapter is a brief discussion of some of the commonly used synthesis commands.

- ***set_false_path***
  This command disables maximum and minimum timing checks on the mentioned path. It is used only for paths that are false and need not be considered in timing checks.

  From Fig. 82, assume that ***reset*** signal is a false path:

```
dc_shell> set_false_path -through reset
```

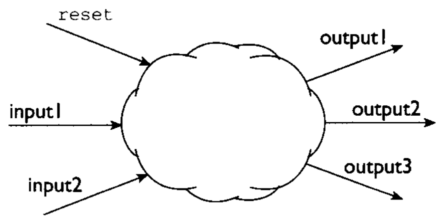


FIGURE 82 Diagram Showing **RESET** as a False Path.

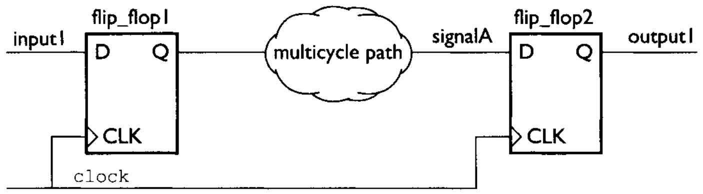


FIGURE 83 Diagram Showing Multicycle Path.

- ***set_multicycle_path***
  Multicycle paths are paths that consist of logic that has delays of more than one clock cycle. When a design has multicycle paths, it is important for the designer to use the command ***set_multicycle_path*** to set those paths to multicycle. This would 'inform' ***Design Compiler*** that a certain path that is failing timing checks is in fact a multicycle path and hence ***Design Compiler*** can spend more effort on fixing other critical nonmulticycle paths.

  From Fig. 83, to set ***signalA*** as a multicycle path that requires 2 cycles:

```
dc_shell> set_multicycle_path 2 -from flip-
flop1/CLK -to flip_flop2/D
```

- ***set_min_delay & set_max_delay***
  If the designer knows for sure the minimum delay required from a start-point to an end-point of a path, that delay can be set using ***set_min_delay*** command.

  From Fig. 83:

```
dc_shell> set_min_delay <minimum_delay> -from
flip_flop1/CLK -to flip_flop2/D
```

  Similarly, if a designer knows for sure the maximum delay required from a start point to an end point of a path, that delay can be set using ***set_max_delay*** command.

  From Fig. 83:

```
dc_shell> set_max_delay <maximum_delay> -from
flip_flop1/CLK -to flip_flop2/D
```

- ***set_max_transition***
  This command puts a limit on the maximum transition time allowed for a net. ***Design Compiler*** will optimize the design and attempt to ensure that each net

has a transition time less than that which is specified in ***set_max_transition*** command:

```
dc_shell> set_max_transition <time_value>
<list_of_ports_or_designs>
```

For example, to set max transition of 1.0 ns on all design in current design view,

```
dc_shell> design_name = find (design, "*")
dc_shell> set_max_transition 1.0 design_name
```

- ***set_max_fanout*** & ***set_fanout_load***
  The command ***set_max_fanout*** sets the maximum number of fanout allowed on an input port of a design:

```
dc_shell> set_max_fanout <fanout_number>
<list_of_input_ports_or_designs>
```

As an example, to set a maximum fanout of 5 on all inputs of all designs in current design view,

```
dc_shell> design_name = find (design, "*")
dc_shell> set_max_fanout 5 design_name
```

To set the fanout load on an output port of a design, use the command ***set_fanout_load***:

```
dc_shell> set_fanout_load <fanout_number>
<list_of_output_port_names>
```

In this example, to set a fanout load of 5 on all output ports in current design view,

```
dc_shell> set_fanout_load 5 all_outputs()
```

- ***set_max_capacitance***
  This command allows the designer to set maximum capacitance on a given design or port. It is similar to ***set_max_transition*** but it uses the total capacitance on a net as cost and not transition time:

```
dc_shell> set_max_capacitance
<capacitance_value>
<list_of_port_or_design_name>
```

Here, to set max capacitance of 5 on all designs in current design view,

```
dc_shell> design_name = find (design, "*")
dc_shell> set_max_capacitance 5 design_name
```

- ***set_dont_touch***
  This command is used when any design, cell or net is not to be changed by Design Compiler during optimization. However, the designer needs to be careful when using this command. Synthesis results may not be the most optimal when this command is used:

```
dc_shell> set_dont_touch
<list_of_cells_nets_or_designs>
```

Thus to not allow ***Design Compiler*** to change the clock network:

```
dc_shell> set_dont_touch clock
```

- ***report_lib***
  This command makes a report on the library that is stated with the command. The listed command will make a report on all conditions in the library class. Information such as operating conditions, wireload models, cells and others are shown:

```
dc_shell> report_lib class
```

```
****************************************
Report : library
Library: class
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************

Library Type              : Technology
Tool Created              : 1998.02
Date Created              : February 7, 1992
Library Version           : 1.800000
Time Unit                 : 1ns
Capacitive Load Unit      : 0.100000ff
Pulling Resistance Unit   : 1kilo-ohm
Voltage Unit              : 1V
Current Unit              : 1µA
Bus Naming Style          : %s[%d] (default)
```

```
Operating Conditions:

Name   Library  Process Temp    Volt  Interconnect Model
---------------------------------------------------------
WCCOM   class    1.50    70.00  4.75    worst_case_tree
WCIND   class    1.50    85.00  4.75    worst_case_tree
WCMIL   class    1.50  125.00  4.50    worst_case_tree


Input Voltages:

Name             Vil      Vih      Vimin      Vimax
---------------------------------------------------------
CMOS_SCHMITT     1.00     4.00     -0.30      VDD + 0.300

Output Voltages:

Name             Vol      Voh      Vomin      Vomax
---------------------------------------------------------
TTL              0.40     2.40     -0.30      VDD + 0.300

default_wire_load_capacitance: 1.000000
default_wire_load_resistance:  1.000000
default_wire_load_area:        1.000000



Wire Loading Model:

Name            :   05x05
Location        :   class
Resistance      :   0
Capacitance     :   1
Area            :   0
Slope           :   0.186

Fanout   Length   Points Average Cap Std Deviation
---------------------------------------------------------
     1     0.39

Name            :   10x10
Location        :   class
Resistance      :   0
Capacitance     :   1
Area            :   0
Slope           :   0.311

Fanout   Length   Points Average Cap Std Deviation
---------------------------------------------------------
     1     0.53
```

```
Name            :   20x20
Location        :   class
Resistance      :   0
Capacitance     :   1
Area            :   0
Slope           :   0.547

Fanout   Length   Points Average Cap Std Deviation
---------------------------------------------------------
     1     0.86

Wire Loading Model Selection Group:

Name            : class


Selection                       Wire load name

min area      max area
-----------------------------------------
    0.00       1000.00          05x05
 1000.00       2000.00          10x10
 2000.00       3000.00          20x20

Wire Loading Model Mode: top.


Wire Loading Model Selection Group: class.


Porosity information:
    No porosity information specified.


In_place optimization mode:  match_footprint


Timing Ranges:
    No timing ranges specified.


Components:
    Attributes:
        af - active falling
        ah - active high
        al - active low
        ar - active rising
         b - black box (function unknown)
        ce - clock enable
         d - dont_touch
        mo - map_only
         p - preferred
         r - removable
        s - statetable
      sa0 - dont_fault stuck-at-0
      sa1 - dont_fault stuck-at-1
     sa01 - dont_fault both stuck-at-0 and stuck-at-1
        t - test cell
        u - dont_use

Cell            Footprint       Attributes
---------------------------------------------
AN2             "an2"
AN2I            "an2"
AN2P
.......
.......
OR4
OR4P
```

- ***report_area***
  This command reports the number of references, nets, ports and cells in the current design view. It also reports the area composed of combinational logic, noncombinational logic and net interconnect area.

Example 55 shows synthesizable VHDL code for a 32-bit comparator.

EXAMPLE 55 Synthesizable VHDL Code for a 32-Bit Comparator

Filename: comparator.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY comparator_ent IS
PORT (
      inputA : IN std_logic_vector (31 downto 0);
      inputB : IN std_logic_vector (31 downto 0);
      output : OUT std_logic
      );
END comparator_ent;

ARCHITECTURE comparator_arch OF comparator_ent IS
BEGIN
      output <= '0' WHEN (inputA /= inputB)
      ELSE '1';
END comparator_arch;
```

This module is synthesized using a set of design constraints.

```
dc_shell> read -format vhdl comparator.vhd
dc_shell> current_design = comparator_ent
dc_shell> create_clock -name clock -period 10
dc_shell> set_input_delay 3.0 -clock clock
inputA*
dc_shell> set_input_delay 3.0 -clock clock
inputB*
dc_shell> set_output_delay 3.0 -clock clock
output
dc_shell> compile -map_effort medium
dc_shell> report_area
```

```
Information: Updating design information... (UID-85)
****************************************
Report : area
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************


Library(s) Used:

    class (File: /synopsys/libraries/syn/class.db)

Number of ports:              65
Number of nets:              127
Number of cells:              63
Number of references:          4

Combinational area:       128.000000
Noncombinational area:      0.000000
Net Interconnect area:     undefined  (Wire load has
                                        zero net area)

Total cell area:          128.000000
Total area:                undefined
```

From the report obtained, notice that the noncombinational area is zero while the combinational area is 128 units. This means that the design **comparator_ent** consists only of combinational logic. There is no noncombinational logic in the synthesized design. The report also shows the net interconnect area to be zero. This value of zero occurs because there was no net area information in the wireload model used for the design. Wireloads are net load models created by the designer using back-annotated layout information. Wireload models are discussed in detail in Chapter 13.

- ***report_cell***

  report_cell command lists all the cells and subdesigns that are present in the current design view. Using the same comparator as in Example 55, the command ***report_cell*** makes a report on all the cells in the current design of ***comparator_ent***.

```
dc_shell> report_cell
```

```
****************************************
Report : cell
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************


Attributes:
        b - black box (unknown)
        h - hierarchical
        n - noncombinational
        r - removable
        u - contains unmapped logic

Cell          Reference       Library     Area
Attributes
-----------------------------------------------------
U6            AN2I            class       2.00
U7            ND2I            class       1.00
U8            ND2I            class       1.00
U9            NR2I            class       1.00
U10           ND2I            class       1.00
U11           ND2I            class       1.00
U12           NR2I            class       1.00
U13           ND2I            class       1.00
.........
.........
.........
U61           ENI             class       3.00
U62           ENI             class       3.00
U63           ENI             class       3.00
U64           ENI             class       3.00
U65           ENI             class       3.00
U66           ENI             class       3.00
U67           ENI             class       3.00
U68           ENI             class       3.00
-----------------------------------------------------
Total 63 cells                          128.00
```

The first column of the report is the name of the cells in the design, the second column is the reference name of the cell, the third column is the library name and the final column is the area of the cell.

This command can also be used if the designer only wishes to have a report on certain cells.

For example, if a report of cell **U32** is needed, the list of cells would be **U32**.

```
dc_shell> report_cell U32
```

```
****************************************
Report : cell
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************

Attributes:
       b - black box (unknown)
       h - hierarchical
       n - noncombinational
       r - removable
       u - contains unmapped logic

Cell         Reference      Library    Area
Attributes
-------------------------------------------------------
U32          ND2I           class      1.00
-------------------------------------------------------
Total 1 cells                          1.00
```

report_cell command is not only confined to obtaining general information of a cell or a list of cells, but it is also very useful when a designer wants to know the detailed interconnection information of the cell.

```
dc_shell> report_cell -connections U31
```

```
****************************************
Report : cell
       -connections
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************
```

```
Connections for cell 'U31':
    Reference:          NR2I
    Library:            class

    Input Pins          Net
----------------------------------------
    A                   n52
    B                   n55

    Output Pins         Net
----------------------------------------
    Z                   n58
```

The report shows that cell **U31** is a two-input NOR gate (**2NR** is the cell **U31** reference name indicating a two-input NOR gate) with input pin *A* connected to net **n52** and input pin *B* connected to net **n55** while output pin *Z* is connected to net **n58**.

- ***report_net***
  The command **report_net** makes a report of all nets in the current design of **comparator_ent**. Command **report_net** options are similar to those of **report_cell**.

```
dc_shell> report_net <list_of_nets>
```

will report all the nets that are listed in the **<list_of_nets>**.

```
dc_shell> report_net <net> -connections
```

will report the connections of the **<net>**.

To obtain a report of all the nets in the current design:

```
dc_shell> report_net
```

```
****************************************
Report : net
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top
```

```
Design           Wire Loading Model           Library
-----------------------------------------------------
comparator_ent        05x05                     class

Net           Fanout  Fanin   Load  Resistance  Pins
Attributes
-----------------------------------------------------
inputA[0]          1      1   1.39        0.00     2
inputA[1]          1      1   1.39        0.00     2
inputA[2]          1      1   1.39        0.00     2
inputA[3]          1      1   1.39        0.00     2
inputA[4]          1      1   1.39        0.00     2
inputA[5]          1      1   1.39        0.00     2
........
........
inputB[0]          1      1   2.39        0.00     2
inputB[1]          1      1   2.39        0.00     2
inputB[2]          1      1   2.39        0.00     2
inputB[3]          1      1   2.39        0.00     2
inputB[4]          1      1   2.39        0.00     2
inputB[5]          1      1   2.39        0.00     2
........
........
n6                 1      1   1.39        0.00     2
n7                 1      1   1.39        0.00     2
n8                 1      1   1.39        0.00     2
n9                 1      1   1.39        0.00     2
n10                1      1   1.39        0.00     2
........
........
n63                1      1   1.39        0.00     2
n64                1      1   1.39        0.00     2
n65                1      1   1.39        0.00     2
n66                1      1   1.39        0.00     2
n67                1      1   1.39        0.00     2

output             1      1   0.39        0.00     2
-----------------------------------------------------
Total 127 nets   127    127 207.53        0.00   254
Maximum            1      1   2.39        0.00     2
Average         1.00   1.00   1.63        0.00  2.00
```

From the report, the fourth column shows the loading on the net. The value of this column changes with back-annotated information from layout. The fifth column shows the resistance of the net. The current report shows resistance of zero as there is currently no back-annotated information from layout.

- ***report_timing***
  This command is especially useful when performing timing analysis on a synthesized design. Using the same design ***comparator_ent*** of Example 55, different timing reports are obtained through different options of this command.

  To obtain a timing report of maximum delay:

```
dc_shell> report_timing -delay max
```

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design           Wire Loading Model          Library
------------------------------------------------------
comparator_ent         05x05                   class

  Startpoint: inputA[17] (input port clocked by clock)
  Endpoint: output (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                    Incr     Path
------------------------------------------------------
  clock clock (rise edge)                  0.00     0.00
  clock network delay (ideal)              0.00     0.00
  input external delay                     3.00     3.00 r
  inputA[17] (in)                          0.00     3.00 r
  U58/Z (ENI)                              0.35     3.35 r
  U25/Z (ND2I)                             0.12     3.47 f
  U27/Z (NR2I)                             0.57     4.05 r
  U28/Z (ND2I)                             0.12     4.17 f
  U36/Z (NR2I)                             0.57     4.74 r
  U6/Z (AN2I)                              0.30     5.03 r
  output (out)                             0.00     5.03 r
  data arrival time                                 5.03
  clock clock (rise edge)            10.00    10.00
  clock network delay (ideal)         0.00    10.00
  output external delay              -3.00     7.00
  data required time                           7.00
-----------------------------------------------------
  data required time                           7.00
  data arrival time                           -5.03
-----------------------------------------------------
  slack (MET)                                  1.97
```

To obtain a timing report of the two worst paths:

```
dc_shell> report_timing -nworst 2
```

```
****************************************
Report : timing
        -path full
        -delay max
        -nworst 2
        -max_paths 2
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top

Design          Wire Loading Model          Library
------------------------------------------------------
comparator_ent        05x05                  class

  Startpoint: inputA[17] (input port clocked by clock)
  Endpoint: output (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                               Incr      Path
-----------------------------------------------------
  clock clock (rise edge)             0.00      0.00
  clock network delay (ideal)         0.00      0.00
  input external delay                3.00      3.00 r
  inputA[17] (in)                     0.00      3.00 r
  U58/Z (ENI)                         0.35      3.35 r
  U25/Z (ND2I)                        0.12      3.47 f
  U27/Z (NR2I)                        0.57      4.05 r
  U28/Z (ND2I)                           0.12     4.17 f
  U36/Z (NR2I)                           0.57     4.74 r
  U6/Z (AN2I)                            0.30     5.03 r
  output (out)                           0.00     5.03 r
  data arrival time                               5.03

  clock clock (rise edge)               10.00    10.00
  clock network delay (ideal)            0.00    10.00
  output external delay                 -3.00     7.00
  data required time                              7.00
-----------------------------------------------------------
  data required time                              7.00
  data arrival time                              -5.03
-----------------------------------------------------------
  slack (MET)                                     1.97



  Startpoint: inputA[17] (input port clocked by clock)
  Endpoint: output (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                   Incr     Path
-----------------------------------------------------------
  clock clock (rise edge)                0.00     0.00
  clock network delay (ideal)            0.00     0.00
  input external delay                   3.00     3.00 f
  inputA[17] (in)                        0.00     3.00 f
  U58/Z (ENI)                            0.35     3.35 r
  U25/Z (ND2I)                           0.12     3.47 f
  U27/Z (NR2I)                           0.57     4.05 r
  U28/Z (ND2I)                           0.12     4.17 f
  U36/Z (NR2I)                           0.57     4.74 r
  U6/Z (AN2I)                            0.30     5.03 r
  output (out)                           0.00     5.03 r
  data arrival time                               5.03

  clock clock (rise edge)               10.00    10.00
  clock network delay (ideal)            0.00    10.00
  output external delay                 -3.00     7.00
  data required time                              7.00
-----------------------------------------------------------
  data required time                              7.00
  data arrival time                              -5.03
-----------------------------------------------------------
  slack (MET)                                     1.97
```

To obtain a timing report from one input to another:

```
dc_shell> report_timing -from inputA[0] -to
output
```

```
Performing report_timing on port 'inputA[0]'.
Performing report_timing on port 'output'.

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : comparator_ent
Version: 1998.02-1
Date   : Fri Mar 26 23:09:44 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design          Wire Loading Model          Library
-----------------------------------------------------
comparator_ent        05x05                   class

  Startpoint: inputA[0] (input port clocked by clock)
  Endpoint: output (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                  Incr      Path
-----------------------------------------------------
  clock clock (rise edge)                0.00      0.00
  clock network delay (ideal)            0.00      0.00
  input external delay                   3.00      3.00 r
  inputA[0] (in)                         0.00      3.00 r
  U47/Z (ENI)                            0.35      3.35 r
  U15/Z (ND2I)                           0.12      3.47 f
  U16/Z (NR2I)                           0.57      4.05 r
  U20/Z (ND2I)                           0.12      4.17 f
  U21/Z (NR2I)                           0.57      4.74 r
  U6/Z (AN2I)                            0.30      5.03 r
  output (out)                           0.00      5.03 r
  data arrival time                                5.03

  clock clock (rise edge)               10.00     10.00
  clock network delay (ideal)            0.00     10.00
  output external delay                  -3.00    7.00
  data required time                              7.00
-----------------------------------------------------------
  data required time                              7.00
  data arrival time                              -5.03
-----------------------------------------------------------
  slack (MET)                                     1.97
```

- ***find***

  The command ***find*** is another very useful command that a designer can use when searching for a certain element such as cell, net, reference, design, pin, and even library.

  To find nets in current design that start with the alphabet ***'n'***:

```
dc_shell> find (net, "n*")
```

```
{"n6", "n7", "n8", "n9", "n10", "n11", "n12", "n13",
"n14", "n15", "n16", "n17", "n18", "n19", "n20",
"n21", "n22", "n23", "n24", "n25", "n26", "n27",
"n28", "n29", "n30", "n31", "n32", "n33", "n34",
"n35",
   "n36", "n37", "n38", "n39", "n40", "n41", "n42",
"n43", "n44", "n45", "n46", "n47", "n48", "n49",
"n50", "n51", "n52", "n53", "n54", "n55", "n56",
"n57", "n58", "n59", "n60", "n61", "n62", "n63",
"n64", "n65", "n66", "n67"}
```

  To find cells in current design that starts with the alphabet ***'U'***:

```
dc_shell> find (cell, "U*")
```

```
{"U6", "U7", "U8", "U9", "U10", "U11", "U12", "U13",
"U14", "U15", "U16", "U17", "U18", "U19", "U20",
"U21", "U22", "U23", "U24", "U25", "U26", "U27",
"U28", "U29", "U30", "U31", "U32", "U33", "U34",
"U35",
   "U36", "U37", "U38", "U39", "U40", "U41", "U42",
"U43", "U44", "U45", "U46", "U47", "U48", "U49",
"U50", "U51", "U52", "U53", "U54", "U55", "U56",
"U57", "U58", "U59", "U60", "U61", "U62", "U63",
"U64", "U65", "U66", "U67", "U68"}
```

To find all ports in current design:

```
dc_shell> find (port, input*)
```

```
{"inputA[31]", "inputA[30]", "inputA[29]",
"inputA[28]", "inputA[27]", "inputA[26]",
"inputA[25]", "inputA[24]", "inputA[23]",
"inputA[22]", "inputA[21]", "inputA[20]",
"inputA[19]", "inputA[18]", "inputA[17]",
"inputA[16]", "inputA[15]", "inputA[14]",
"inputA[13]", "inputA[12]", "inputA[11]",
"inputA[10]", "inputA[9]", "inputA[8]",
"inputA[7]", "inputA[6]", "inputA[5]", "inputA[4]",
"inputA[3]", "inputA[2]",
"inputA[1]", "inputA[0]", "inputB[31]",
"inputB[30]", "inputB[29]", "inputB[28]",
"inputB[27]", "inputB[26]", "inputB[25]",
"inputB[24]", "inputB[23]", "inputB[22]",
"inputB[21]", "inputB[20]", "inputB[19]",
"inputB[18]", "inputB[17]", "inputB[16]",
"inputB[15]", "inputB[14]", "inputB[13]",
"inputB[12]", "inputB[11]", "inputB[10]",
"inputB[9]", "inputB[8]", "inputB[7]", "inputB[6]",
"inputB[5]", "inputB[4]", "inputB[3]", "inputB[2]",
"inputB[1]", "inputB[0]"}
```

- ***group -hdl_block***
  Blocks or different levels of hierarchy can be created in ***Design Compiler*** by using the command ***group***. This command would enable designers to group together different cells and subdesigns to create a new hierarchy. This is similar to creating a new schematic for those cells that are grouped together.

```
dc_shell> group <cell_list> -design_name
<new_design_name> -cell_name <new_cell_name>
```

<cell_list> would be the list of cells or instance names that are to be grouped together. For example, to group a list of 5 cells with instance names ***I1, I2, I3, I4*** and ***I5,*** assign a new design name of ***new_module*** and new cell name of ***INEW***.

```
dc_shell> group {I1, I2, I3, I4, I5} -
design_name new_module -cell_name INEW
```

The group command can also be used to create levels of hierarchy for VHDL code using the VHDL syntax of ***BLOCK***.

Example 56 is a VHDL code of four AND gates. Two AND gates are inferred in **block1** and the remaining two AND gates are inferred in **block2**.

EXAMPLE 56 VHDL Code for Inference of Four AND Gates Using *BLOCK* statements

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY more_block_ent IS
PORT (
      inputA : IN std_logic_vector (3 downto 0);
      inputB : IN std_logic_vector (3 downto 0);
      outputC : OUT std_logic_vector (3 downto 0)
      );
END more_block_ent;

ARCHITECTURE more_block_arch OF more_block_ent IS
BEGIN
      block1: BLOCK
      BEGIN
              PROCESS (inputA, inputB)
              BEGIN
                      FOR i IN 0 to 1 LOOP
                              outputC(i) <= inputA(i) AND
                                      inputB(i);
                      END LOOP;
              END PROCESS;
      END BLOCK block1;

      block2: BLOCK
      BEGIN
              PROCESS (inputA, inputB)
              BEGIN
                      FOR i IN 2 to 3 LOOP
                              outputC(i) <= inputA(i) AND
                                      inputB(i);
              END LOOP;
      END PROCESS;
      END BLOCK block2;

END more_block_arch;
```

The code of Example 56 is synthesized using the medium effort compile option.

```
dc_shell> read -format vhdl more_block.vhd
dc_shell> current_design = more_block_ent
dc_shell> compile -map_effort medium
```

Figure 84 shows the synthesized AND gates.

The ***group*** command is used to create a block for ***block1***.

```
dc_shell> group -hdl_block block1
```

Figure 85 shows ***block1*** being created as an additional level of hierarchy.

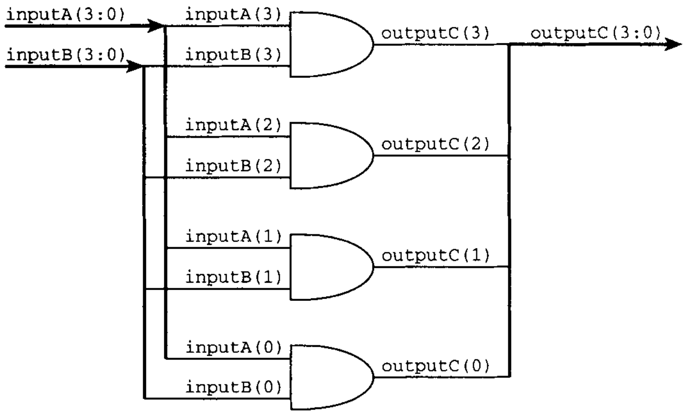


FIGURE 84 Diagram Showing Inference of Four AND Gates.

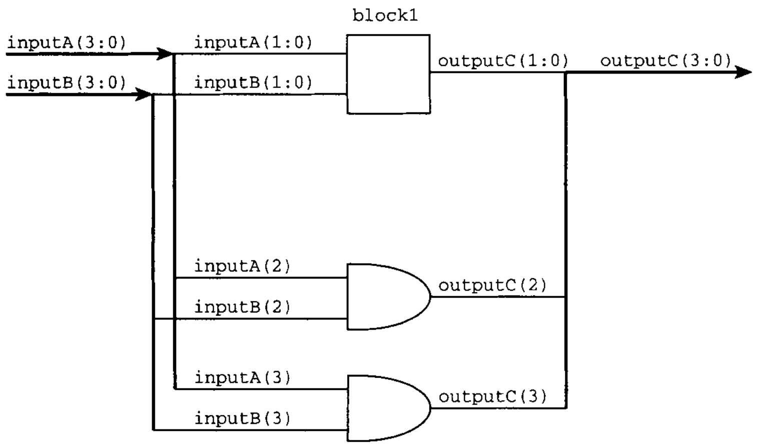


FIGURE 85 Diagram Showing Grouping of Two AND Gates into **BLOCK1**.

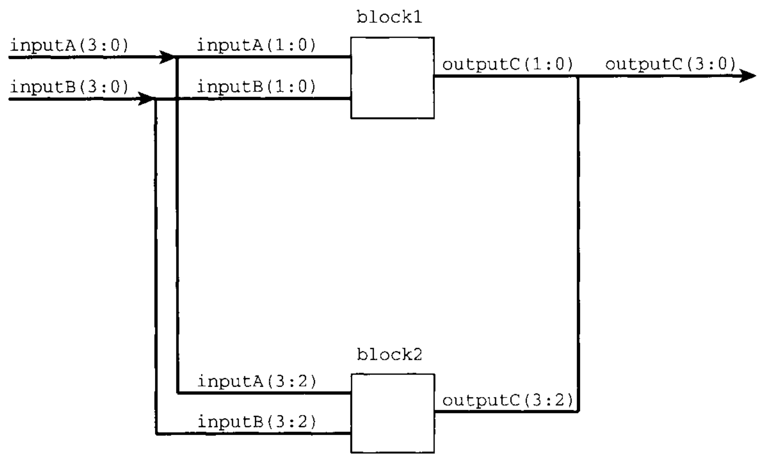


FIGURE 86 Diagram Showing Grouping of AND Gates into **BLOCK1** and **BLOCK2**

Another group command is used to create another block for ***block2***.

```
dc_shell> group -hdl_block block2
```

Figure 86 shows both ***block1*** and ***block2*** being created.

The ungroup command can be used if the designer wishes to break the hierarchy of ***block1*** and ***block2***.

```
dc_shell> ungroup -all
```

- ***GENERATE and LOOP Syntax in Synthesis***
  From Example 56 that shows VHDL code using ***BLOCK*** syntax, do you notice how the ***LOOP*** syntax is used? In synthesis, a ***LOOP*** statement is rolled out before it is synthesized.

EXAMPLE 57 Example of VHDL Code Using *LOOP* Syntax

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY more_loop_ent IS
PORT (
      inputA : IN std_logic_vector (3 downto 0);
      inputB : IN std_logic_vector (3 downto 0);
      outputC : OUT std_logic_vector (3 downto 0)
      );
END more_loop_ent;
```

```
ARCHITECTURE more_loop_arch OF more_loop_ent IS
BEGIN
      PROCESS (inputA, inputB)
      BEGIN
            FOR i IN 0 to 3 LOOP
                  outputC(i) <= inputA(i) AND inputB(i);
            END LOOP;
      END PROCESS;
END more_loop_arch;
```

Roll out of the statement occurs before synthesis.

Example 57 is rolled out into individual statements before it is synthesized.

```
outputC (0) <= inputA (0) AND inputB (0);
outputC (1) <= inputA (1) AND inputB (1);
outputC (2) <= inputA (2) AND inputB (2);
outputC (3) <= inputA (3) AND inputB (3);
```

The VHDL syntax of ***GENERATE*** also has the same function as that of ***LOOP*** syntax. Example 58 shows VHDL code using ***GENERATE*** syntax. Both Examples 57 and 58 will synthesize to the same logic.

EXAMPLE 58 Example of VHDL Code Using *GENERATE* Syntax

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY more_generate_ent IS
PORT (
      inputA : IN std_logic_vector (3 downto 0);
      inputB : IN std_logic_vector (3 downto 0);
      outputC : OUT std_logic_vector (3 downto 0)
      );
END more_generate_ent;

ARCHITECTURE more_generate_arch OF more_generate_ent IS
BEGIN

      MULT_AND_gate: FOR i IN 0 to 3 GENERATE
            outputC(i) <= inputA(i) AND inputB(i);
      END GENERATE;
END more_generate_arch;
```

- ***compare_design***
  The command ***compare_design*** can be used by the designer when he/she wishes to compare the functionality of different designs using Boolean-comparisons.

As already mentioned, ***GENERATE*** and ***LOOP*** syntax offer similar synthesis results, thus we can use the ***compare_design*** to compare the functionality of synthesis results of Example 57 (using ***LOOP*** syntax) and Example 58 (using ***GENERATE*** syntax).

```
dc_shell> read -format db {more_loop_ent.db,
more_generate_ent.db}
dc_shell> compare_design -effort medium
more_generate_ent more_loop_ent
```

The ***effort*** option is used to specify the amount of CPU time to be used for comparing the two designs.

```
Verifying Designs more_generate_ent and more_loop_ent
(Medium effort) Verification Succeeded
```

The returned results of ***Design Compiler*** show that both the synthesized results of ***LOOP*** syntax and ***GENERATE*** syntax are functionally equal.

Apart from the two examples shown using ***LOOP*** and ***GENERATE*** syntax, VHDL code in a ***LOOP*** syntax can also contain the syntax ***NEXT***. Example 59 is a VHDL example using ***LOOP*** and ***NEXT*** syntax.

EXAMPLE 59 VHDL Example Using *LOOP* and *NEXT*

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY more_next_ent IS
PORT (
      inputA : IN std_logic_vector (7 downto 0);
      inputB : IN std_logic_vector (7 downto 0);
      outputC : OUT std_logic_vector (7 downto 0)
      );
END more_next_ent;

ARCHITECTURE more_next_arch OF more_next_ent IS
BEGIN
      PROCESS (inputA, inputB)
      BEGIN
            FOR i IN 0 to 7 LOOP
                  IF (i < 4) THEN
                        NEXT;
                  ELSE
```

```
                    outputC(i) <= inputA(i) AND
                        inputB(i);
                END IF;
            END LOOP;
        END PROCESS;

END more_next_arch;
```

When Example 59 is synthesized, synthesis results only show four AND gates. Bits zero to bits three of **inputA** and **inputB** are left unconnected. When synthesis tool observes the syntax **NEXT**, it will skip the **LOOP**. Therefore, when rolled out, Example 59 will appear as follows:

```
outputC (4) <= inputA (4) AND inputB (4);
outputC (5) <= inputA (5) AND inputB (5);
outputC (6) <= inputA (6) AND inputB (6);
outputC (7) <= inputA (7) AND inputB (7);
```

Figure 87 shows the synthesized result of Example 59. The logic circuit synthesized consist only of four AND gates for inputA bits 4 to 7 and inputB bits 4 to 7.

Using this synthesized result, a design comparison is performed between this design of Example 59 with the design of **more_generate_ent** of Example 58.

```
dc_shell> read -format db
{more_generate_ent.db, more_next_ent.db}
dc_shell> compare_design -effort medium
more_generate_ent more_next_ent
```

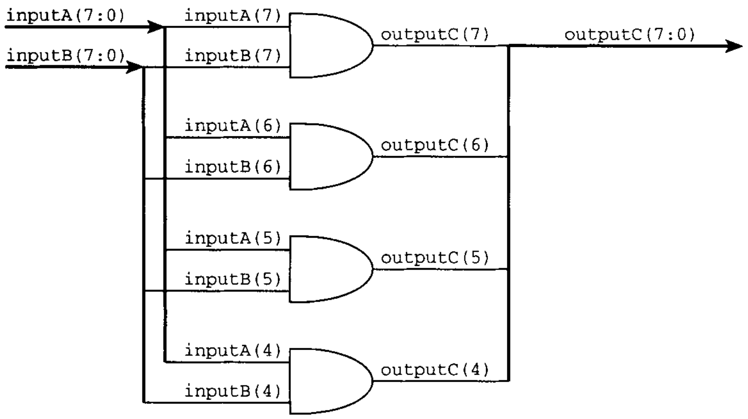


FIGURE 87 Diagram Showing Synthesis Results of VHDL Code Using ***LOOP AND NEXT*** statement.

Results from ***Design Compiler*** show that both of the designs are different and the differences are listed in what follows.

```
In the original design endpoint outputC[3] is connected to a
non-constant net.
In the optimized design endpoint outputC[3] is unconnected.
In the original design endpoint outputC[1] is connected to a
non-constant net.
In the optimized design endpoint outputC[1] is unconnected.
In the original design endpoint outputC[2] is connected to a
non-constant net.
In the optimized design endpoint outputC[2] is unconnected.
In the original design endpoint outputC[0] is connected to a
non-constant net.
In the optimized design endpoint outputC[0] is unconnected.
Endpoint outputC[7] is not present in the original design.
In the optimized design endpoint outputC[7] is connected to a
non-constant net.
Endpoint outputC[5] is not present in the original design.
In the optimized design endpoint outputC[5] is connected to a
non-constant net.
Endpoint outputC[6] is not present in the original design.
In the optimized design endpoint outputC[6] is connected to a
non-constant net.
Endpoint outputC[4] is not present in the original design.
In the optimized design endpoint outputC[4] is connected to a
non-constant net.
Error: Could not align these output ports in optimized design
'more_next_ent': (FV-15)
outputC[4]
outputC[5]
outputC[6]
outputC[7]
Error: The following involve unconnected endpoints: (FV-14)
Endpoint more_generate_ent/outputC[0] is connected while
endpoint more_next_ent/outputC[0] is unconnected.
Endpoint more_generate_ent/outputC[1] is connected while
endpoint more_next_ent/outputC[1] is unconnected.
Endpoint more_generate_ent/outputC[2] is connected while
endpoint more_next_ent/outputC[2] is unconnected.
Endpoint more_generate_ent/outputC[3] is connected while
endpoint more_next_ent/outputC[3] is unconnected.
Information: Verification Failed. (OPT-103)
Information: Verification terminated abnormally. (OPT-100)
```

- ***set_disable_timing***
  This command is used by a designer when there are timing arcs in a design that the designer wishes to disable.

  Example 60 shows an example of VHDL design that uses combinational feedback loops.

EXAMPLE 60 Design of VHDL Code Consisting of Combinational Feedback Loop

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY timing_arc_ent IS
PORT (inputA : IN std_logic;
      inputB : IN std_logic;
      inputC : IN std_logic;
      selector : IN std_logic_vector (1 downto 0);
      output : OUT std_logic);
END timing_arc_ent;

ARCHITECTURE timing_arc_arch OF timing_arc_ent IS
SIGNAL internal_output : std_logic;
BEGIN
      PROCESS (inputA, inputB, inputC, selector, internal_output)
      BEGIN
                  CASE selector IS
                        WHEN "00" =>
                              internal_output <= inputA;
                        WHEN "01" =>
                              internal_output <= inputB;
                        WHEN "10" =>
                              internal_output <= inputC;
                        WHEN "11" =>
                              internal_output <= internal_output;
                        WHEN OTHERS =>
                              NULL;
                  END CASE;
      END PROCESS;

      output <= internal_output;

END timing_arc_arch;
```

Input constraints are placed on the design and a medium-effort compilation is executed.

```
dc_shell> read -format vhdl timing_arc.vhd
dc_shell> current_design = timing_arc_ent
dc_shell> create_clock -name clock -period 10
dc_shell> set_input_delay 3 -clock clock inputA
dc_shell> set_input_delay 3 -clock clock inputB
dc_shell> set_input_delay 3 -clock clock inputC
dc_shell> set_input_delay 3 -clock clock
selector
dc_shell> set_output_delay 4 -clock clock
output*
dc_shell> compile -map_effort medium
```

Note that when compile command is executed, ***Design Compiler*** will give a warning that a timing arc between pins **DATA4_0** and **Z_0** on cell **'*cell*5'** is being disabled. The timing arc discovered by ***Design Compiler*** is restored when compilation is completed.

```
Loading design 'timing_arc_ent'

Beginning Resource Allocation  (constraint driven)
-----------------------------------------------
Allocating blocks in 'timing_arc_ent'
Information: Timing loop detected. (OPT-150)
*cell*5/DATA4_0 *cell*5/Z_0
Warning: Disabling timing arc between pins 'DATA4_0'
and 'Z_0' on cell '*cell*5'
to break a timing loop (OPT-314)
Warning: Disabling timing arc between pins 'DATA4_0'
and 'Z_0' on cell '*cell*5'
to break a timing loop (OPT-314)
Allocating blocks in 'timing_arc_ent'
Information: Timing loop detected. (OPT-150)
cell*6/U1/U1/DATA4_0 *cell*6/U1/U1/Z_0
Warning: Disabling timing arc between pins 'DATA4_0'
and 'Z_0' on cell '*cell*6/U1/U1'
to break a timing loop (OPT-314)

Beginning Mapping Optimizations  (Medium effort)
-----------------------------------------------
Structuring 'timing_arc_ent'
Mapping 'timing_arc_ent'
Information: Changed wire load model for
'timing_arc_ent' from '(none)' to '05x05'. (OPT-170)
Information: Timing loop detected. (OPT-150)
*cell*9/syn154/A *cell*9/syn154/Z *cell*9/syn132/B
*cell*9/syn132/Z *cell*10/B *cell*10/Z *cell
*9/syn161/A *cell*9/syn161/Z
```

```
Warning: Disabling timing arc between pins 'B' and
'Z' on cell '*cell*9/syn132'
to break a timing loop (OPT-314)


                               TOTAL NEG    DESIGN RULE
TRIALS  AREA   DELTA DELAY  SLACK        COST
------------------------------------------------------------
0
----------------------------------------
0


Beginning Area-Recovery Phase  (cleanup)
----------------------------------------


                               TOTAL NEG    DESIGN RULE
TRIALS  AREA   DELTA DELAY  SLACK        COST
------------------------------------------------------------
   7    14.0   0.00         0.0          0.0

  14
----------------------------------------
  21

Optimization Complete
----------------------------------------
Transferring Design 'timing_arc_ent' to database
'timing_arc_ent.db'
```

To obtain knowledge of timing arcs in a design, ***report_timing*** command can be used as well.

```
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
Information: Timing loop detected. (OPT-150)
       U16/A U16/Z U11/B U11/Z U10/B U10/Z U17/A U17/Z
Warning: Disabling timing arc between pins 'B' and
          'Z' on cell 'U11' to break a timing loop
          (OPT-314)
```

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : timing_arc_ent
Version: 1998.02-1
Date   : Sat Mar 27 14:58:22 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top

Design          Wire Loading Model          Library
-----------------------------------------------------
timing_arc_ent         05x05                  class

  Startpoint: selector[0]
              (input port clocked by clock)
  Endpoint: output (output port clocked by clock)
  Path Group: clock
  Path Type: max

  Point                                  Incr     Path
-----------------------------------------------------
  clock clock (rise edge)                0.00     0.00
  clock network delay (ideal)            0.00     0.00
  input external delay                   3.00     3.00 r
  selector[0] (in)                       0.00     3.00 r
  U11/Z (MUX21L)                         0.92     3.92 r
  U10/Z (MUX21L)                         0.44     4.36 f
  U17/Z (IVI)                            0.25     4.61 r
  output (out)                           0.00     4.61 r
  data arrival time                               4.61

  clock clock (rise edge)               10.00    10.00
  clock network delay (ideal)            0.00    10.00
  output external delay                 -4.00     6.00
  data required time                              6.00
-----------------------------------------------------
  data required time                              6.00
  data arrival time                              -4.61
-----------------------------------------------------
  slack (MET)                                     1.39
```

From the timing report, a timing arc is detected through these cells pins: ***'U16/A U16/Z U11/B U11/Z U10/B U10/Z U17/A U17/Z'***. To disable this timing arc, the command ***set_disable_timing*** is used on cell ***U11***.

```
dc_shell> set_disable_timing  {U11}
```

Apart from using ***report_timing*** to obtain a report on timing arcs, ***report_design*** command can also be used. This command will make a report on any timing arcs that are disabled in the current design (together with a large amount of other information on the current design).

```
dc_shell> report_design
```

```
****************************************
Report : design
Design : timing_arc_ent
Version: 1998.02-1
Date   : Sat Mar 27 15:04:49 1999
****************************************

Library(s) Used:
    class (File: /synopsys/libraries/syn/class.db)

Local Link Library:
    {class.db}

Flip-Flop Types:
    No flip-flop types specified.

Latch Types:
    No latch types specified.

Operating Conditions:
    No operating conditions specified.

Wire Loading Model:
    Selected automatically from the total cell area.

Name             :   05x05
Location         :   class
Resistance       :   0
Capacitance      :   1
Area             :   0
Slope            :   0.186
Fanout   Length   Points Average Cap Std Deviation
-----------------------------------------------------------
     1     0.39

Wire Loading Model Mode: top.
```

```
Timing Ranges:
    No timing ranges specified.

Pin Input Delays:
    None specified.

Pin Output Delays:
    None specified.

Disabled Timing Arcs:

    Object          Name          From Pin     To Pin
-------------------------------------------------------------
    Cell            U11

Required Licenses:
    None Required

Design Parameters:
    None specified.
```

Cell **U11** timing arc is disabled.

Do you notice from the report that the cell **U11** is reported to be a disabled timing arc?

8.7 TOP-DOWN AND BOTTOMS-UP COMPILATION

Top-Down and ***Bottoms-Up*** compilations are two methods used in synthesis compilation, and both have advantages and disadvantages. To determine which is more suitable to be used would depend on the design under consideration.

Top-Down compilation is easier to execute than ***Bottoms-Up*** compilation. In ***Top-Down*** compilation, the designer need only be concerned with top-level design constraints. The design constraints of submodules in lower-level hierarchy need not be considered. Submodule timing information is handled by ***Design Compiler*** from the top-level hierarchy.

Figure 88 shows a ***TOP***-level module containing five submodules ***A, B, C, D*** and ***E***. For ***Top-Down*** compilation, only the top-level inputs and outputs need to be configured with timing information. However, submodules need not be configured with

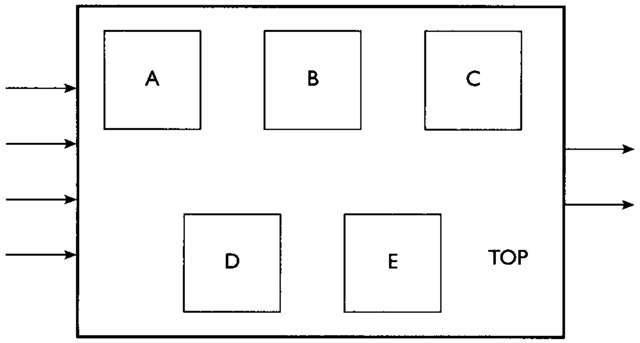


FIGURE 88 Diagram Showing a Top-Level Design Containing Five Submodules.

any timing information. During ***Top-Down*** compilation, ***Design Compiler*** will propagate timing information from toplevel down towards submodules.

Top-Down compilation is not advisable for large designs. The compilation time can be much longer by using the ***Top-Down*** compilation method, especially if the design is large. However, if the design is too large, ***Design Compiler*** might crash due to insufficient memory.

The compilation script for the design in Fig. 88 for a ***Top-Down*** compilation.

```
dc_shell> read -format vhdl {A.vhd, B.vhd,
C.vhd, D.vhd, E.vhd, TOP.vhd}
dc_shell> current_design = TOP_ent
dc_shell> {set the timing information for only
TOP level inputs and outputs}
dc_shell> compile -map_effort medium
```

When compiling large designs, it is advisable to use ***Bottoms-Up*** compilation. In this method, compilation begins at the submodule level and moves up towards the top level.

```
dc_shell> read -format vhdl {A.vhd, B.vhd,
C.vhd, D.vhd, E.vhd, TOP.vhd}
dc_shell> current_design = A_ent
dc_shell> {set timing information for sub-module A}
dc_shell> compile -map_effort medium
dc_shell> set_dont_touch A_ent
dc_shell> current_design = B_ent
dc_shell> {set timing information for sub-module B}
dc_shell> compile -map_effort medium
dc_shell> set_dont_touch B_ent
dc_shell> current_design = C_ent
dc_shell> {set timing information for sub-module C}
dc_shell> compile -map_effort medium
dc_shell> set_dont_touch C_ent
dc_shell> current_design = D_ent
dc_shell> {set timing information for sub-module D}
dc_shell> compile -map_effort medium
dc_shell> set_dont_touch D_ent
dc_shell> current_design = E_ent
dc_shell> {set timing information for sub-module E}
dc_shell> compile -map_effort medium
dc_shell> set_dont_touch E_ent
dc_shell> current_design = TOP_ent
dc_shell> compile -map_effort medium
```

From the `dc_shell` script, do you notice the difference between ***Top-Down*** and ***Bottoms-Up*** compilation method? Do you notice that upon compilation of

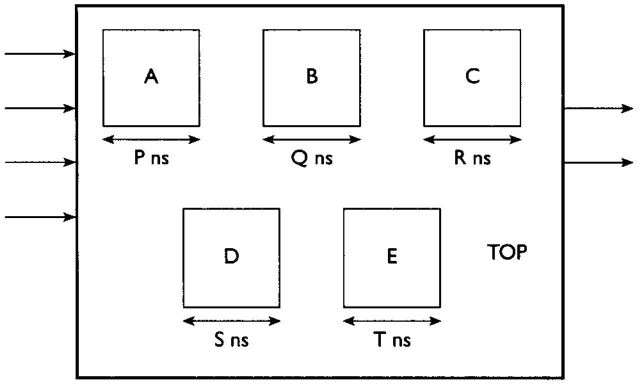


FIGURE 89 Diagram Showing Time Budgeting for TOP-Level Module

each submodule, a ***set_dont_touch*** command is executed for the **Bottoms-Up** compilation method? This would set a ***dont_touch*** attribute on those submodules, and would ensure that ***Design Compiler*** does not recompile these submodules during compilation of the ***TOP***-level module.

In the ***Bottoms-Up*** method, the designer needs to know the timing information for the inputs and outputs for each submodule. However, the designer only has timing information concerning the inputs and outputs of the ***TOP***-level module. Therefore, to obtain timing information for submodules based on the known timing information for the ***TOP***-level module, the designer needs to perform time budgeting on the submodules.

From Fig. 89, each submodule is estimated to require a certain amount of time to generate the outputs based on the inputs. For example, submodule ***A*** is estimated to require ***P***-ns delay from input to output, and submodule ***B*** is estimated to require ***Q***-ns delay from input to output.

With this estimated timing information on each of the submodules, each one of them are compiled independently.

For designs that have many submodules, it may be difficult for the designer to make manual time-budget estimations for each of the submodules. However, there are tools on the market (for example, Synopsys's ***PrimeTime***) with features that allow for auto time budgeting on submodules.

Bottoms-Up compilation is usually much faster than ***Top-Down*** compilation. However, in ***Bottoms-Up*** compilation, the designer needs to be fluent in making estimations on time budgeting for each submodule. ***Top-Down*** compilation is a lot easier because the designer need not be concerned with time-budgeting. However, the compilation time of the ***Top-Down*** method is usually much longer than ***Bottoms-Up*** compilation.

This Page Intentionally Left Blank

9

GTECH INSTANTIATION

GTECH components are Synopsys components that are not mapped to any logic function. These components are technology-independent but are functionally accurate. During synthesis, ***Design Compiler*** will try to map ***GTECH*** components to logic cells in the synthesis technology library that have the best match to the ***GTECH*** component. If ***Design Compiler*** is unable to find such a match, it will use equivalent logic cells to build the component.

EXAMPLE 61 Instantiation of a GTECH XNOR4 Component

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY GTECH;
USE GTECH.GTECH_components.ALL;


ENTITY gtech_xnor_ent IS
PORT (
      input1 : IN std_logic;
      input2 : IN std_logic;
      input3 : IN std_logic;
      input4 : IN std_logic;
      output1 : OUT std_logic
      );
END gtech_xnor_ent;
```

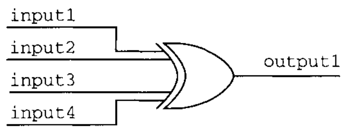


FIGURE 90 Diagram Showing GTECH XNOR4 Mapped to Four-Input XNOR Logic Cell.

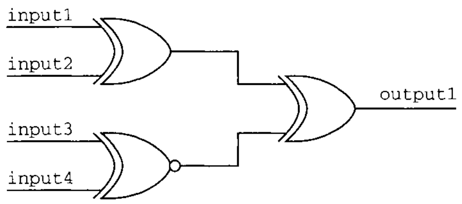


FIGURE 91 Diagram Showing GTECH XNOR4 Mapped to Two-Input XNOR Logic Cell and Two-Input XOR Logic Cell.

```
ARCHITECTURE gtech_xnor_arch OF gtech_xnor_ent IS
BEGIN
      – instantiate GTECH_XNOR4
      DUT : GTECH_XNOR4 port map (input1, input2, input3, input4,
output1);
END gtech_xnor_arch;
```

From Example 61, which shows the instantiation of a four-input ***GTECH*** XNOR component, if the synthesis technology library has a four-input XNOR logic cell, the ***GTECH*** component would map to the four-input XNOR logic cell (see Fig. 90). However, if ***Design Compiler*** can only find two-input XNOR and two-input XOR logic cells, the ***GTECH*** component will map to the circuit shown in Fig. 91.

10

DESIGNWARE LIBRARY

DesignWare is a library that consists of high-level functional modules that allow a designer the flexibility to infer them in VHDL code. Examples of VHDL code that infers DesignWare components are as follows:

```
output <= input1 + input2;
output <= input1 - input2;
output <= input1 * input2.
```

Apart from inferring ***DesignWare*** components, the designer can also instantiate these components. However, the designer must realize that some of the components in the ***DesignWare*** library are defined in Synopsys's standard synthetic library while others are not. Further, the components that are supported by Synopsys's standard synthetic library can be directly instantiated or inferred from the VHDL code. For the other remaining components that are not defined, the designer ***MUST*** set the variable ***`synthetic_library`*** to point to the ***`DesignWare sldb`*** file. This file ***MUST*** also be included to the ***`link_library`*** variable.

Example 62 is a VHDL file that instantiates ***Asynchronous FIFO Controller With Dynamic Flag Logic*** ***`(DW03_fifocntl_a_df)`*** component from ***DesignWare*** library ***`DW03.`***

EXAMPLE 62 VHDL Showing DesignWare Component Instantiation

```
LIBRARY IEEE, DW03, DWARE;
USE IEEE.std_logic_1164.ALL;
USE DWARE.DWpackages.ALL;
USE DW03.DW03_components.ALL;
```

```
ENTITY designware_ent IS
GENERIC (depth : INTEGER := 16);
PORT (level : IN std_logic_vector (bit_width(depth)-1 downto 0);
      wrqb : IN std_logic;
      rrqb : IN std_logic;
      test_mode : IN std_logic;
      reset : IN std_logic;
      w_addr : OUT std_logic_vector (bit_width(depth)-1 downto
                                     0);
      r_addr : OUT std_logic_vector (bit_width(depth)-1 downto
                                     0);
      web : OUT std_logic;
      full : OUT std_logic;
      empty : OUT std_logic;
      threshold : OUT std_logic
      );
END designware_ent;

ARCHITECTURE designware_arch OF designware_ent IS
BEGIN
       INST1: DW03_fifocntl_a_df GENERIC MAP (depth => depth)
                                PORT MAP (level => level,
                                          wrqb => wrqb,
                                          rrqb => rrqb,
                                          test_mode => test_mode,
                                          reset => reset,
                                          w_addr => w_addr,
                                          r_addr => r_addr,
                                          web => web,
                                          full => full,
                                          empty => empty,
                                          threshold => threshold);

END designware_arch;

CONFIGURATION designware_config OF designware_ent IS
FOR designware_arch
       FOR INST1 : DW03_fifocntl_a_df
              USE ENTITY DW03.DW03_fifocntl_a_df(str);
       END FOR;
END FOR;
END designware_config;
```

This component is not defined in Synopsys's standard synthetic library (***standard.sldb***). Therefore, in order to allow ***Design Compiler*** to be able to instantiate this ***DesignWare*** component, the variables ***synthetic_library*** and ***link_library*** must be set.

```
dc_shell> synthetic_library = dw03.sldb
dc_shell> link_library = link_library + dw03.sldb
```

Only when these two variables are set can the component be instantiated. If these variables are not set to point the ***synthetic_library*** to ***dw03.sldb***, upon compilation ***Design Compiler*** will give a warning message that the component ***DW03_fifocntl_a_df*** is not mapped.

In order to view the components that are defined in Synopsys's standard synthetic library, use the command ***report_synlib***.

```
dc_shell> report_synlib standard.sldb
```

Example 63 is a VHDL code that instantiates the SRAM module from DesignWare library ***DW03.*** However for this example, the SRAM module that is instantiated requires another DesignWare module that cannot be referenced by **Design Compiler.**

EXAMPLE 63 VHDL Code for Instantiation of SRAM Module from DesignWare DW03 Library

```
LIBRARY IEEE, DW03, DW01, DWARE;
USE IEEE.std_logic_1164.ALL;
USE DWARE.DWpackages.ALL;
USE DW03.DW03_components.ALL;
USE DW01.DW01_components.ALL;

ENTITY sram_ent IS
GENERIC (
        data_width : INTEGER := 16;
        depth : INTEGER := 64);
PORT (
        datain : IN std_logic_vector (data_width-1 downto 0);
        waddr : IN std_logic_vector (bit_width(depth)-1 downto 0);
        raddr : IN std_logic_vector (bit_width(depth)-1 downto 0);
        wrb : IN std_logic;
        rdb : IN std_logic;
        test_mode : IN std_logic;
        clk : IN std_logic;
        dataout : OUT std_logic_vector (data_width-1 downto 0));
END sram_ent;
```

```
ARCHITECTURE sram_arch OF sram_ent IS
BEGIN
        SRAM_U0: DW03_ram2_s_1
                 GENERIC MAP (depth => depth,
                              data_width => data_width)
                 PORT MAP (datain => datain,
                           waddr => waddr,
                           raddr => raddr,
                           wrb => wrb,
                           rdb => rdb,
                           test_mode => test_mode,
                           clk => clk,
                           dataout => dataout);
END sram_arch;

-pragma translate_off
CONFIGURATION sram_config OF sram_ent IS
FOR sram_arch
        FOR SRAM_U0: DW03_ram2_s_1 USE CONFIGURATION
DW03.DW03_ram2_s_1_cfg_sim;
        END FOR;
END FOR;
END sram_config;
-pragma translate_on
```

The SRAM module of Example 63 is 16-bits wide (designated by **data_width**) and has a depth of 64 (designated by **depth**). When this example is read into ***Design Compiler*** and compiled, an error will occur.

```
dc_shell> read -format vhdl sram.vhd
dc_shell> current_design = sram_ent
dc_shell> compile  -map_effort medium
```

```
Error: Cannot find the synthetic library implementation 'str' of module
   'DW01_decode'. (SYNDB-20)
```

The error message states that the SRAM module being instantiated cannot be found in the current pointed DesignWare library. This module is from **DW01_decode** from DesignWare library **DW01.** To check this error message, the setting of variable ***synthethic_library*** and ***link_library*** is echoed.

```
dc_shell> echo synthethic_library
dc_shell> echo link_library
```

```
{class.db standard.db dw03.sldb}
```

Both the variable ***synthethic_library*** and ***link_library*** are pointing to ***standard.sldb***, ***class.db*** and ***dw03.sldb***. At present, there are no links to DesignWare library ***DW01,*** which is exactly why Design Compiler is giving the error message that it cannot find module ***DW01_module***.

To solve this problem, the variables ***synthethic_library*** and ***link_library*** are made to point to the existing links as well as ***dw01.sldb***.

```
dc_shell> synthethic_library = synthethic_library +
dw01.sldb
dc_shell> link_library = link_library + dw01.sldb
```

With these variables set, a recompile will be successful.

10.1 CREATING YOUR OWN DESIGNWARE LIBRARY

DesignWare libraries are a good method for designing reuse of certain components. By keeping reusable components in a ***DesignWare*** library, a designer can easily reuse that same design in multiple designs.

Example 64 is similar to the VHDL shifter example of Chapter 5, except that in this example the Shifter can be of any bits wide.

EXAMPLE 64 VHDL Code of an N-Bit Shifter

Filename: MY_DW_shifter.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY MY_DW_shifter IS
GENERIC (width : INTEGER);
PORT (
     data : IN std_logic_vector( width-1 downto 0);
     load : IN std_logic;
     enable : IN std_logic;
     clock : IN std_logic;
     mode : IN std_logic_vector (1 downto 0);
     output : OUT std_logic_vector (width-1 downto 0)
     );
END MY_DW_shifter;
```

Generic declaration of width to specify the width of the shifter.

```
ARCHITECTURE arch OF MY_DW_shifter IS
SIGNAL internal_output : std_logic_vector (width-1 downto 0);
BEGIN
       PROCESS (clock)
       BEGIN
              IF (clock = '1' AND clock'EVENT) THEN
                     IF (enable = '0') THEN
                            IF (load = '0') THEN
                                   internal_output <= data;
                            ELSE
                                   IF (mode = "00") THEN
                                      — shift left
                                      internal_output <=
                                      internal_output(width-2
                                      downto 0) & '0';
                                   ELSIF (mode = "01") THEN
                                      — shift right
                                      internal_output <= '0' &
                                      internal_output (width-1
                                      downto 1);
                                   ELSIF (mode = "10") THEN
                                      — shift barrel right
                                      internal_output <=
                                      internal_output(0) &
                                      internal_output (width-1
                                      downto 1);
                                   ELSIF (mode = "11") THEN
                                      — shift barrel left
                                      internal_output <=
                                      internal_output(width-2
                                      downto 0) &
                                      internal_output(width-1);
                                   ELSE
                                      internal_output <= (others =>
                                      '0');
                                   END IF;
                            END IF;
                     ELSE
                            internal_output <= (others => '0');
                     END IF;
              END IF;
       END PROCESS;

       output <= internal_output;

END arch;
```

Before using this shifter to create a ***DesignWare*** component, the shifter is first analyzed into a library defined with the library name ***MY_DW***.

```
unix> mkdir MY_DW
unix> dc_shell
dc_shell> define_design_lib MY_DW -path ./MY_DW
dc_shell> analyze -format vhdl -lib MY_DW
MY_DW_shifter.vhd
```

To ensure that the shifter design is in the library defined with the name ***MY_DW***, the command ***report_design_lib*** is used.

```
dc_shell> report_design_lib MY_DW
```

```
**************************************
Contents of current design libraries
   MY_DW (/SYN/VHDL/MY_DW)
      Entity         : p     MY_DW_shifter
      Architecture   :  m    MY_DW_shifter(arch)

p -- This design has parameters.
m -- This architecture is the most recently
analyzed.
s -- This file is out of date with respect to its
source.
```

Now that the shifter design has been analyzed and stored into ***MY_DW*** library, a synthetic library description for the shifter is written. This synthetic library description will contain descriptions to the input and output pins of the shifter, information on the width of the pins and the desired implementation of the shifter.

Filename: MY_DW_SL.sl

```
library (MY_DW_SL.sldb) {
       module (MY_DW_shifter) {
              design_library : "MY_DW"
              parameter (width) {
                     hdl_parameter : TRUE;
                     }
       pin (data) {
              direction : input;
              bit_width : "width";
              }
```

Declaration of *MY_DW_SL* library

Declaration of design library name

Declaration of pin direction and bit size

```
        pin (load) {
                direction : input;
                bit_width : "1";
                }
        pin (enable) {
                direction : input;
                bit_width : "1";
                }
        pin (clock) {
                direction : input;
                bit_width : "1";
                }
        pin (mode) {
                direction : input;
                bit_width : "2";
                }
        pin (output) {
                direction : output;
                bit_width : "width";
                }
        implementation (arch) {}
        }

}
```

Declaration of implementation name of the shifter

The synthetic library description is compiled and linked to the variables ***synthethic_library*** and ***link_library***.

```
dc_shell> read_lib MY_DW_SL.sl
dc_shell> write_lib MY_DW_SL.sldb
dc_shell> synthethic_library = {standard.sldb,
MY_DW_SL.sldb}
dc_shell> link_library = link_library +
MY_DW_SL.sldb
```

Upon completion of these commands, we have created a ***DesignWare*** component called ***MY_DW_shifter*** in a ***DesignWare*** library ***MY_DW***. Example 65 shows a VHDL code instantiating the newly created ***DesignWare*** component ***MY_DW_shifter***.

EXAMPLE 65 VHDL Code to Instantiate *MY_DW_shifter*

Filename: MY_DW_shifter_infer.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY shifter_infer_ent IS
GENERIC (width : INTEGER := 4);
PORT (
     data : IN std_logic_vector( width-1 downto 0);
     load : IN std_logic;
     enable : IN std_logic;
     clock : IN std_logic;
     mode : IN std_logic_vector (1 downto 0);
     output : OUT std_logic_vector (width-1 downto 0)
     );
END shifter_infer_ent;

ARCHITECTURE shifter_infer_arch OF shifter_infer_ent IS

COMPONENT MY_DW_shifter
GENERIC (width : INTEGER);
PORT (
     data : IN std_logic_vector( width-1 downto 0);
     load : IN std_logic;
     enable : IN std_logic;
     clock : IN std_logic;
     mode : IN std_logic_vector (1 downto 0);
     output : OUT std_logic_vector (width-1 downto 0)
   );
END COMPONENT;

BEGIN
       DUT_shifter: MY_DW_shifter
                      GENERIC MAP (width => width)
                      PORT MAP (
                      data => data,
                      load => load,
                      enable => enable,
                      clock => clock,
                      mode => mode,
                      output => output
                                  );

END shifter_infer_arch;
```

Example 65 is compiled. Synthesis results will show the component ***MY_DW_shifter*** being instantiated in ***shifter_infer_ent*** design.

```
dc_shell> read -format vhdl MY_DW_shifter_infer.
vhd
dc_shell> current_design = shifter_infer_ent
dc_shell> compile -map_effort medium
```

```
Loading target library 'class'
  Loading design 'shifter_infer_ent'

Information: Design 'shifter_infer_ent' has no
optimization constraints set. (OPT-108)


  Beginning Resource Allocation  (area only)
  -------------------------------------
  Allocating blocks in 'shifter_infer_ent'
  Allocating blocks in 'shifter_infer_ent'
  Allocating blocks in 'MY_DW_shifter_4'
  Allocating blocks in 'MY_DW_shifter_4'
  Building model 'MY_DW_shifter_4' (arch)
Information: Changed wire load model for 'MY_DW_
shifter_4' from '(none)' to '05x05'. (OPT-170)
  Structuring 'shifter_infer_ent_MY_DW_shifter_4_0'
  Mapping 'shifter_infer_ent_MY_DW_shifter_4_0'
  Selecting implementations in 'shifter_infer_ent'

  Beginning Mapping Optimizations  (Medium effort)
  -------------------------------------
Information: Changed wire load model for
'shifter_infer_ent' from '(none)' to '05x05'. (OPT-
170)
  Structuring 'shifter_infer_ent_MY_DW_shifter_4_0'
  Mapping 'shifter_infer_ent_MY_DW_shifter_4_0'
  Selecting implementations in 'shifter_infer_ent'


                                   TOTAL NEG   DESIGN RULE
TRIALS  AREA   DELTA DELAY  SLACK        COST
------------------------------------------------------------

 0
-----
 0
```

```
Beginning Area-Recovery Phase  (cleanup)
-----------------------------------

                       TOTAL NEG      DESIGN RULE
TRIALS         AREA    DELTA DELAY    SLACK        COST
--------------------------------------------------------------

 88
-----
 88

Optimization Complete
-----------------------------------
  Transferring Design 'shifter_infer_ent_MY_DW_
shifter_4_0' to database 'shifter_infer_ent.db'
  Transferring Design 'shifter_infer_ent' to data-
base 'shifter_infer_ent.db'
Current design is 'shifter_infer_ent'.
```

A DesignWare shifter is successfully created from the shifter code of Example 64.

This Page Intentionally Left Blank

11

TESTABILITY ISSUES IN SYNTHESIS

With the growing complexity of ASIC designs, one important factor that every designer needs to consider is the testability of the design. In today's ASIC designs, a design is not considered complete if testability of the design is not taken into account. An important reason that testability of a design has come into play is the very fact that the cost of testing a design in the manufacturing process takes a rather large percentage of the total cost of manufacturing the very design itself.

A manufactured ASIC chip based on a certain design that does not take into account testability may have many defects that are not detected during the testing phase of manufacturing. Such defects would render the chip useless and therefore a mechanism must be established to enable those defects to be caught during the testing phase. Most easily observable defects are the 'stuck-at zero' and 'stuck-at-one' faults.

To address this design-testability issue, many test methodologies have been introduced. However, the most widely used test methodology is the scan technique. In this technique, all sequential elements in a design are replaced with a scan-equivalent element that uses a certain scan style on which the designer chooses to implement. There are several scan styles that can be used but the one most commonly used is the multiplexed flip-flop scan style.

Apart from the scan style, the designer also needs to consider the scan methodology involved, and make a decision on whether to use a full scan method or a partial scan method. A full scan method involves replacing all the sequential elements in a design with a scan equivalent. A partial scan method involves replacing only part of the sequential elements in design with scan equivalents. Whichever method is used by the designer would greatly depend on the design being considered. Replacing all sequential cells with a scan equivalent would obviously give a high fault-coverage value as a partial scan method can only yield a certain percentage of fault coverage. The percentage obtained would depend largely on the selected sequential cells that are replaced by scan equivalents. At this point, the reader may question why a full scan method is not always used as it does yield a higher fault coverage? The answer to that question comes down to the final area and timing considerations of a design. Replacing sequential elements with their scan equivalents always increases the area of the design as well as the timing of the logic paths covered by the scan equivalents. Therefore, whether a partial scan method or full scan method is used would depend largely on whether the designer can afford to compromise the additional area and timing requirements that are introduced with full scan methods.

In general, test insertion should be done after synthesis and before layout. Figure 92 shows a synthesis/layout flow with test insertion.

11.1 MULTIPLEXED FLIP-FLOP SCAN STYLE

In this scan style, as the name suggests, a flip-flop is replaced with its scan equivalent, which is a multiplexed flip-flop. This scan-equivalent cell has four inputs and one output, which allows for the 'test-scanning' of a design.

Description of use of each of the inputs and the output is as follows:

- ***Data*** — This pin has the data-in information for a normal flip-flop when the scan mode is not enabled.
- ***Scan data-in*** — This pin has the scan data-in information during a scan mode
- ***Scan enable*** — This pin is the scan mode pin. It is used to select the input to the multiplexer that is used as the input to the flip-flop.
- ***Clock/Scan clock*** — This pin is used as the normal clock pin during nonscan modes and the scan clock pin during the scan mode.
- ***Data/Scan data-out*** — The information on this pin could either be the data-out for a normal mode or scan mode depending on whether the scan mode is enabled.

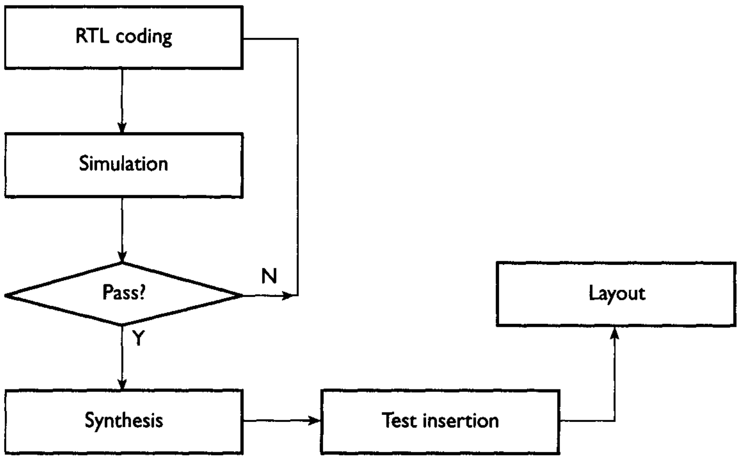


FIGURE 92 Diagram Showing Synthesis/Layout Flow Involving Test Insertion.

The use of these pins is different in scan mode than during nonscan mode. Figure 94 shows the multiplexed flip-flop being used during the scan mode and during the nonscan mode.

Figure 94 shows that there are two modes in which the design can operate. The ***Scan Mode*** (Scan enable is pulled to a logical '1') and ***Normal Mode*** (Scan enable is pulled to a logical '0').

During Scan Mode:

- Scan enable is pulled to a logical '1.' Because this signal is connected to the select signal of the multiplexer, the data going into the input of the flip-flop would always be the Scan in-data. Therefore, the data coming out of the flip-flop would always be the Scan out-data.
- The input to the clock of the flip-flop would be the Scan clock

During Normal Mode:

- Scan enable is pulled to a logical '0.' The input to the flip-flop would always be the data from the combinational logic A. Therefore, the output from the flip-flop would always be the normal data from combinational logic A.
- The input to the clock of the flip-flop would be the normal clock of the design.

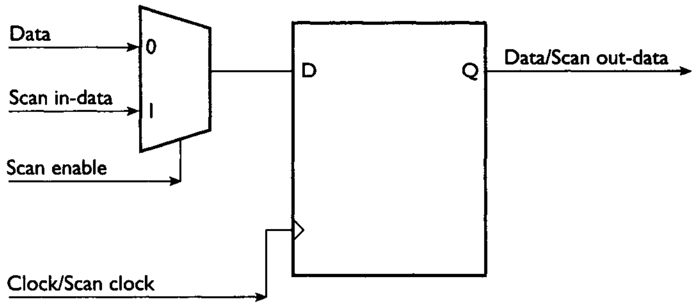


FIGURE 93 Diagram Showing a Multiplexed Flip-Flop Used as a Scan-Equivalent Cell.

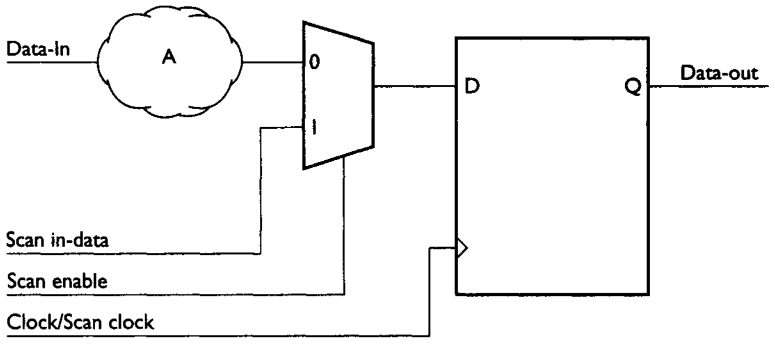


FIGURE 94 Diagram Showing Use of Multiplexed Flip-flop during Scan Mode and Normal Mode.

For a design that has more than one flip-flop, all the flip-flops in the design are replaced by a multiplexed flip-flop for the full scan method. All these multiplexed flip-flops are then connected together to form a scan chain. This would enable a scan in-data to pass from one multiplexed flip-flop to another multiplexed flip-flop during every scan clock cycle. Figure 95 shows how a scan chain is implemented on a full-scale design with more than one flip-flop.

In Fig. 95, do you notice how all the multiplexed flip-flops are connected together to form a scan chain to scan out data? Figure 95 is a design with three multiplexed flip-flops. Therefore, it would require three Scan clock cycles to scan out data from the Scan in-data port. In general, an N-stage scan chain would require N scan clock cycles to scan out data from the scan in-data port.

11.2 USING SYNOPSYS TEST COMPILER FOR SCAN INSERTION

Apart from ***Design Compiler*** and ***FSM Compiler***, Synopsys's synthesis tool also comes with ***Test Compiler***. ***Test Compiler*** allows a designer to implement testability features in the design. When a designer wishes to implement scan methodology in a design, it is important for the designer to remember not to use any of the scan-equivalent cells in the technology library during synthesis. The scan-equivalent cells will be used to replace the synthesized sequential cells during scan insertion using ***Test Compiler***.

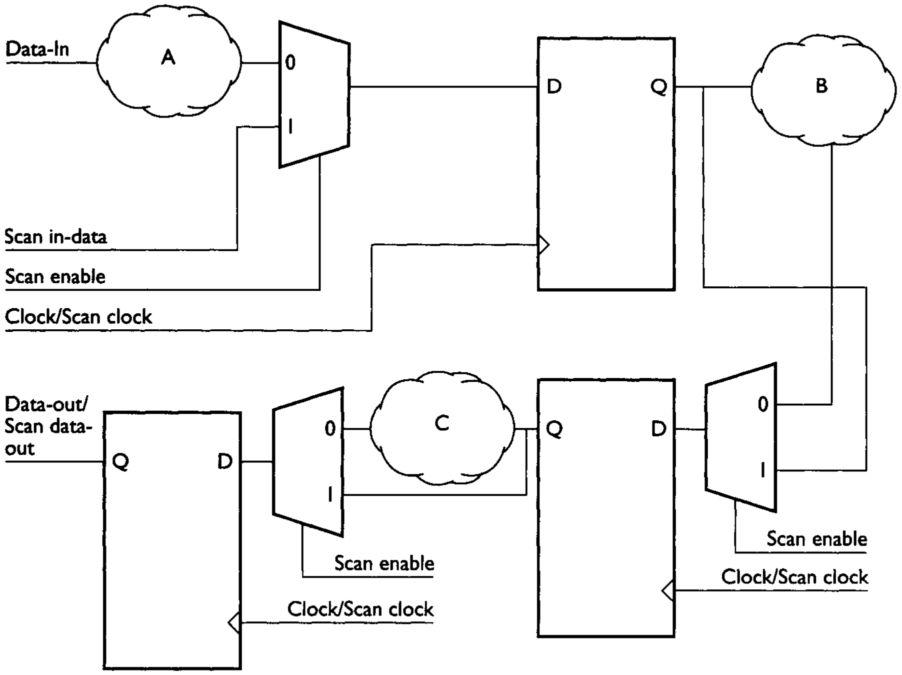


FIGURE 95 Diagram Showing a Scan Chain for a Design.

Using the same shifter example as in Chapter 5, the VHDL code is recompiled with the same set of design constraints used in Appendix B. However, in this synthesis compilation, Design Compiler is instructed not to use scan-equivalent cells during synthesis.

```
dc_shell> read -format vhdl shifter.vhd
dc_shell> current_design = shifter_ent
dc_shell> set_scan_style multiplexed_flip_flop
```

The command **set_scan_style** would ensure that ***Design Compiler*** does not use the multiplexed flip-flop during synthesis of the shifter design.

```
dc_shell> create_clock clock -name clock -period 5
dc_shell> set_input_delay 2.3 -max -clock clock
data*
dc_shell> set_input_delay 2.3 -max -clock clock
enable
dc_shell> set_input_delay 2.3 -max -clock clock
load
dc_shell> set_input_delay 2.5 -max -clock clock
mode*
dc_shell> compile -map_effort medium
```

During compilation, you will notice ***that Design Compiler*** issues a message '***Information: Choosing a test methodology will restrict the optimization of sequential cells. (UIO-12)***'. This message informs the designer that the synthesis results of the shifter design may not be the most optimum for sequential cells. This is because we have chosen a specific scan style, therefore not allowing ***Design Compiler*** to use multiplexed flip-flop in the synthesis of the shifter design.

```
Information: Choosing a test methodology will
restrict the optimization of sequential cells.
(UIO-12)

Loading target library 'class'
Loading design 'shifter_ent'

Beginning Resource Allocation  (constraint driven)
--------------------------------------
Structuring 'shifter_ent'
Mapping 'shifter_ent'
Allocating blocks in 'shifter_ent'
Allocating blocks in 'shifter_ent'
```

```
  Beginning Mapping Optimizations  (Medium effort)
  -------------------------------------
  Structuring 'shifter__ent
  Mapping 'shifter_ent'
Information: Changed wire load model for
'shifter_ent' from '(none)' to '05x05'. (OPT-170)

                                   TOTAL NEG    DESIGN RULE
TRIALS  AREA   DELTA DELAY   SLACK        COST
--------------------------------------------------------------
  53     86.0          0.08         0.1           0.0
   1     86.0          0.08         0.1           0.0
   1     86.0          0.08         0.1           0.0
   1     86.0          0.08         0.1           0.0
   8     86.0          0.00         0.0           0.0
   1     86.0          0.00         0.0           0.0
   1     86.0          0.00         0.0           0.0
   3
---------
  94

  Beginning Area-Recovery Phase  (cleanup)
  ----------------------------------------

                                   TOTAL NEG    DESIGN RULE
TRIALS  AREA   DELTA DELAY   SLACK        COST
--------------------------------------------------------------
    152
---------
    152

  Optimization Complete
  -----------------------------
  Transferring Design 'shifter_ent' to database
'shifter_ent.db'
Current design is 'shifter_ent'.
```

Once compilation of the design is complete, the ***set_test_methodology*** command can be used to set either a full scan or a partial scan method. The command ***check_test*** is also executed to check the design prior to and after scan insertion.

```
dc_shell> set_test_methodology full_scan
dc_shell> check_test
```

```
Loading design 'shifter_ent'

Information: Starting test design rule checking.
(TEST-222)
  ...full scan rules enabled...
  ...basic checks...
  ...basic sequential cell checks...
  ...checking combinational feedback loops...
  ...inferring test protocol...
Information: Inferred system/test clock port clock
(45.0,55.0). (TEST-260)
  ...simulating parallel vector...
  ...simulating parallel vector...
  ...simulating serial scan-in...
  ...simulating parallel vector...
  ...binding scan-in state...
  ...simulating parallel vector...
  ...simulating capture clock rising edge at port
clock...
  ...simulating capture clock falling edge at port
clock...
  ...creating capture clock groups...
Information: Inferred capture clock group : clock.
(TEST-262)
  ...binding scan-out state...
  ...simulating serial scan-out...
  ...simulating parallel vector...
Information: Test design rule checking completed.
(TEST-123)

***************************************************
  Test Design Rule Violation Summary

  Total violations: 0
***************************************************

***************************************************
  Sequential Cell Summary

  0 out of 4 sequential cells have violations
***************************************************

SEQUENTIAL CELLS WITHOUT VIOLATIONS
   *   4 cells are valid scan cells
```

Set how many scan chains you want and insert the scan chain.

```
dc_shell> set_scan_configuration -chain_count 1
dc_shell> insert_scan
```

```
Loading design 'shifter_ent'
  Using test design rule information from previous
check_test run
  Architecting Scan Chains
  Inserting Scan Cells
  Routing Scan Chains
  Routing Global Signals
  Mapping New Logic
  Beginning Mapping Optimizations

                             TOTAL NEG   DESIGN RULE
TRIALS  AREA   DELTA DELAY   SLACK       COST
-----------------------------------------------------
  17    94.0          0.48        1.9           0.0
   1    94.0          0.48        1.9           0.0
   9    97.0          0.46        1.7           0.0
  15    98.0          0.44        1.7           0.0
   1    98.0          0.44        1.7           0.0
   1    98.0          0.44        1.7           0.0
  11    98.0          0.42        1.6           0.0
   7   101.0          0.40        1.5           0.0
  37
------
  99

Transferring design 'shifter_ent' to database
'shifter_ent.db'
```

With the chain already inserted, group the core logic into a new level of hierarchy.

```
dc_shell> group -design_name Core -cell_name
shifter_ent filter(find(cell "*"),"-
(@is_combination
al == true) || (@is_sequential == true) ||
(@is_hierarchical == true) || (@is_black_box ==
true)")
```

```
Performing filter on cell 'U9'.
Performing filter on cell 'U10'.
Performing filter on cell 'U11'.
Performing filter on cell 'U12'.
Performing filter on cell 'U13'.
Performing filter on cell 'U14'.
Performing filter on cell 'U15'.
Performing filter on cell 'U16'.
Performing filter on cell 'U17'.
Performing filter on cell 'U18'.
Performing filter on cell 'U19'.
Performing filter on cell 'U20'.
Performing filter on cell 'U21'.
Performing filter on cell 'U22'.
Performing filter on cell 'U23'.
Performing filter on cell 'U24'.
Performing filter on cell 'U25'.
Performing filter on cell 'U26'.
Performing filter on cell 'U27'.
Performing filter on cell 'U28'.
.......
.......
Performing group on cell 'U54'.
Performing group on cell 'U55'.
Performing group on cell 'internal_output_reg[0]'.
Performing group on cell 'internal_output_reg[1]'.
Performing group on cell 'internal_output_reg[2]'.
Performing group on cell 'internal_output_reg[3]'.
```

Insert the ***JTAG*** using the ***-no_pads*** options. This option will inform ***Test Compiler*** not to insert any pad cells.

```
dc_shell> insert_jtag -no_pads
```

```
Information: 2-bit JTAG Instruction Register (IR)
being synthesized. (TEST-232)

Initiating JTAG Boundary Scan synthesis on
'shifter_ent' ...

- Synthesizing the TAP Controller ...
- Synthesizing the Instruction Register ...
- Synthesizing the Instruction decode logic ...
- Synthesizing the Bypass Register ...
- Synthesizing the Boundary Scan Register ...
- Transferring (new) design 'JTAG_BSRINCLKOBS' to
database shifter_ent.db
- Transferring (new) design 'JTAG_BSRINBOTH' to data-
base shifter_ent.db
- Transferring (new) design 'JTAG_BSROUTBOTH' to
database shifter_ent.db
- Synthesizing the TDI and TDO Logic ...
- Transferring (new) design 'JTAG_TAP' to database
shifter_ent.db
- Transferring (new) design 'JTAG_BR' to database
shifter_ent.db
Warning: Target library contains no replacement for
register 'JTAG_IR/OUT_BIT_1' (**FFGEN**). (TRANS-4)
Warning: Target library contains no replacement for
register 'JTAG_IR/OUT_BIT_0' (**FFGEN**). (TRANS-4)
- Transferring (new) design 'JTAG_IR2' to database
shifter_ent.db
- Eliminating generic logic ...
- Updating db design ...
- Transferring design 'shifter_ent' to database
shifter_ent.db ...

  JTAG Boundary Scan synthesis completed for
'shifter_ent'.
```

Test Compiler will issue a warning '***Warning: Target library contains no replacement for register*** **`'JTAG_IR/OUT_BIT_1' (**FFGEN**). (TRANS-4)'`** as the technology library on which synthesis is based (`class.db`) does not contain any register cells with the functionality required for ***Design Compiler*** to map to. Therefore, it is very important for the designer to realize that prior to inserting testability functions into a design, the technology library on which the design is mapped must have the relevant information to support testability insertions.

However, if the technology library on which the synthesis is based does contain all the relevant cells for testability insertion, the following commands are common commands used in ***Test Compiler*** to complete the test insertion.

- ***`write -format db`*** `-hier -output` ***`my_jtag.db`***
  Usage: Save the database into `my_jtag.db` filename
- ***`set_port_is_pad <list_of_ports>`***
  Usage: Pads will be inserted to all ports listed in `<list_of_ports>`
- ***`insert_pads`***
  Usage: This command when executed will insert pads into ports that have the ***`set_port_is_pad`*** attribute.
- ***`create_test_patterns`***
  Usage: This command will execute ***ATPG*** (Auto Test Pattern Generator) and create a vector file.

12 FPGA SYNTHESIS

The Field Programmable Gate Array (FPGA) is used extensively in ASIC designs, especially in prototyping new logic devices. These new designs can be programmed into FPGA very quickly and verified with minimum cost.

The following examples are based on the Xilinx FPGA 4000E synthesis library.

When synthesizing a VHDL design into FPGA, it is essential for the designer to follow certain steps (refer to Fig. 96).

From Fig. 96, a verified VHDL source code is read into ***Design Compiler***. A set of design constraints is then set on the design. Following this, the command ***insert_pads*** is executed. This command is used to map I/O pads to the pins of the design. Other commands that can be used together with this command are as follows: ***set_pad_type*** and ***set_port_is_pad***.

```
set_port_is_pad <list_of_ports> <list_of_designs>
set_pad_type <pad_type_name> <list_of_ports>
```

The command ***set_port_is_pad*** would place attributes on the list of ports used in the command. The attributes would enable ***Design Compiler*** to map I/O pads to those pads. The command ***set_pad_type*** enables the designer to choose the type of I/O pad to which the respective port is to be mapped.

Once pad mapping is completed, the designer can then proceed to compile the design. When compilation is completed (assuming that the set of design constraints set earlier is met; if not, the designer can optimize the synthesized design), the designer executes the command ***replace_fpga*** and finally writes out the database.

The ***replace_fpga*** command is very useful to the designer especially when the designer wants to convert the synthesized database (which consists of configurable logic blocks [CLB] and input/output blocks [IOB]) into schematics containing logic gates. This would allow the designer to view the schematics of the design with logic gates instead of CLB and IOB.

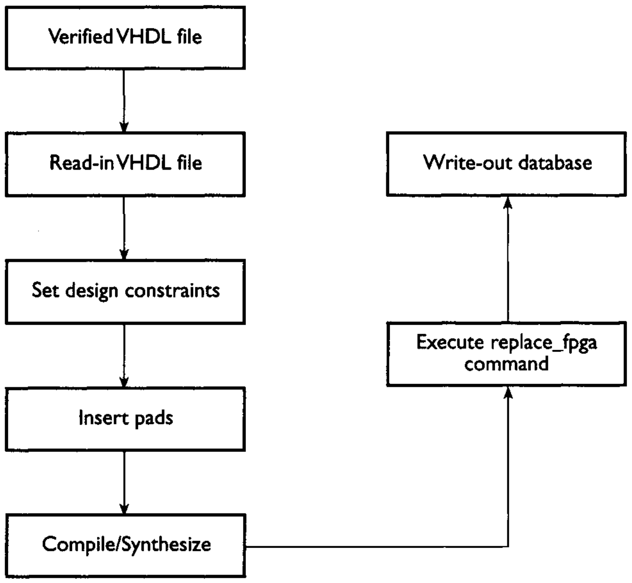


FIGURE 96 Diagram Showing Flow of Steps for Synthesizing a Design into FPGA.

> *Note:* Each Xilinx FPGA consists of CLB and IOB; CLB are programmable logic blocks that can be used to program combinational logic, sequential logic, three-state devices and even decoder logic. The CLB are used to implement the required combinational and sequential logic (including decoders and three-state devices) of a design. The IOB are blocks that can be programmed into input ports, output ports or I/O ports.

The VHDL example of the shifter design of Chapter 5 is read into ***Design Compiler*** and a set of design constraints is set on the ports of the shifter.

```
dc_shell> read -format vhdl shifter.vhd
dc_shell> create_clock clock -name clock -period
15
dc_shell> set_input_delay 2.3 -max -clock clock
data*
dc_shell> set_input_delay 2.3 -max -clock clock
enable
dc_shell> set_input_delay 2.3 -max -clock clock
load
dc_shell> set_input_delay 2.5 -max -clock clock
mode*
```

When the constraints have been set on the design, the command ***set_port_is_pad*** is executed to set the ***port_is_pad*** attribute on the design.

```
dc_shell> set_port_is_pad
```

Once the ***port_is_pad*** attribute has been set, the command ***insert_pads*** is executed to map the pins with ***port_is_pad*** attribute to I/O pads.

```
dc_shell> insert_pads
```

```
Loading design 'shifter_ent'
  Inserting IO Pads in  Design 'shifter_ent'
  Transferring Design 'shifter_ent' to database
'shifter_ent.db'
Current design is 'shifter_ent'.
```

When the I/O pads are inserted, the design is compiled.

```
dc_shell> compile  -map_effort high
```

```
Loading design 'shifter_ent'

Beginning Mapping Optimizations  (High effort)
-----------------------------------------
Structuring 'shifter_ent'
Mapping 'shifter_ent'

                               TOTAL NEG   DESIGN RULE
TRIALS  AREA   DELTA DELAY  SLACK       COST
------------------------------------------------------------

  0
--------
  0

Beginning Area-Recovery Phase  (cleanup)
-----------------------------------------

                               TOTAL NEG   DESIGN RULE
TRIALS  AREA   DELTA DELAY  SLACK       COST
------------------------------------------------------------

 24
--------
 24
Optimization Complete
-----------------------------------------
Transferring Design 'shifter_ent' to database
'shifter_ent.db'
Current design is 'shifter_ent'.
```

Upon completion of compilation, ***report_timing*** is executed to check for timing.

```
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : shifter_ent
Version: 1998.02-1
Date   : Sun Apr 11 00:05:26 1999
****************************************

Operating Conditions: WCCOM   Library: xfpga_4000e-2
Wire Loading Model Mode: top

Startpoint: mode[0] (input port clocked by clock)
Endpoint: internal_output_reg[0]
          (rising edge-triggered flip-flop clocked by
clock)
Path Group: clock
Path Type: max

 Point                                    Incr     Path
-----------------------------------------------------
 clock clock (rise edge)                  0.00     0.00
 clock network delay (ideal)              0.00     0.00
 input external delay                     2.50     2.50 r
 mode[0] (in)                             0.00     2.50 r
 U84/PAD (iob_4000)                       0.00     2.50 r
 U84/I1 (iob_4000)                        2.05     4.55 r
 U110/X (clb_4000)                        1.60     6.15 f
 U108/X (clb_4000)                        2.73     8.88 r
 internal_output_reg[0]/O (iob_4000)      0.00     8.88 r
 data arrival time                                 8.88

 clock clock (rise edge)                 15.00    15.00
 clock network delay (ideal)              0.00    15.00
 internal_output_reg[0]/OK (iob_4000)     0.00    15.00 r
 library setup time                      -3.78    11.22
 data required time                               11.22
-----------------------------------------------------
```

```
 data required time                                        11.22
 data arrival time                                         -8.88
--------------------------------------------------------------------
 slack (MET)                                                2.34
```

With timing checked to meet specification, ***report_area*** and ***report_cell*** are executed to obtain a report on the area and cell used.

```
dc_shell> report_area
dc_shell> report_cell
```

```
****************************************
Report : area
Design : shifter_ent
Version: 1998.02-1
Date   : Sun Apr 11 00:05:30 1999
****************************************

Library(s) Used:

   xfpga_4000e-2 (File: /XILINX_FPGA_LIB/
xfpga_4000e-2.db)

Number of ports:            13
Number of nets:             55
Number of cells:            19
Number of references:        2

Combinational area:         0.000000
Noncombinational area:      19.000000
Net Interconnect area:      undefined  (No wire load
                            specified)

Total cell area:            19.000000
Total area:                 undefined
1
design_analyzer> report_cell

****************************************
Report : cell
Design : shifter_ent
Version: 1998.02-1
Date   : Sun Apr 11 00:05:37 1999
****************************************
```

```
Attributes:
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                Reference   Library          Area
Attributes
-----------------------------------------------------------------
U79                 iob_4000    xfpga_4000e-2    1.00 n
U81                 iob_4000    xfpga_4000e-2    1.00 n
U84                 iob_4000    xfpga_4000e-2    1.00 n
U86                 iob_4000    xfpga_4000e-2    1.00 n
U88                 iob_4000    xfpga_4000e-2    1.00 n
U90                 iob_4000    xfpga_4000e-2    1.00 n
U92                 iob_4000    xfpga_4000e-2    1.00 n
U94                 iob_4000    xfpga_4000e-2    1.00 n
U96                 iob_4000    xfpga_4000e-2    1.00 n
U98                 iob_4000    xfpga_4000e-2    1.00 n
U100                iob_4000    xfpga_4000e-2    1.00 n
U102                clb_4000    xfpga_4000e-2    1.00 n
U104                clb_4000    xfpga_4000e-2    1.00 n
U106                clb_4000    xfpga_4000e-2    1.00 n
U108                clb_4000    xfpga_4000e-2    1.00 n
U110                clb_4000    xfpga_4000e-2    1.00 n
U112                clb_4000    xfpga_4000e-2    1.00 n
internal_
output_reg[0]       iob_4000    xfpga_4000e-2    1.00 n
internal_
output_reg[3]       iob_4000    xfpga_4000e-2    1.00 n
-----------------------------------------------------------------
Total 19 cells                                   19.00

Detailed FPGA Configuration Information:
Cell Name: U79      TYPE: IOB
          OUT:O
          PAD    I1:  I2:           TRI:

Cell Name: U81      TYPE: IOB
          OUT:O
          PAD    I1:  I2:           TRI:

Cell Name: U84      TYPE: IOB
          OUT:O
          PAD:FAST    I1:PAD I2:    TRI:
```

```
Cell Name: U86      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U88      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U90      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U92      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U94      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U96      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U98      TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U100     TYPE: IOB
        OUT:O
        PAD:FAST    I1:PAD I2:    TRI:

Cell Name: U102     TYPE: CLB
        X:F         Y:       XQ:         YQ:
        H1:         DIN:     SR:         EC:
        DX:         DY:      FFX:NOT     FFY:NOT
EQUATE F = ((F1 F2' F3 F4) + (F1 F2' F3' F4'))


Cell Name: U104     TYPE: CLB
        X:F         Y:G      XQ:         YQ:
        H1:         DIN:     SR:         EC:
        DX:         DY:      FFX:NOT     FFY:NOT
        EQUATE F = (F1 F2')
        EQUATE G = (G1' G2')
```

```
Cell Name: U106      TYPE: CLB
         X:            Y:         XQ:QX        YQ:QY
         H1:C4         DIN:       SR:          EC:
         DX:H          DY:H       FFX:K        FFY:K
         EQUATE F = ((F1 F3) + (F2 F4))
         EQUATE G = G1
         EQUATE H = (F + (H1 G))
         FFX_NAME:*cell*110   FFY_NAME:internal_out-
put_reg[2]

Cell Name: U108      TYPE: CLB
         X:H           Y:         XQ:QX        YQ:
         H1:           DIN:       SR:          EC:
         DX:H          DY:        FFX:K        FFY:NOT
         EQUATE F = (F1 F2 F3 F4)
         EQUATE G = ((G1 G3) + (G2 G4))
         EQUATE H = (F + G)
         FFX_NAME:*cell*98    FFY_NAME:

Cell Name: U110      TYPE: CLB
         X:F           Y:         XQ:QX        YQ:QY
         H1:C4         DIN:       SR:          EC:
         DX:H          DY:H       FFX:K        FFY:K
         EQUATE F = ((F1 F2' F3' F4) + (F1 F2' F3 F4'))
         EQUATE G = ((G1 G3) + (G2 G4))
         EQUATE H = (G + (H1 F))
         FFX_NAME:*cell*104   FFY_NAME:internal_out-
put_reg[1]

Cell Name: U112      TYPE: CLB
         X:H           Y:         XQ:QX        YQ:
         H1:           DIN:       SR:          EC:
         DX:H          DY:        FFX:K        FFY:NOT
         EQUATE F = (F2 F1' F3 F4)
         EQUATE G = ((G2 G3) + (G1 G4))
         EQUATE H = (F + G)
         FFX_NAME:*cell*116   FFY_NAME:

Cell Name: internal_output_reg[0]      TYPE: IOB
         OUT:OQ:RESET:OK
         PAD           I1:        I2:          TRI:
         INFF_NAME:    OUT_NAME:internal_output_reg[0]
```

```
Cell Name: internal_output_reg[3]          TYPE: IOB
        OUT:OQ:RESET:OK
        PAD           I1:         I2:          TRI:
        INFF_NAME:  OUT_NAME:internal_output_reg[3]
```

With a report on area and cells obtained, ***report_fpga*** command can also be executed to obtain information on the FPGA database.

```
dc_shell> report_fpga -one_level
```

```
****************************************
Report : fpga
          -one_level
Design : shifter_ent
Version: 1998.02-1
Date   : Sun Apr 11 00:05:50 1999
****************************************


  Xilinx FPGA Design Statistics
  ------------------------------
    FG Function Generators:       11
    H Function Generators:         4
    Number of CLB cells:           6
    Number of Hard Macros and
         Other Cells:              0
    Number of CLB in
         Other Cells:              0
    Total Number of CLB:           6

    Number of Ports:              13
    Number of Clock Pads:          0
    Number of IOB:                13

    Number of Flip Flops:          8
    Number of 3-State Buffers:     0

    Total Number of Cells:        19
```

The command ***replace_fpga*** can be used to convert the synthesized database from CLB and IOB into combinational logic form.

```
dc_shell> replace_fpga
```

```
Loading design 'shifter_ent'

  Replacing Programmable Cells by Gates
---------------------------------------------------

  Finished
---------------

  Transferring Design 'shifter_ent' to database
'shifter_ent.db'
Current design is 'shifter_ent'.
```

When the designer is satisfied with the synthesized result, the database is saved.

```
dc_shell> write -format db -hierarchy -output
```

```
Writing to file shifter_ent_fpga.db
```

13

SYNTHESIS LINKS TO LAYOUT

13.1 FORWARD-ANNOTATION

A design that is synthesized and optimized must be forward-annotated to a layout tool for placement and routing. The passing of a set of information from the synthesis tool to the layout tool is termed ***forward-annotation***.

The most important information that ***forward-annotation*** must consist of is the synthesized database (netlist), and critical paths timing information. The timing information is used as the driving mechanism by the layout tool for placing and routing of the synthesized design. For example, gates of critical paths are placed and routed close to each other. Then the layout tool will try to make the best possible placement and routing that can achieve the most optimized layout database based on the timing information it obtained from synthesis.

From Fig. 97, a common format used for netlist database is the electronic data interchange format (EDIF). For timing information, the standard delay format (SDF) is generally used. There are other formats that can be used to forward-annotate the information from synthesis to layout. However, ***EDIF*** and ***SDF*** are the ones most commonly used.

Using the synthesized database of Appendix D (synthesis results of the pipeline microcontroller example of Chapter 6), an ***EDIF*** and an ***SDF*** file are generated.

```
dc_shell> read -format db microc_ent.db
dc_shell> current_design = microc_ent
dc_shell> write -format edif -hierarchy -output
microc_ent.edif
dc_shell> write_timing -format sdf -output
microc.sdf
```

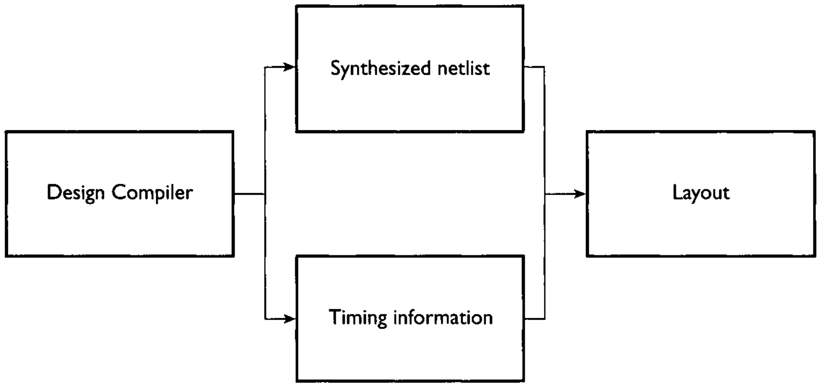


FIGURE 97 Diagram Showing Forward-Annotation of Information from Synthesis to Layout

Appendix E shows a sample of the EDIF file while Appendix F shows a sample of the SDF file that is obtained from the synthesized design of the microcontroller example of Chapter 6.

13.2 WIRELOAD MODELS

Forward-Annotation involves path timing information and the netlist database of a design. Because the timing information is used as a driving mechanism to constrain the layout tool, the question arises as to the accuracy of the timing information from synthesis.

You probably would have noticed from all the earlier synthesis examples that the use of wireload models has been at a minimum. However, wireload models are especially important if the design being synthesized is to converge with layout.

Wireload models are statistical models that are used by ***Design Compiler*** to estimate wiring loading between cells in a design. ***Design Compiler*** uses the information in a wireload model to calculate the estimated pin-to-pin delays between cells. Therefore, if the wireload model being used by ***Design Compiler*** is not an accurate representation of RC information in layout, the synthesized result might not be able to meet design requirements in layout although they do meet those requirements in synthesis.

In all of the earlier examples, only three types of wireload models have been used: '***05x05***'; '***10x10***'; and '***20x20***'. Example 66 shows these wireload models. They are statistical wireload models that are part of the synthesis technology library of `'class.db'`.

EXAMPLE 66 Examples of Wireload Models

```
wire_load("05x05") {
        resistance : 0 ;
        capacitance : 1 ;
        area : 0 ;
        slope : 0.186 ;
        fanout_length(1,0.39) ;
    }
    wire_load("10x10") {
        resistance : 0 ;
        capacitance : 1 ;
        area : 0 ;
        slope : 0.311 ;
        fanout_length(1,0.53) ;
    }
    wire_load("20x20") {
        resistance : 0 ;
        capacitance : 1 ;
        area : 0 ;
        slope : 0.547 ;
        fanout_length(1,0.86) ;
    }
```

By using an estimated wireload model, the synthesized netlist of a design together with ***SDF*** information on critical paths are used as input to constrain the layout tool for placement and routing of the design cells. When layout is completed, accurate physical RC information can be extracted from the layout tool. This information is then used to build more accurate wireload models. The updated models are back-annotated into ***Design Compiler*** to enable ***Design Compiler*** to reoptimize the design based on more accurate timing information. Paths that are now not meeting timing specifications are reoptimized.

Figure 98 shows how RC data can be extracted from layout to build more accurate wireload models to be back-annotated into synthesis for reoptimization.

The flow shown in Fig. 98 is iterated in a loop until the synthesized results converge with layout. When convergence occurs, the layout database and the synthesis database both are able to meet the necessary specifications of the design.

Referring back to Fig. 98, if initially the wireload model estimation is not accurate or too optimistic, the designer might find him or herself having problems trying to get the flow to converge. Therefore, it is important for the designer to be able to make more accurate wireload estimations during the early phases of synthesis. To do this, floorplanning tools can be used.

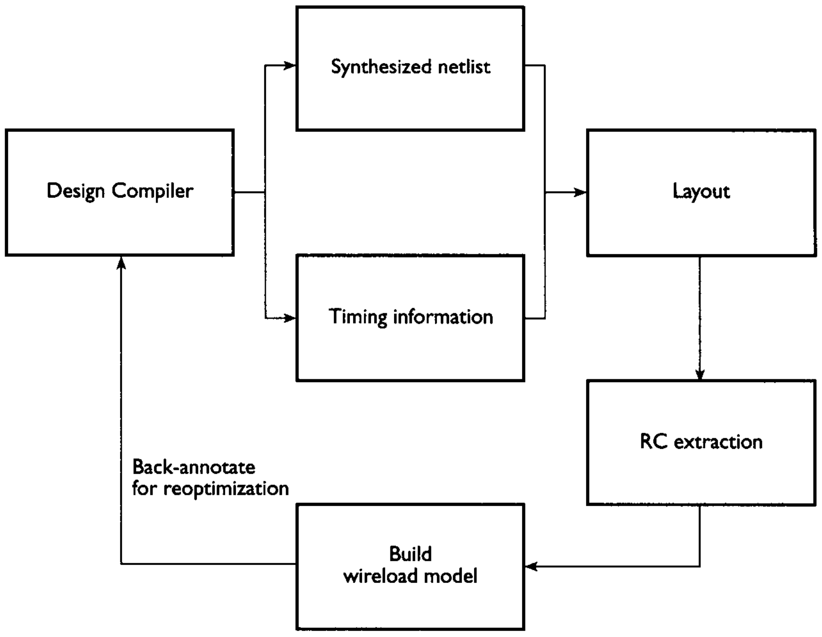


FIGURE 98 Diagram Showing Back-Annotation of RC Information

13.3 FLOORPLANNING A DESIGN

Floorplanning is the concept whereby certain cells and blocks are grouped together into different regions on a chip. As a result, information on the interconnects between different cells can be estimated more accurately. It is important for the designer to understand that floorplanning can very well affect both synthesis and layout. If the floorplan of a design is made overly pessimistic, the layout tool may later be over constrained and therefore unable to meet the required specification. If the floorplan is made overly optimistic, inaccurate information is obtained.

Floorplanning can be used before or after a design is synthesized. It can be used to partition a design into different blocks and placed on different portions of a chip before the design is synthesized. This method is commonly used when dealing with complex designs that are large. In such huge designs, interconnects between different functional blocks on the fullchip top level are difficult to estimate. Therefore, by using this method whereby functional blocks are placed on different portions of a chip, early back-annotation information can be obtained on the interconnects between these top-level functional blocks.

13.4 POST LAYOUT OPTIMIZATION

When a design that have completed layout and the data back annotated into ***Design Compiler***, some new timing violations that did not occur before might now occur. The designer is encouraged to make as little changes as possible to the layout database while fixing these violations. This would help in not requiring for a re-layout of the design if too many changes are made on the database.

A command that are often used for these post layout optimizations is the ***in_place*** optimization option during compile.

```
dc_shell> compile -in_place
```

However ***in_place*** optimization is limited on the violations that it can fix. During ***in_place*** optimization, the net structure of the design is not changed. Only violations that does not require net structure changes are fixed.

14

DESIGN GUIDELINE TO FOLLOW FOR EFFICIENT SYNTHESIS

A designer is always encouraged to follow certain design guidelines to achieve efficient synthesizable design. The following guidelines are among the few that are normally encountered and used by many designers.

- ***Naming convention***
Creating and using a good naming convention on a design is often overlooked in most designs. Having a good naming convention means a neat and systematic design, as well as ease of readability by other designers. By having a good naming convention, it is also easy for a designer to associate the function of a signal merely by looking at the name of that signal. For example, a designer may wish to use the capital letter 'B' at the end of every signal that is active low.
- ***Usage of* Std_logic *type***
It is advisable for a designer to use *std_logic* type (or *std_logic_vector* for a bus) when designing for synthesis in VHDL. By using only one type, the designer need not be concerned with conversion functions when integrating different modules together.
- ***Usage of Loopback Signal***
For designers who wish to use output ports that are looped back internally into the design, they are encouraged to define the output port as *OUTPUT* and create an additional signal that is associated with the output port and looped back internally into the design. Designers are not encouraged to declare the output port as *BUFFER* and loop it internally. Usage of the *BUFFER* declaration will cause problems when different modules are integrated together as all ports linked to the said port delcared as *BUFFER* must also be delcared as *BUFFER* type.
- ***Complete Sensitivity List***
If a designer uses a sequential PROCESS and does not have the full set of signals in the sensitivity list, simulation results for pre-synthesis and post-synthesis

might be different. When a VHDL file with incomplete sensitivity list is read into Design Compiler (`dc_shell> read -format vhdl example.vhd`), Design Compiler will issue a warning message that the file being read in does not have a complete sensitivity list.

• ***Use Separate PROCESS for Combinational Logic and Sequential Logic***

It is always encouraged for a designer to always separate combinational logic and sequential logic. Have a `PROCESS` for combinational logic and a separate `PROCESS` for sequential logic. By so doing, the designer have more flexibility if the designer wishes to create another level of hierarchy by using the group command to create hierarchy for combinational logic and sequential logic. By using separate `PROCESS`, the VHDL code also becomes much more readable.

• ***IF statements and CASE statements***

`IF` statements will synthesize to priority encoders, while `CASE` statements synthesize to multiplexers. However, when using any of these statements, it is strongly advisable for the designer to list the full set of conditions and not leave out any unspecified conditions. This would ensure that unwanted latches are not inferred during synthesis.

• ***Signal and Variable Usage***

When using signals and variables, always remember that signal assignment only occurs on the next simulation tick while variable assignment occurs immediately.

• ***Do Not Use Hierarchies in Combinational Logic***

It is a poor partitioning practice to create hierarchies in combinational logic. By so doing, optimization is not complete as logic sharing is not allowed across hierarchical boundaries.

• ***All Outputs to Be Driven by Registers***

This is the ideal situation whereby all output ports in a design are registered (-driven by flip-flops). This will ensure that no output constrain requirements are needed as the paths start from a register.

• ***Remove any Glue Logic Between Blocks***

If a design has glue logic between different blocks, it is encouraged to move the glue logic into the blocks. Design Compiler upon synthesis optimization must maintain the port definitions on a block, otherwise it is unable to optimize the glue logic with other combinational logic in the blocks. It is more efficient to not have any glue logic between blocks.

• ***Usage of FSM Compiler***

In order for a designer to use FSM Compiler to optimize a state machine design, it is a good partitioning practice to always separate random logic and state machine into different blocks. As a result, the designer can use FSM Compiler to optimize the state machine block and Design Compiler to optimize the random logic block.

APPENDIX A

(STD_LOGIC_1164 LIBRARY)

```
-- -------------------------------------------------------------------
--
--   Title      :  std_logic_1164 multi-value logic system
--   Library    :  This package shall be compiled into a library
--              :  symbolically named IEEE.
--              :
--   Developers:  IEEE model standards group (par 1164)
--   Purpose    :  This packages defines a standard for designers
--              :  to use in describing the interconnection data types
--              :  used in vhdl modeling.
--              :
--   Limitation:  The logic system defined in this package may
--              :  be insufficient for modeling switched transistors,
--              :  since such a requirement is out of the scope of this
--              :  effort. Furthermore, mathematics, primitives,
--              :  timing standards, etc. are considered orthogonal
--              :  issues as it relates to this package and are therefore
--              :  beyond the scope of this effort.
--              :
--   Note       :  No declarations or definitions shall be included in,
--              :  or excluded from this package. The "package declaration"
--              :  defines the types, subtypes and declarations of
--              :  std_logic_1164. The std_logic_1164 package body shall be
--              :  considered the formal definition of the semantics of
--              :  this package. Tool developers may choose to implement
--              :  the package body in the most efficient manner available
--              :  to them.
--              :
```

```
-- ----------------------------------------------------------------------
--   modification history :
-- ----------------------------------------------------------------------
--  version | mod. date:|
--   v4.200 | 01/02/92   |
--   v4.200 | 02/26/92   | Added Synopsys Synthesis Comments
--   v4.200 | 06/01/92   | Modified the "xnor"s to be xnor functions.
--          |            | (see Note bellow)
-- ----------------------------------------------------------------------
--
-- Note: Before the VHDL'92 language being officially adopted as
--       containing the "xnor" functions, Synopsys will support
--       the xnor functions (non-overloaded).
--
--       GongWen Huang Synopsys, Inc.
--
--

library SYNOPSYS;
use SYNOPSYS.ATTRIBUTES.ALL;


PACKAGE std_logic_1164 IS

    -------------------------------------------------------------------
    -- logic state system  (unresolved)
    -------------------------------------------------------------------
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing  Unknown
                         '0',  -- Forcing  0
                         '1',  -- Forcing  1
                         'Z',  -- High Impedance
                         'W',  -- Weak     Unknown
                         'L',  -- Weak     0
                         'H',  -- Weak     1
                         '-'   -- Don't care
                       );

    attribute ENUM_ENCODING of std_ulogic : type is "U D 0 1 Z D 0 1 D";

    -------------------------------------------------------------------
    -- unconstrained array of std_ulogic for use with the resolution function
    -------------------------------------------------------------------
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <>) OF std_ulogic;
    -------------------------------------------------------------------
    -- resolution function
    -------------------------------------------------------------------
    FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;
    --synopsys translate_off
    attribute REFLEXIVE of resolved: function is TRUE;
    attribute RESULT_INITIAL_VALUE of resolved: function is std_ulogic'POS('Z');
    --synopsys translate_on



    -------------------------------------------------------------------
    -- *** industry standard logic type ***
    -------------------------------------------------------------------
    SUBTYPE std_logic IS resolved std_ulogic;

    -------------------------------------------------------------------
    -- unconstrained array of std_logic for use in declaring signal arrays
    -------------------------------------------------------------------
    TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;

    -------------------------------------------------------------------
    -- common subtypes
    -------------------------------------------------------------------
    SUBTYPE X01     IS resolved std_ulogic RANGE 'X' TO '1'; -- ('X','0','1')
    SUBTYPE X01Z    IS resolved std_ulogic RANGE 'X' TO 'Z'; -- ('X','0','1','Z')
    SUBTYPE UX01    IS resolved std_ulogic RANGE 'U' TO '1'; -- ('U','X','0','1')
    SUBTYPE UX01Z   IS resolved std_ulogic RANGE 'U' TO 'Z'; -- ('U','X','0','1','Z')

    -------------------------------------------------------------------
    -- overloaded logical operators
    -------------------------------------------------------------------

    FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
    FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
--  function "xnor" ( l : std_ulogic; r : std_ulogic ) return ux01;
    function xnor   ( l : std_ulogic; r : std_ulogic ) return ux01;
    FUNCTION "not"  ( l : std_ulogic                 ) RETURN UX01;

    -------------------------------------------------------------------
    -- vectorized overloaded logical operators
    -------------------------------------------------------------------
    FUNCTION "and"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
    FUNCTION "nand" ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "nand" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "or"   ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "nor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "nor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

    FUNCTION "xor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "xor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

--  -----------------------------------------------------------------------
--  Note : The declaration and implementation of the "xnor" function is
--  specifically commented until at which time the VHDL language has been
--  officially adopted as containing such a function. At such a point,
--  the following comments may be removed along with this notice without
--  further "official" balloting of this std_logic_1164 package. It is
--  the intent of this effort to provide such a function once it becomes
--  available in the VHDL standard.
--  -----------------------------------------------------------------------
--  function "xnor" ( l, r : std_logic_vector  ) return std_logic_vector;
--  function "xnor" ( l, r : std_ulogic_vector ) return std_ulogic_vector;
    function xnor   ( l, r : std_logic_vector  ) return std_logic_vector;
    function xnor   ( l, r : std_ulogic_vector ) return std_ulogic_vector;

    FUNCTION "not"  ( l : std_logic_vector  ) RETURN std_logic_vector;
    FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector;

    -------------------------------------------------------------------
    -- conversion functions
    -------------------------------------------------------------------
    FUNCTION To_bit       ( s : std_ulogic
    --synopsys synthesis_off
                      ; xmap : BIT := '0'
    --synopsys synthesis_on
                    ) RETURN BIT;

    FUNCTION To_bitvector ( s : std_logic_vector
    --synopsys synthesis_off
                      ; xmap : BIT := '0'
    --synopsys synthesis_on
                    ) RETURN BIT_VECTOR;

    FUNCTION To_bitvector ( s : std_ulogic_vector
--synopsys synthesis_off
                 ; xmap : BIT := '0'
--synopsys synthesis_on
               ) RETURN BIT_VECTOR;

FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic;
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector;
FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector;
FUNCTION To_StdULogicVector ( b : BIT_VECTOR        ) RETURN std_ulogic_vector;
FUNCTION To_StdULogicVector ( s : std_logic_vector  ) RETURN std_ulogic_vector;

-------------------------------------------------------------------
-- strength strippers and type convertors
-------------------------------------------------------------------

FUNCTION To_X01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( s : std_ulogic        ) RETURN  X01;
FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( b : BIT               ) RETURN  X01;

FUNCTION To_X01Z ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( s : std_ulogic        ) RETURN  X01Z;
FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( b : BIT               ) RETURN  X01Z;

FUNCTION To_UX01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_UX01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_UX01  ( s : std_ulogic        ) RETURN  UX01;
FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_UX01  ( b : BIT               ) RETURN  UX01;

-------------------------------------------------------------------
-- edge detection
-------------------------------------------------------------------
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;

-------------------------------------------------------------------
-- object contains an unknown
-------------------------------------------------------------------
--synopsys synthesis_off
```

```
    FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  BOOLEAN;
    FUNCTION Is_X ( s : std_logic_vector  ) RETURN  BOOLEAN;
    FUNCTION Is_X ( s : std_ulogic        ) RETURN  BOOLEAN;
    --synopsys synthesis_on

END std_logic_1164;


-- ----------------------------------------------------------------------
--
--   Title     :  std_logic_1164 multi-value logic system
--   Library   :  This package shall be compiled into a library
--             :  symbolically named IEEE.
--             :
--   Developers:  IEEE model standards group (par 1164)
--   Purpose   :  This packages defines a standard for designers
--             :  to use in describing the interconnection data types
--             :  used in vhdl modeling.
--             :
--   Limitation:  The logic system defined in this package may
--             :  be insufficient for modeling switched transistors,
--             :  since such a requirement is out of the scope of this
--             :  effort. Furthermore, mathematics, primitives,
--             :  timing standards, etc. are considered orthogonal
--             :  issues as it relates to this package and are therefore
--             :  beyond the scope of this effort.
--             :
--   Note      :  No declarations or definitions shall be included in,
--             :  or excluded from this package. The "package declaration"
--             :  defines the types, subtypes and declarations of
--             :  std_logic_1164. The std_logic_1164 package body shall be
--             :  considered the formal definition of the semantics of
--             :  this package. Tool developers may choose to implement
--             :  the package body in the most efficient manner available
--             :  to them.
--             :
-- ----------------------------------------------------------------------
--   modification history :
-- ----------------------------------------------------------------------
--  version | mod. date:|
--   v4.200 | 01/02/91  |
--   v4.200 |  02/26/92 | Added Synopsys Synthesis Comments
-- ----------------------------------------------------------------------

PACKAGE BODY std_logic_1164 IS
    ---------------------------------------------------------------------
```

```
-- local types
-------------------------------------------------------------------
--synopsys synthesis_off
TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

-------------------------------------------------------------------
-- resolution function
-------------------------------------------------------------------
CONSTANT resolution_table : stdlogic_table := (
--      ---------------------------------------------------------
--      | U    X    0    1    Z    W    L    H    -          |   |
--      ---------------------------------------------------------
        ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
        ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
        ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
        ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
        ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
        ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
        ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
    );
--synopsys synthesis_on

FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
    -- pragma resolution_method three_state
    -- pragma subpgm_id 183
    --synopsys synthesis_off
    VARIABLE result : std_ulogic := 'Z';  -- weakest state default
    --synopsys synthesis_on
BEGIN
    -- the test for a single driver is essential otherwise the
    -- loop would return 'X' for a single driver of '-' and that
    -- would conflict with the value of a single driver unresolved
    -- signal.
    --synopsys synthesis_off
    IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
    ELSE
        FOR i IN s'RANGE LOOP
            result := resolution_table(result, s(i));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END resolved;
```

```
-------------------------------------------------------------------
-- tables for logical operations
-------------------------------------------------------------------

--synopsys synthesis_off
-- truth table for "and" function
CONSTANT and_table : stdlogic_table := (
--      ----------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -         |   |
--      ----------------------------------------------------
        ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
);

-- truth table for "or" function
CONSTANT or_table : stdlogic_table := (
--      ----------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -         |   |
--      ----------------------------------------------------
        ( 'U', 'U', 'U', '1', 'U', 'U', 'U', '1', 'U' ),  -- | U |
        ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | X |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 0 |
        ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ),  -- | 1 |
        ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | Z |
        ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | W |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | L |
        ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ),  -- | H |
        ( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' )   -- | - |
);

-- truth table for "xor" function
CONSTANT xor_table : stdlogic_table := (
--      ----------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -         |   |
--      ----------------------------------------------------
        ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ),  -- | U |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | X |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 0 |
        ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' ),  -- | 1 |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | Z |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | W |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | L |
        ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' ),  -- | H |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )   -- | - |
);

-- truth table for "not" function
CONSTANT not_table: stdlogic_1d :=
--  -------------------------------------------------
--  |   U    X    0    1    Z    W    L    H    -   |
--  -------------------------------------------------
     ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' );
--synopsys synthesis_on

-------------------------------------------------------------------
-- overloaded logical operators ( with optimizing hints )
-------------------------------------------------------------------

FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
-- pragma built_in SYN_AND
-- pragma subpgm_id 184
BEGIN
--synopsys synthesis_off
    RETURN (and_table(l, r));
--synopsys synthesis_on
END "and";

FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
-- pragma built_in SYN_NAND
-- pragma subpgm_id 185
BEGIN
--synopsys synthesis_off
    RETURN  (not_table ( and_table(l, r)));
--synopsys synthesis_on
END "nand";

FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
-- pragma built_in SYN_OR
-- pragma subpgm_id 186
BEGIN
--synopsys synthesis_off
    RETURN (or_table(l, r));
--synopsys synthesis_on
END "or";
```

```
    FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    -- pragma built_in SYN_NOR
    -- pragma subpgm_id 187
    BEGIN
    --synopsys synthesis_off
        RETURN  (not_table ( or_table( l, r )));
    --synopsys synthesis_on
    END "nor";

    FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    -- pragma built_in SYN_XOR
    -- pragma subpgm_id 188
    BEGIN
    --synopsys synthesis_off
        RETURN (xor_table(l, r));
    --synopsys synthesis_on
    END "xor";

--  function "xnor"  ( l : std_ulogic; r : std_ulogic ) return ux01 is
--  -- pragma built_in SYN_XNOR
--  -- pragma subpgm_id 189
--  begin
--  --synopsys synthesis_off
--      return not_table(xor_table(l, r));
--  --synopsys synthesis_on
--  end "xnor";

    function xnor  ( l : std_ulogic; r : std_ulogic ) return ux01 is
    -- pragma built_in SYN_XNOR
    -- pragma subpgm_id 189
    begin
    --synopsys synthesis_off
        return not_table(xor_table(l, r));
    --synopsys synthesis_on
    end xnor;

    FUNCTION "not"  ( l : std_ulogic ) RETURN UX01 IS
    -- pragma built_in SYN_NOT
    -- pragma subpgm_id 190
    BEGIN
    --synopsys synthesis_off
        RETURN (not_table(l));
    --synopsys synthesis_on
    END "not";
```

```
-------------------------------------------------------------------
-- and
-------------------------------------------------------------------
FUNCTION "and"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
    -- pragma built_in SYN_AND
    -- pragma subpgm_id 198
    --synopsys synthesis_off
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'and' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := and_table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "and";
---------------------------------------------------------------------
FUNCTION "and"  ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- pragma built_in SYN_AND
    -- pragma subpgm_id 191
    --synopsys synthesis_off
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'and' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := and_table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "and";
------------------------------------------------------------------
-- nand
------------------------------------------------------------------
FUNCTION "nand"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
    -- pragma built_in SYN_NAND
    -- pragma subpgm_id 199
    --synopsys synthesis_off
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'nand' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := not_table(and_table (lv(i), rv(i)));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "nand";
---------------------------------------------------------------------
FUNCTION "nand"  ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- pragma built_in SYN_NAND
    -- pragma subpgm_id 192
    --synopsys synthesis_off
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'nand' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := not_table(and_table (lv(i), rv(i)));
        END LOOP;
        END IF;
        RETURN result;
        --synopsys synthesis_on
    END "nand";
    -------------------------------------------------------------------
    -- or
    -------------------------------------------------------------------
    FUNCTION "or"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
        -- pragma built_in SYN_OR
        -- pragma subpgm_id 200
        --synopsys synthesis_off
        ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
        ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
        VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
        --synopsys synthesis_on
    BEGIN
        --synopsys synthesis_off
        IF ( l'LENGTH /= r'LENGTH ) THEN
            ASSERT FALSE
            REPORT "arguments of overloaded 'or' operator are not of the same length"
            SEVERITY FAILURE;
        ELSE
            FOR i IN result'RANGE LOOP
                result(i) := or_table (lv(i), rv(i));
            END LOOP;
        END IF;
        RETURN result;
        --synopsys synthesis_on
    END "or";
    ---------------------------------------------------------------------
    FUNCTION "or"  ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS
        -- pragma built_in SYN_OR
        -- pragma subpgm_id 193
        --synopsys synthesis_off
        ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
        ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
        VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
        --synopsys synthesis_on
    BEGIN
        --synopsys synthesis_off
        IF ( l'LENGTH /= r'LENGTH ) THEN
            ASSERT FALSE
            REPORT "arguments of overloaded 'or' operator are not of the same length"
            SEVERITY FAILURE;
        ELSE
            FOR i IN result'RANGE LOOP
            result(i) := or_table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "or";
-------------------------------------------------------------------
-- nor
-------------------------------------------------------------------
FUNCTION "nor"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
    -- pragma built_in SYN_NOR
    -- pragma subpgm_id 201
    --synopsys synthesis_off
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'nor' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := not_table(or_table (lv(i), rv(i)));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "nor";
-------------------------------------------------------------------
FUNCTION "nor"  ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- pragma built_in SYN_NOR
    -- pragma subpgm_id 194
    --synopsys synthesis_off
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'nor' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := not_table(or_table (lv(i), rv(i)));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "nor";
---------------------------------------------------------------------
-- xor
-------------------------------------------------------------------
FUNCTION "xor"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
    -- pragma built_in SYN_XOR
    -- pragma subpgm_id 202
    --synopsys synthesis_off
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'xor' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := xor_table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
    --synopsys synthesis_on
END "xor";
---------------------------------------------------------------------
FUNCTION "xor"  ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- pragma built_in SYN_XOR
    -- pragma subpgm_id 195
    --synopsys synthesis_off
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
```

```
            REPORT "arguments of overloaded 'xor' operator are not of the same length"
            SEVERITY FAILURE;
        ELSE
            FOR i IN result'RANGE LOOP
                result(i) := xor_table (lv(i), rv(i));
            END LOOP;
        END IF;
        RETURN result;
        --synopsys synthesis_on
    END "xor";
--  -------------------------------------------------------------------
--  -- xnor
--  -------------------------------------------------------------------
--  -----------------------------------------------------------------------
--  Note : The declaration and implementation of the "xnor" function is
--  specifically commented until at which time the VHDL language has been
--  officially adopted as containing such a function. At such a point,
--  the following comments may be removed along with this notice without
--  further "official" ballotting of this std_logic_1164 package. It is
--  the intent of this effort to provide such a function once it becomes
--  available in the VHDL standard.
--  -----------------------------------------------------------------------
--  function "xnor"  ( l,r : std_logic_vector ) return std_logic_vector is
--      -- pragma built_in SYN_XNOR
--      -- pragma subpgm_id 203
--      --synopsys synthesis_off
--      alias lv : std_logic_vector ( 1 to l'length ) is l;
--      alias rv : std_logic_vector ( 1 to r'length ) is r;
--      variable result : std_logic_vector ( 1 to l'length );
--      --synopsys synthesis_on
--  begin
--      --synopsys synthesis_off
--      if ( l'length /= r'length ) then
--          assert false
--          report "arguments of overloaded 'xnor' operator are not of the same length"
--          severity failure;
--      else
--          for i in result'range loop
--              result(i) := not_table(xor_table (lv(i), rv(i)));
--          end loop;
--      end if;
--      return result;
--      --synopsys synthesis_on
--  end "xnor";
--  ---------------------------------------------------------------------
--  function "xnor"  ( l,r : std_ulogic_vector ) return std_ulogic_vector is
--      -- pragma built_in SYN_XNOR
--      -- pragma subpgm_id 196
--      --synopsys synthesis_off
--      alias lv : std_ulogic_vector ( 1 to l'length ) is l;
--      alias rv : std_ulogic_vector ( 1 to r'length ) is r;
--      variable result : std_ulogic_vector ( 1 to l'length );
--      --synopsys synthesis_on
--  begin
--      --synopsys synthesis_off
--      if ( l'length /= r'length ) then
--          assert false
--          report "arguments of overloaded 'xnor' operator are not of the same length"
--          severity failure;
--      else
--          for i in result'range loop
--              result(i) := not_table(xor_table (lv(i), rv(i)));
--          end loop;
--      end if;
--      return result;
--      --synopsys synthesis_on
--  end "xnor";

    function xnor  ( l,r : std_logic_vector ) return std_logic_vector is
        -- pragma built_in SYN_XNOR
        -- pragma subpgm_id 203
        --synopsys synthesis_off
        alias lv : std_logic_vector ( 1 to l'length ) is l;
        alias rv : std_logic_vector ( 1 to r'length ) is r;
        variable result : std_logic_vector ( 1 to l'length );
        --synopsys synthesis_on
    begin
        --synopsys synthesis_off
        if ( l'length /= r'length ) then
            assert false
            report "arguments of overloaded 'xnor' operator are not of the same length"
            severity failure;
        else
            for i in result'range loop
                result(i) := not_table(xor_table (lv(i), rv(i)));
            end loop;
        end if;
        return result;
        --synopsys synthesis_on
    end xnor;
    -------------------------------------------------------------------
    function xnor  ( l,r : std_ulogic_vector ) return std_ulogic_vector is
    -- pragma built_in SYN_XNOR
    -- pragma subpgm_id 196
    --synopsys synthesis_off
    alias lv : std_ulogic_vector ( 1 to l'length ) is l;
    alias rv : std_ulogic_vector ( 1 to r'length ) is r;
    variable result : std_ulogic_vector ( 1 to l'length );
    --synopsys synthesis_on
begin
    --synopsys synthesis_off
    if ( l'length /= r'length ) then
        assert false
        report "arguments of overloaded 'xnor' operator are not of the same length"
        severity failure;
    else
        for i in result'range loop
            result(i) := not_table(xor_table (lv(i), rv(i)));
        end loop;
    end if;
    return result;
    --synopsys synthesis_on
end xnor;



-------------------------------------------------------------------
-- not
-------------------------------------------------------------------
FUNCTION "not"  ( l : std_logic_vector ) RETURN std_logic_vector IS
    -- pragma built_in SYN_NOT
    -- pragma subpgm_id 204
    --synopsys synthesis_off
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH ) := (OTHERS => 'X');
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := not_table( lv(i) );
    END LOOP;
    RETURN result;
    --synopsys synthesis_on
END;
---------------------------------------------------------------------
FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- pragma built_in SYN_NOT
    -- pragma subpgm_id 197
    --synopsys synthesis_off
    ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
    VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH ) := (OTHERS => 'X');
    --synopsys synthesis_on
BEGIN
    --synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := not_table( lv(i) );
    END LOOP;
    RETURN result;
    --synopsys synthesis_on
END;
-------------------------------------------------------------------
-- conversion tables
-------------------------------------------------------------------
--synopsys synthesis_off

TYPE logic_x01_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF X01;
TYPE logic_x01z_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF X01Z;
TYPE logic_ux01_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF UX01;
----------------------------------------------------------
-- table name : cvt_to_x01
--
-- parameters :
--          in  :  std_ulogic  -- some logic value
-- returns      :  x01         -- state value of logic value
-- purpose      :  to convert state-strength to state only
--
-- example      : if (cvt_to_x01 (input_signal) = '1' ) then ...
--
----------------------------------------------------------
CONSTANT cvt_to_x01 : logic_x01_table := (
                      'X',  -- 'U'
                      'X',  -- 'X'
                      '0',  -- '0'
                      '1',  -- '1'
                      'X',  -- 'Z'
                      'X',  -- 'W'
                      '0',  -- 'L'
                      '1',  -- 'H'
                      'X'   -- '-'
                     );

----------------------------------------------------------
-- table name : cvt_to_x01z
--
```

```
-- parameters :
--        in   :  std_ulogic  -- some logic value
-- returns     :  x01z        -- state value of logic value
-- purpose     :  to convert state-strength to state only
--
-- example     : if (cvt_to_x01z (input_signal) = '1' ) then ...
--
---------------------------------------------------------
CONSTANT cvt_to_x01z : logic_x01z_table := (
                     'X',  -- 'U'
                     'X',  -- 'X'
                     '0',  -- '0'
                     '1',  -- '1'
                     'Z',  -- 'Z'
                     'X',  -- 'W'
                     '0',  -- 'L'
                     '1',  -- 'H'
                     'X'   -- '-'
                    );

---------------------------------------------------------
-- table name : cvt_to_ux01
--
-- parameters :
--        in   :  std_ulogic  -- some logic value
-- returns     :  ux01        -- state value of logic value
-- purpose     :  to convert state-strength to state only
--
-- example     : if (cvt_to_ux01 (input_signal) = '1' ) then ...
--
---------------------------------------------------------
CONSTANT cvt_to_ux01 : logic_ux01_table := (
                     'U',  -- 'U'
                     'X',  -- 'X'
                     '0',  -- '0'
                     '1',  -- '1'
                     'X',  -- 'Z'
                     'X',  -- 'W'
                     '0',  -- 'L'
                     '1',  -- 'H'
                     'X'   -- '-'
                    );
--synopsys synthesis_on

------------------------------------------------------------------
-- conversion functions
```

```
-------------------------------------------------------------------
FUNCTION To_bit       ( s : std_ulogic
--synopsys synthesis_off
                 ; xmap : BIT := '0'
--synopsys synthesis_on
               ) RETURN BIT IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 205
BEGIN
--synopsys synthesis_off
        CASE s IS
            WHEN '0' | 'L' => RETURN ('0');
            WHEN '1' | 'H' => RETURN ('1');
            WHEN OTHERS => RETURN xmap;
        END CASE;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_bitvector ( s : std_logic_vector
--synopsys synthesis_off
                 ; xmap : BIT := '0'
--synopsys synthesis_on
               ) RETURN BIT_VECTOR IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 206
--synopsys synthesis_off
    ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
    VARIABLE result : BIT_VECTOR ( s'LENGTH-1 DOWNTO 0 );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE sv(i) IS
            WHEN '0' | 'L' => result(i) := '0';
            WHEN '1' | 'H' => result(i) := '1';
            WHEN OTHERS => result(i) := xmap;
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_bitvector ( s : std_ulogic_vector
--synopsys synthesis_off
                 ; xmap : BIT := '0'
--synopsys synthesis_on
```

```
                ) RETURN BIT_VECTOR IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 207
--synopsys synthesis_off
    ALIAS sv : std_ulogic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
    VARIABLE result : BIT_VECTOR ( s'LENGTH-1 DOWNTO 0 );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE sv(i) IS
            WHEN '0' | 'L' => result(i) := '0';
            WHEN '1' | 'H' => result(i) := '1';
            WHEN OTHERS => result(i) := xmap;
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
-------------------------------------------------------------------
FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 208
BEGIN
--synopsys synthesis_off
    CASE b IS
        WHEN '0' => RETURN '0';
        WHEN '1' => RETURN '1';
    END CASE;
--synopsys synthesis_on
END;
-------------------------------------------------------------------
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 209
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( b'LENGTH-1 DOWNTO 0 ) IS b;
    VARIABLE result : std_logic_vector ( b'LENGTH-1 DOWNTO 0 );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 210
--synopsys synthesis_off
    ALIAS sv : std_ulogic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
    VARIABLE result : std_logic_vector ( s'LENGTH-1 DOWNTO 0 );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := sv(i);
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_StdULogicVector ( b : BIT_VECTOR        ) RETURN std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 211
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( b'LENGTH-1 DOWNTO 0 ) IS b;
    VARIABLE result : std_ulogic_vector ( b'LENGTH-1 DOWNTO 0 );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_StdULogicVector ( s : std_logic_vector ) RETURN std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 212
--synopsys synthesis_off
    ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
    VARIABLE result : std_ulogic_vector ( s'LENGTH-1 DOWNTO 0 );
```

```
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := sv(i);
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;

-------------------------------------------------------------------
-- strength strippers and type convertors
-------------------------------------------------------------------
-- to_x01
-------------------------------------------------------------------
FUNCTION To_X01  ( s : std_logic_vector ) RETURN  std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 213
--synopsys synthesis_off
    ALIAS sv : std_logic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_logic_vector ( 1 TO s'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_x01 (sv(i));
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 214
--synopsys synthesis_off
    ALIAS sv : std_ulogic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_ulogic_vector ( 1 TO s'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_x01 (sv(i));
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
```

```
-------------------------------------------------------------------
FUNCTION To_X01  ( s : std_ulogic ) RETURN  X01 IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 215
BEGIN
--synopsys synthesis_off
    RETURN (cvt_to_x01(s));
--synopsys synthesis_on
END;
-------------------------------------------------------------------
FUNCTION To_X01  ( b : BIT_VECTOR ) RETURN  std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 216
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
    VARIABLE result : std_logic_vector ( 1 TO b'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
-------------------------------------------------------------------
FUNCTION To_X01  ( b : BIT_VECTOR ) RETURN  std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 217
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
    VARIABLE result : std_ulogic_vector ( 1 TO b'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
```

```
END;
-------------------------------------------------------------------
FUNCTION To_X01  ( b : BIT ) RETURN  X01 IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 218
BEGIN
--synopsys synthesis_off
        CASE b IS
            WHEN '0' => RETURN('0');
            WHEN '1' => RETURN('1');
        END CASE;
--synopsys synthesis_on
END;
-------------------------------------------------------------------
-- to_x01z
------------------------------------------------------------------
FUNCTION To_X01Z  ( s : std_logic_vector ) RETURN  std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 219
--synopsys synthesis_off
    ALIAS sv : std_logic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_logic_vector ( 1 TO s'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_x01z (sv(i));
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
-------------------------------------------------------------------
FUNCTION To_X01Z  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 220
--synopsys synthesis_off
    ALIAS sv : std_ulogic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_ulogic_vector ( 1 TO s'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_x01z (sv(i));
    END LOOP;
    RETURN result;
--synopsys synthesis_on
```

```
END;
--------------------------------------------------------------------
FUNCTION To_X01Z  ( s : std_ulogic ) RETURN  X01Z IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 221
BEGIN
--synopsys synthesis_off
    RETURN (cvt_to_x01z(s));
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_X01Z  ( b : BIT_VECTOR ) RETURN  std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 222
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
    VARIABLE result : std_logic_vector ( 1 TO b'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_X01Z  ( b : BIT_VECTOR ) RETURN  std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 223
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
    VARIABLE result : std_ulogic_vector ( 1 TO b'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
```

```
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_X01Z  ( b : BIT ) RETURN  X01Z IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 224
BEGIN
--synopsys synthesis_off
        CASE b IS
            WHEN '0' => RETURN('0');
            WHEN '1' => RETURN('1');
        END CASE;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
-- to_ux01
-------------------------------------------------------------------
FUNCTION To_UX01  ( s : std_logic_vector ) RETURN  std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 225
--synopsys synthesis_off
    ALIAS sv : std_logic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_logic_vector ( 1 TO s'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_ux01 (sv(i));
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_UX01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 226
--synopsys synthesis_off
    ALIAS sv : std_ulogic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_ulogic_vector ( 1 TO s'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_ux01 (sv(i));
    END LOOP;
    RETURN result;
```

```
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_UX01  ( s : std_ulogic ) RETURN  UX01 IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 227
BEGIN
--synopsys synthesis_off
    RETURN (cvt_to_ux01(s));
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_UX01  ( b : BIT_VECTOR ) RETURN  std_logic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 228
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
    VARIABLE result : std_logic_vector ( 1 TO b'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_UX01  ( b : BIT_VECTOR ) RETURN  std_ulogic_vector IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 229
--synopsys synthesis_off
    ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
    VARIABLE result : std_ulogic_vector ( 1 TO b'LENGTH );
--synopsys synthesis_on
BEGIN
--synopsys synthesis_off
    FOR i IN result'RANGE LOOP
        CASE bv(i) IS
            WHEN '0' => result(i) := '0';
            WHEN '1' => result(i) := '1';
        END CASE;
    END LOOP;
    RETURN result;
--synopsys synthesis_on
END;
--------------------------------------------------------------------
FUNCTION To_UX01  ( b : BIT ) RETURN  UX01 IS
-- pragma built_in SYN_FEED_THRU
-- pragma subpgm_id 230
BEGIN
--synopsys synthesis_off
        CASE b IS
            WHEN '0' => RETURN('0');
            WHEN '1' => RETURN('1');
        END CASE;
--synopsys synthesis_on
END;

-------------------------------------------------------------------
-- edge detection
-------------------------------------------------------------------
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
-- pragma subpgm_id 231
BEGIN
--synopsys synthesis_off
    RETURN (s'EVENT AND (To_X01(s) = '1') AND
                        (To_X01(s'LAST_VALUE) = '0'));
--synopsys synthesis_on
END;

FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
-- pragma subpgm_id 232
BEGIN
--synopsys synthesis_off
    RETURN (s'EVENT AND (To_X01(s) = '0') AND
                        (To_X01(s'LAST_VALUE) = '1'));
--synopsys synthesis_on
END;

-------------------------------------------------------------------
-- object contains an unknown
-------------------------------------------------------------------
--synopsys synthesis_off
FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  BOOLEAN IS
-- pragma subpgm_id 233
BEGIN
    FOR i IN s'RANGE LOOP
        CASE s(i) IS
```

```
                WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
                WHEN OTHERS => NULL;
            END CASE;
        END LOOP;
        RETURN FALSE;
    END;
    --------------------------------------------------------------------
    FUNCTION Is_X ( s : std_logic_vector  ) RETURN  BOOLEAN IS
    -- pragma subpgm_id 234
    BEGIN
        FOR i IN s'RANGE LOOP
            CASE s(i) IS
                WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
                WHEN OTHERS => NULL;
            END CASE;
        END LOOP;
        RETURN FALSE;
    END;
    --------------------------------------------------------------------
    FUNCTION Is_X ( s : std_ulogic        ) RETURN  BOOLEAN IS
    -- pragma subpgm_id 235
    BEGIN
        CASE s IS
            WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
            WHEN OTHERS => NULL;
        END CASE;
        RETURN FALSE;
    END;
    --synopsys synthesis_on

END std_logic_1164;
```

APPENDIX B

(SHIFTER SYNTHESIS RESULTS)

The shifter design example from Chapter 5.1 is synthesized with design constraints.

Filename: shifter.vhd

```
dc_shell> read -format vhdl shifter.vhd
dc_shell> current_design = shifter_ent
dc_shell> create_clock clock -name clock -period 5
dc_shell> set_input_delay 2.3 -clock clock data*
dc_shell> set_input_delay 2.3 -clock clock enable
dc_shell> set_input_delay 2.3 -clock clock load
dc_shell> set_input_delay 2.5 -clock clock mode*
```

Output delay is not set for this example as the output of the shifter is from a flip-flop. Design is compiled with ***map_effort medium*** option.

```
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max
-max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : shifter_ent
```

```
Version: 1998.02-1
Date   : Tue Mar 16 13:03:18 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design                Wire Loading Model          Library
----------------------------------------------------------
shifter_ent           05x05                       class

  Startpoint: mode[1] (input port clocked by clock)
  Endpoint: internal_output_reg[1]
           (rising edge-triggered flip-flop clocked by
           clock)
  Path Group: clock
  Path Type: max

  Point                                    Incr      Path
  ----------------------------------------------------------
  clock clock (rise edge)                   0.00      0.00
  clock network delay (ideal)               0.00      0.00
  input external delay                      2.50      2.50 f
  mode[1] (in)                              0.00      2.50 f
  U18/Z (IVI)                               0.24      2.74 r
  U15/Z (ND2I)                              0.19      2.93 f
  U14/Z (MUX21LP)                           0.58      3.51 r
  U34/Z (ND2I)                              0.19      3.70 f
  U21/Z (AO3P)                              0.82      4.51 r
  internal_output_reg[1]/TE (FD2S)          0.00      4.51 r
  data arrival time                                   4.51

  clock clock (rise edge)                   5.00      5.00
  clock network delay (ideal)               0.00      5.00
  internal_output_reg[1]/CP (FD2S)          0.00      5.00 r
  library setup time                       -1.25      3.75
  data required time                                  3.75
  ----------------------------------------------------------
  data required time                                  3.75
  data arrival time                                  -4.51
  ----------------------------------------------------------
  slack (VIOLATED)                                   -0.76
```

With the violation of 0.76 ns, a **map_effort high** of incremental mapping is executed.

```
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : shifter_ent
Version: 1998.02-1
Date   : Tue Mar 16 13:04:05 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design                Wire Loading Model         Library
---------------------------------------------------------
shifter_ent           05x05                      class

  Startpoint: mode[0] (input port clocked by clock)
  Endpoint: internal_output_reg[3]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr      Path
  -------------------------------------------------------
  clock clock (rise edge)                   0.00      0.00
  clock network delay (ideal)               0.00      0.00
  input external delay                      2.50      2.50 f
  mode[0] (in)                              0.00      2.50 f
  U47/Z (IVI)                               0.24      2.74 r
  U49/Z (ND2I)                              0.12      2.86 f
  U48/Z (IVI)                               0.29      3.15 r
  U51/Z (MUX21LP)                           0.44      3.59 f
  U50/Z (MUX21L)                            0.52      4.10 r
  internal_output_reg[3]/TE (FD2S)          0.00      4.10 r
  data arrival time                                   4.10
  clock clock (rise edge)                  5.00     5.00
  clock network delay (ideal)              0.00     5.00
  internal_output_reg[3]/CP (FD2S)         0.00     5.00 r
  library setup time                      -1.25     3.75
  data required time                                3.75
  -------------------------------------------------------
  data required time                                3.75
  data arrival time                                -4.10
  -------------------------------------------------------
  slack (VIOLATED)                                 -0.35
```

A ***group_path*** command is executed because a violation of 0.35 ns is still observed.

```
dc_shell> group_path -name critical1 -from
mode[0] -to internal_output_reg[3] -weight 5
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : shifter_ent
Version: 1998.02-1
Date   : Tue Mar 16 13:06:55 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design               Wire Loading Model            Library
------------------------------------------------------------
shifter_ent          05x05                         class

  Startpoint: mode[0] (input port clocked by clock)
  Endpoint: internal_output_reg[3]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: critical1
  Path Type: max

  Point                                     Incr      Path
  -----------------------------------------------------------
  clock clock (rise edge)                    0.00      0.00
  clock network delay (ideal)                0.00      0.00
  input external delay                       2.50      2.50 f
  mode[0] (in)                               0.00      2.50 f
  U105/Z (IVI)                               0.24      2.74 r
  U107/Z (ND2I)                              0.12      2.86 f
  U102/Z (ND2I)                              0.25      3.11 r
  U98/Z (IVI)                                0.12      3.24 f
  U96/Z (ND2I)                               0.25      3.49 r
  U95/Z (ND2I)                               0.19      3.68 f
  internal_output_reg[3]/TE (FD2S)           0.00      3.68 f
  data arrival time                                    3.68

  clock clock (rise edge)                    5.00      5.00
  clock network delay (ideal)                0.00      5.00
  internal_output_reg[3]/CP (FD2S)           0.00      5.00 r
  library setup time                        -1.25      3.75
  data required time                                   3.75
  -----------------------------------------------------------
  data required time                                   3.75
  data arrival time                                   -3.68
  -----------------------------------------------------------
  slack (MET)                                          0.07


  Startpoint: mode[0] (input port clocked by clock)
  Endpoint: internal_output_reg[1]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr      Path
  ----------------------------------------------------------
  clock clock (rise edge)                    0.00      0.00
  clock network delay (ideal)                0.00      0.00
  input external delay                       2.50      2.50 f
```

```
mode[0] (in)                               0.00       2.50 f
U57/Z (ENI)                                0.42       2.92 f
U53/Z (ND2I)                               0.25       3.18 r
U87/Z (ND2I)                               0.12       3.30 f
U85/Z (ND2I)                               0.25       3.55 r
U83/Z (ND2I)                               0.19       3.73 f
internal_output_reg[1]/TE (FD2S)          0.00       3.73 f
data arrival time                                     3.73

clock clock (rise edge)                    5.00       5.00
clock network delay (ideal)                0.00       5.00
internal_output_reg[1]/CP (FD2S)           0.00       5.00 r
library setup time                        -1.25       3.75
data required time                                    3.75
--------------------------------------------------------
data required time                                    3.75
data arrival time                         -3.73
--------------------------------------------------------
slack (MET)                                           0.02
```

APPENDIX C
(COUNTER SYNTHESIS RESULTS)

The counter design example from Chapter 5.2 is synthesized with design constraints.

Filename: counter.vhd

```
dc_shell> read -format vhdl counter.vhd
dc_shell> create_clock -name clock clock -period 5
dc_shell> set_input_delay 2.0 data* -clock clock
dc_shell> set_input_delay 2.0 enable -clock clock
dc_shell> set_input_delay 2.0 load -clock clock
dc_shell> set_input_delay 2.0 mode* -clock clock
```

Output delay is not set for this example as the output of the counter is from a flip-flop.

```
dc_shell> current_design = counter_ent
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
```

```
Design : counter_ent
Version: 1998.02-1
Date   : Tue Mar 16 14:28:01 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design                 Wire Loading Model          Library
------------------------------------------------------------
counter_ent            05x05                       class

  Startpoint: mode (input port clocked by clock)
  Endpoint: internal_output_reg[2]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr       Path
  -----------------------------------------------------------
  clock clock (rise edge)                   0.00       0.00
  clock network delay (ideal)               0.00       0.00
  input external delay                      2.00       2.00 f
  mode (in)                                 0.00       2.00 f
  U36/Z (NR2I)                              0.57       2.57 r
  U49/Z (AN2I)                              0.39       2.96 r
  U73/Z (MUX21L)                            0.44       3.40 f
  U58/Z (ND2I)                              0.25       3.65 r
  U43/Z (IVI)                               0.17       3.83 f
  U42/Z (MUX21LP)                           0.54       4.37 r
  internal_output_reg[2]/TE (FD2S)          0.00       4.37 r
  data arrival time                                    4.37

  clock clock (rise edge)                   5.00       5.00
  clock network delay (ideal)               0.00       5.00
  internal_output_reg[2]/CP (FD2S)          0.00       5.00 r
  library setup time                       -1.25       3.75
  data required time                                   3.75
  -----------------------------------------------------------
  data required time                                   3.75
  data arrival time                                   -4.37
  -----------------------------------------------------------
  slack (VIOLATED)                                    -0.62
```

With a setup violation of 0.62 ns, a ***map_effort high*** compilation with incremental mapping is executed.

```
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : counter_ent
Version: 1998.02-1
Date   : Tue Mar 16 14:28:52 1999
****************************************


Operating Conditions:
Wire Loading Model Mode: top

Design                 Wire Loading Model          Library
-----------------------------------------------------------
counter_ent            05x05                       class

  Startpoint: mode (input port clocked by clock)
  Endpoint: internal_output_reg[3]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                        Incr      Path
  -----------------------------------------------------------
  clock clock (rise edge)                       0.00      0.00
  clock network delay (ideal)                   0.00      0.00
  input external delay                          2.00      2.00 f
  mode (in)                                     0.00      2.00 f
  U87/Z (IVI)                                   0.24      2.24 r
  U97/Z (ND2I)                                  0.12      2.36 f
  U95/Z (ND2I)                                  0.25      2.61 r
  U60/Z (ND2I)                                  0.19      2.80 f
  U94/Z (MUX21LP)                               0.54      3.34 r
  U84/Z (MUX21LP)                               0.44      3.78 f
  internal_output_reg[3]/TE (FD2S)              0.00      3.78 f
  data arrival time                                       3.78

  clock clock (rise edge)                       5.00      5.00
  clock network delay (ideal)                   0.00      5.00
  internal_output_reg[3]/CP (FD2S)              0.00      5.00 r
  library setup time                           -1.25      3.75
  data required time                                      3.75
 ----------------------------------------------------------
  data required time                                      3.75
  data arrival time                                      -3.78
  ---------------------------------------------------------
  slack (VIOLATED)                                       -0.03
```

APPENDIX D

(PIPELINE MICROCONTROLLER SYNTHESIS RESULTS — TOP-DOWN COMPILATION)

The pipeline microcontroller example from Chapter 6 is synthesized with timing constraints. The timing information is only on the top level of the microcontroller. Compilation performed is ***Top-Down***.
Read in the vhdl files.

```
dc_shell> read -format vhdl {decode.vhd
predecode.vhd rf.vhd ex.vhd microc.vhd}
```

Set the design constraints. Output delays are not set as outputs are flip-flop driven.

```
dc_shell> current_design = microc_ent
dc_shell> create_clock clock -name clock -period 25
dc_shell> set_input_delay 3.0 -clock clock -max data*
dc_shell> set_input_delay 3.0 -clock clock -max source1*
dc_shell> set_input_delay 3.0 -clock clock -max destina-
tion*
dc_shell> set_input_delay 3.0 -clock clock -max source2*
dc_shell> set_input_delay 3.0 -clock clock -max inst
```

Compile with ***map_effort medium***.

```
dc_shell> compile -map_effort medium
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : microc_ent
Version: 1998.02-1
Date   : Wed Mar 17 14:08:58 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design                 Wire Loading Model          Library
------------------------------------------------------------
microc_ent             20x20                       class

  Startpoint: DUT_decode/d_command_reg[0]
              (rising edge-triggered flip-flop clocked by
clock)
  Endpoint: DUT_execute/int_ex_data_reg[31]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                      Incr     Path
  ------------------------------------------------------------
  clock clock (rise edge)                    0.00     0.00
  clock network delay (ideal)                0.00     0.00
  DUT_decode/d_command_reg[0]/CP (FD1S)      0.00     0.00 r
  DUT_decode/d_command_reg[0]/Q (FD1S)       1.36     1.36 r
  DUT_decode/d_command[0] (decode_ent)       0.00     1.36 r
  DUT_execute/d_command[0] (execute_ent)     0.00     1.36 r
  DUT_execute/U308/Z (IVI)                   0.36     1.72 f
  DUT_execute/U311/Z (ND2I)                  0.37     2.09 r
  DUT_execute/U193/Z (MUX21LP)               0.47     2.56 f
  DUT_execute/U437/Z (NR2I)                  0.65     3.21 r
  DUT_execute/U521/Z (ND2I)                  0.48     3.69 f
  DUT_execute/U171/Z (MUX21LP)               1.16     4.85 r
  DUT_execute/mul_87/mult/mult/A[0]
(execute_ent_DW02_mult_16_16_0)              0.00     4.85 r
  DUT_execute/mul_87/mult/mult/U584/Z2
(B3IP)                                       0.62     5.46 r
```

```
  DUT_execute/mul_87/mult/mult/U1555/Z
(ND2I)                                     0.25     5.72 f
  DUT_execute/mul_87/mult/mult/U1556/Z
(ND2I)                                     0.27     5.99 r
  DUT_execute/mul_87/mult/mult/U334/Z
(ND2I)                                     0.15     6.14 f
  DUT_execute/mul_87/mult/mult/U335/Z
(IVI)                                      0.33     6.47 r
  DUT_execute/mul_87/mult/mult/U748/Z
(ND2I)                                     0.15     6.62 f
  DUT_execute/mul_87/mult/mult/U165/Z
(ENI)                                      0.51     7.14 f
  DUT_execute/mul_87/mult/mult/U58/Z
(ENI)                                      0.49     7.63 f
  DUT_execute/mul_87/mult/mult/U2465/Z
(MUX21L)                                   0.88     8.51 r
  DUT_execute/mul_87/mult/mult/U837/Z
(ENI)                                      0.51     9.02 f
  DUT_execute/mul_87/mult/mult/U2370/Z
(IVI)                                      0.33     9.35 r
  DUT_execute/mul_87/mult/mult/U2129/Z
(ND2I)                                     0.32     9.67 f
  DUT_execute/mul_87/mult/mult/U167/Z
(AO3P)                                     0.75    10.42 r
  DUT_execute/mul_87/mult/mult/U1672/Z
(ND2I)                                     0.25    10.67 f
  DUT_execute/mul_87/mult/mult/U416/Z
(ND2I)                                     0.39    11.06 r
  DUT_execute/mul_87/mult/mult/U1674/Z
(ND2I)                                     0.32    11.37 f
  DUT_execute/mul_87/mult/mult/U11/Z
(ND3P)                                     0.83    12.21 r
  DUT_execute/mul_87/mult/mult/U842/Z
(ENI)                                      0.51    12.72 f
  DUT_execute/mul_87/mult/mult/U460/Z
(ENI)                                      0.51    13.23 f
  DUT_execute/mul_87/mult/mult/U492/Z
(ENI)                                      0.49    13.72 f
  DUT_execute/mul_87/mult/mult/U110/Z
(MUX21LP)                                  1.01    14.73 r
  DUT_execute/mul_87/mult/mult/U329/Z
(ENI)                                      0.44    15.17 f
  DUT_execute/mul_87/mult/mult/U2526/Z
(ENI)                                      0.44    15.62 f
  DUT_execute/mul_87/mult/mult/U850/Z
(ENI)                                      0.51    16.13 f
```

```
  DUT_execute/mul_87/mult/mult/U2419/Z
(IVI)                                      0.33    16.46 r
  DUT_execute/mul_87/mult/mult/U1927/Z
(ND2I)                                     0.15    16.62 f
  DUT_execute/mul_87/mult/mult/U1928/Z
(ND2I)                                     0.33    16.94 r
  DUT_execute/mul_87/mult/mult/U531/Z
(ENI)                                      0.44    17.39 f
  DUT_execute/mul_87/mult/mult/U2540/Z
(ENI)                                      0.51    17.90 f
  DUT_execute/mul_87/mult/mult/U514/Z
(ENI)                                      0.44    18.35 f
  DUT_execute/mul_87/mult/mult/U203/Z
(ENI)                                      0.44    18.79 f
  DUT_execute/mul_87/mult/mult/U853/Z
(ENI)                                      0.51    19.31 f
  DUT_execute/mul_87/mult/mult/U2435/Z
(IVI)                                      0.33    19.64 r
  DUT_execute/mul_87/mult/mult/U2025/Z
(ND2I)                                     0.25    19.89 f
  DUT_execute/mul_87/mult/mult/U1055/Z
(ND2I)                                     0.31    20.20 r
  DUT_execute/mul_87/mult/mult/U1107/Z
(AO7P)                                     0.36    20.56 f
  DUT_execute/mul_87/mult/mult/U1084/Z
(AN2I)                                     0.62    21.18 f
  DUT_execute/mul_87/mult/mult/U2027/Z
(ND2I)                                     0.27    21.45 r
  DUT_execute/mul_87/mult/mult/U384/Z
(ND2I)                                     0.15    21.60 f
  DUT_execute/mul_87/mult/mult/U385/Z
(IVI)                                      0.33    21.93 r
  DUT_execute/mul_87/mult/mult/FS/B[18]
(execute_ent_DW01_add_30_0)                0.00    21.93 r
  DUT_execute/mul_87/mult/mult/FS/U47/Z
(IVI)                                      0.15    22.08 f
  DUT_execute/mul_87/mult/mult/FS/U136/Z
(ND2I)                                     0.33    22.40 r
  DUT_execute/mul_87/mult/mult/FS/U86/Z
(ND2I)                                     0.25    22.66 f
  DUT_execute/mul_87/mult/mult/FS/U10/Z
(IVI)                                      0.26    22.92 r
  DUT_execute/mul_87/mult/mult/FS/U41/Z
(AN2I)                                     0.36    23.28 r
```

```
  DUT_execute/mul_87/mult/mult/FS/U179/Z
(ND2I)                                     0.15    23.43 f
  DUT_execute/mul_87/mult/mult/FS/U42/Z
(ND2I)                                     0.27    23.70 r
  DUT_execute/mul_87/mult/mult/FS/U43/Z
(IVI)                                      0.23    23.93 f
  DUT_execute/mul_87/mult/mult/FS/U36/Z
(ND2I)                                     0.27    24.20 r
  DUT_execute/mul_87/mult/mult/FS/U189/Z
(ND2I)                                     0.15    24.35 f
  DUT_execute/mul_87/mult/mult/FS/U190/Z
(ND2I)                                     0.39    24.74 r
  DUT_execute/mul_87/mult/mult/FS/U152/Z
(ND2I)                                     0.22     4.96 f
  DUT_execute/mul_87/mult/mult/FS/U16/Z
(ND3P)                                     0.87    25.83 r
  DUT_execute/mul_87/mult/mult/FS/U15/Z
(IVI)                                      0.20    26.03 f
  DUT_execute/mul_87/mult/mult/FS/U191/Z
(AO7P)                                     0.75    26.77 r
  DUT_execute/mul_87/mult/mult/FS/U64/Z
(ENI)                                      0.44    27.22 f
  DUT_execute/mul_87/mult/mult/FS/SUM[29]
(execute_ent_DW01_add_30_0)                0.00    27.22 f
  DUT_execute/mul_87/mult/mult/PRODUCT[31]
(execute_ent_DW02_mult_16_16_0)            0.00    27.22 f
  DUT_execute/U510/Z (IVI)                 0.26    27.48 r
  DUT_execute/U546/Z (ND2I)                0.15    27.63 f
  DUT_execute/U213/Z (ND2I)                0.27    27.90 r
  DUT_execute/U214/Z (IVI)                 0.15    28.05 f
  DUT_execute/int_ex_data_reg[31]/D
(FD1)                                      0.00    28.05 f
  data arrival time                                28.05

  clock clock (rise edge)                 25.00    25.00
  clock network delay (ideal)              0.00    25.00
  DUT_execute/int_ex_data_reg[31]/CP
(FD1)                                      0.00    25.00 r
  library setup time                      -0.80    24.20
  data required time                               24.20
  ----------------------------------------------------------
  data required time                               24.20
  data arrival time                               -28.05
  ----------------------------------------------------------
  slack (VIOLATED)                                 -3.85
```

A setup violation of 3.85 ns is observed. A ***characterize*** command is executed to characterize ***execute_ent***.

```
dc_shell> characterize DUT_execute
dc_shell> current_design = execute_ent
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> current_design = microc_ent
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : microc_ent
Version: 1998.02-1
Date   : Wed Mar 17 16:31:51 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design                 Wire Loading Model         Library
---------------------------------------------------------
microc_ent             20x20                      class

  Startpoint: DUT_execute/int_ex_destination_reg[2]
             (rising edge-triggered flip-flop clocked by
clock)
  Endpoint: DUT_execute/int_ex_data_reg[31]
           (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr    Path
  --------------------------------------------------------
  clock clock (rise edge)                   0.00    0.00
  clock network delay (ideal)               0.00    0.00
  DUT_execute/int_ex_destination_reg[2]/CP
 (FD1)                                      0.00    0.00 r
```

```
  DUT_execute/int_ex_destination_reg[2]/Q
(FD1)                                          1.59     1.59 r
  DUT_execute/U623/Z (ENI)                     0.38     1.96 r
  DUT_execute/U309/Z (ND2I)                    0.15     2.11 f
  DUT_execute/U204/Z (IVI)                     0.33     2.44 r
  DUT_execute/U202/Z (ND2I)                    0.15     2.60 f
  DUT_execute/U207/Z (AN2I)                    0.62     3.21 f
  DUT_execute/U521/Z (ND2I)                    0.46     3.68 r
  DUT_execute/U171/Z (MUX21LP)                 1.08     4.75 r
  DUT_execute/mul_87/mult/mult/A[0]
(execute_ent_DW02_mult_16_16_0)                0.00     4.75 r
  DUT_execute/mul_87/mult/mult/U3230/Z
(IVI)                                          0.23     4.98 f
  DUT_execute/mul_87/mult/mult/U3493/Z
(IVI)                                          0.33     5.31 r
  DUT_execute/mul_87/mult/mult/U3490/Z
(ND2I)                                         0.25     5.56 f
  DUT_execute/mul_87/mult/mult/U3140/Z
(ND2I)                                         0.33     5.89 r
  DUT_execute/mul_87/mult/mult/U2977/Z
(ND2I)                                         0.15     6.04 f
  DUT_execute/mul_87/mult/mult/U3489/Z
(IVI)                                          0.33     6.37 r
  DUT_execute/mul_87/mult/mult/U3485/Z
(ND2I)                                         0.15     6.53 f
  DUT_execute/mul_87/mult/mult/U2988/Z
(ND2I)                                         0.33     6.85 r
  DUT_execute/mul_87/mult/mult/U2979/Z
(ND2I)                                         0.15     7.01 f
  DUT_execute/mul_87/mult/mult/U2982/Z
(ENI)                                          0.44     7.45 f
  DUT_execute/mul_87/mult/mult/U3084/Z
(ENI)                                          0.51     7.96 f
  DUT_execute/mul_87/mult/mult/U3083/Z
(ENI)                                          0.44     8.41 f
  DUT_execute/mul_87/mult/mult/U2501/Z
(ENI)                                          0.49     8.90 f
  DUT_execute/mul_87/mult/mult/U838/Z
(ENI)                                          0.45     9.34 r
  DUT_execute/mul_87/mult/mult/U2371/Z
(IVI)                                          0.23     9.57 f
  DUT_execute/mul_87/mult/mult/U1671/Z
(ND2I)                                         0.33     9.90 r
  DUT_execute/mul_87/mult/mult/U3054/Z
(ND2I)                                         0.25    10.15 f
  DUT_execute/mul_87/mult/mult/U3056/Z
(ND2I)                                         0.27    10.42 r
```

```
  DUT_execute/mul_87/mult/mult/U3361/Z
(IVI)                                      0.23    10.65 f
  DUT_execute/mul_87/mult/mult/U3064/Z
(ND2I)                                     0.27    10.92 r
  DUT_execute/mul_87/mult/mult/U3556/Z
(ND2I)                                     0.25    11.17 f
  DUT_execute/mul_87/mult/mult/U2729/Z
(IVI)                                      0.26    11.44 r
  DUT_execute/mul_87/mult/mult/U1677/Z
(ND2I)                                     0.25    11.69 f
  DUT_execute/mul_87/mult/mult/U1678/Z
(ND2I)                                     0.27    11.96 r
  DUT_execute/mul_87/mult/mult/U1680/Z
(ND2I)                                     0.25    12.21 f
  DUT_execute/mul_87/mult/mult/U762/Z
(ND2I)                                     0.27    12.48 r
  DUT_execute/mul_87/mult/mult/U3288/Z
(IVI)                                      0.15    12.63 f
  DUT_execute/mul_87/mult/mult/U3284/Z
(ND2I)                                     0.33    12.96 r
  DUT_execute/mul_87/mult/mult/U3291/Z
(ND2I)                                     0.22    13.18 f
  DUT_execute/mul_87/mult/mult/U3299/Z
(MUX21L)                                   0.62    13.80 r
  DUT_execute/mul_87/mult/mult/U2511/Z
(ENI)                                      0.49    14.29 f
  DUT_execute/mul_87/mult/mult/U3300/Z
(ENI)                                      0.45    14.74 r
  DUT_execute/mul_87/mult/mult/U2390/Z
(IVI)                                      0.41    15.15 f
  DUT_execute/mul_87/mult/mult/U3499/Z
(AO7P)                                     0.75    15.89 r
  DUT_execute/mul_87/mult/mult/U3529/Z
(ND2I)                                     0.38    16.28 f
  DUT_execute/mul_87/mult/mult/U55/Z
(ENI)                                      0.51    16.79 f
  DUT_execute/mul_87/mult/mult/U2139/Z
(ND2I)                                     0.33    17.12 r
  DUT_execute/mul_87/mult/mult/U520/Z
(ND2I)                                     0.15    17.27 f
  DUT_execute/mul_87/mult/mult/U519/Z
(NR2I)                                     0.65    17.92 r
  DUT_execute/mul_87/mult/mult/U1864/Z
(ND2I)                                     0.15    18.07 f
  DUT_execute/mul_87/mult/mult/U1865/Z
(ND2I)                                     0.33    18.40 r
```

```
  DUT_execute/mul_87/mult/mult/U2701/Z
(ND2I)                                   0.15    18.55 f
  DUT_execute/mul_87/mult/mult/U2702/Z
(IVI)                                    0.26    18.81 r
  DUT_execute/mul_87/mult/mult/U2700/Z
(ND2I)                                   0.15    18.96 f
  DUT_execute/mul_87/mult/mult/U2698/Z
(ND2I)                                   0.27    19.23 r
  DUT_execute/mul_87/mult/mult/U2697/Z
(ND2I)                                   0.25    19.49 f
  DUT_execute/mul_87/mult/mult/U3308/Z
(ND2I)                                   0.27    19.76 r
  DUT_execute/mul_87/mult/mult/U3376/Z
(IVI)                                    0.31    20.07 f
  DUT_execute/mul_87/mult/mult/FS/B[22]
(execute_ent_DW01_add_30_0)              0.00    20.07 f
  DUT_execute/mul_87/mult/mult/FS/U79/Z
(IVI)                                    0.33    20.40 r
  DUT_execute/mul_87/mult/mult/FS/U351/Z
(ND2I)                                   0.15    20.55 f
  DUT_execute/mul_87/mult/mult/FS/U97/Z
(ND2I)                                   0.27    20.82 r
  DUT_execute/mul_87/mult/mult/FS/U143/Z
(ND2I)                                   0.15    20.97 f
  DUT_execute/mul_87/mult/mult/FS/U144/Z
(ND2I)                                   0.33    21.30 r
  DUT_execute/mul_87/mult/mult/FS/U96/Z
(ND2I)                                   0.15    21.45 f
  DUT_execute/mul_87/mult/mult/FS/U303/Z
(ND2I)                                   0.27    21.72 r
  DUT_execute/mul_87/mult/mult/FS/U234/Z
(AN2I)                                   0.43    22.15 r
  DUT_execute/mul_87/mult/mult/FS/U302/Z
(ND2I)                                   0.15    22.30 f
  DUT_execute/mul_87/mult/mult/FS/U305/Z
(ND2I)                                   0.27    22.57 r
  DUT_execute/mul_87/mult/mult/FS/U307/Z
(ND2I)                                   0.25    22.83 f
  DUT_execute/mul_87/mult/mult/FS/U265/Z
(NR2I)                                   0.65    23.47 r
  DUT_execute/mul_87/mult/mult/FS/U264/Z
(ND2I)                                   0.15    23.62 f
  DUT_execute/mul_87/mult/mult/FS/U263/Z
(ND2I)                                   0.27    23.89 r
  DUT_execute/mul_87/mult/mult/FS/U267/Z
(ENI)                                    0.44    24.34 f
```

```
  DUT_execute/mul_87/mult/mult/FS/SUM[29]
(execute_ent_DW01_add_30_0)                   0.00   24.34 f
  DUT_execute/mul_87/mult/mult/PRODUCT[31]
(execute_ent_DW02_mult_16_16_0)               0.00   24.34 f
  DUT_execute/U510/Z (IVI)                    0.26   24.60 r
  DUT_execute/U546/Z (ND2I)                   0.15   24.75 f
  DUT_execute/U213/Z (ND2I)                   0.27   25.02 r
  DUT_execute/U214/Z (IVI)                    0.15   25.17 f
  DUT_execute/int_ex_data_reg[31]/D (FD1)     0.00   25.17 f
  data arrival time                                  25.17

  clock clock (rise edge)                    25.00   25.00
  clock network delay (ideal)                 0.00   25.00
  DUT_execute/int_ex_data_reg[31]/CP
(FD1)                                         0.00   25.00 r
  library setup time                         -0.80   24.20
  data required time                                 24.20
  ---------------------------------------------------------
  data required time                                 24.20
  data arrival time                                 -25.17
  ---------------------------------------------------------
  slack (VIOLATED)                                   -0.97
```

Setup violation is reduced to –0.97 ns. Design ***execute_ent*** is flattened.

```
dc_shell> current_design = execute_ent
dc_shell> ungroup -all -flatten
dc_shell> compile -map_effort high -
incremental_mapping
dc_shell> current_design = microc_ent
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : microc_ent
Version: 1998.02-1
Date   : Wed Mar 17 17:06:39 1999
****************************************
```

```
Operating Conditions:
Wire Loading Model Mode: top

Design                 Wire Loading Model          Library
-----------------------------------------------------------
microc_ent             20x20                       class

  Startpoint: DUT_execute/int_ex_destination_reg[3]
              (rising edge-triggered flip-flop clocked by
clock)
  Endpoint: DUT_execute/int_ex_data_reg[24]
            (rising edge-triggered flip-flop clocked by
clock)
  Path Group: clock
  Path Type: max

  Point                                     Incr    Path
  ---------------------------------------------------------
  clock clock (rise edge)                   0.00    0.00
  clock network delay (ideal)               0.00    0.00
  DUT_execute/int_ex_destination_reg[3]/CP
(FD1)                                       0.00    0.00 r
  DUT_execute/int_ex_destination_reg[3]/Q
(FD1)                                       1.59    1.59 r
  DUT_execute/U624/Z (ENI)                  0.38    1.96 r
  DUT_execute/U310/Z (ND2I)                 0.15    2.11 f
  DUT_execute/U203/Z (IVI)                  0.33    2.44 r
  DUT_execute/U202/Z (ND2I)                 0.15    2.60 f
  DUT_execute/U207/Z (AN2I)                 0.62    3.21 f
  DUT_execute/U521/Z (ND2I)                 0.46    3.68 r
  DUT_execute/U171/Z (MUX21LP)              1.08    4.75 r
  DUT_execute/mul_87/mult/mult/U3230/Z
(IVI)                                       0.23    4.98 f
  DUT_execute/mul_87/mult/mult/U3493/Z
(IVI)                                       0.33    5.31 r
  DUT_execute/mul_87/mult/mult/U3490/Z
(ND2I)                                      0.25    5.56 f
  DUT_execute/mul_87/mult/mult/U3140/Z
(ND2I)                                      0.33    5.89 r
  DUT_execute/mul_87/mult/mult/U2977/Z
(ND2I)                                      0.15    6.04 f
  DUT_execute/mul_87/mult/mult/U3489/Z
(IVI)                                       0.33    6.37 r
  DUT_execute/mul_87/mult/mult/U3485/Z
(ND2I)                                      0.15    6.53 f
```

```
  DUT_execute/mul_87/mult/mult/U2988/Z
(ND2I)                                     0.33     6.85 r
  DUT_execute/mul_87/mult/mult/U2979/Z
(ND2I)                                     0.15     7.01 f
  DUT_execute/mul_87/mult/mult/U2982/Z
(ENI)                                      0.44     7.45 f
  DUT_execute/mul_87/mult/mult/U3084/Z
(ENI)                                      0.51     7.96 f
  DUT_execute/mul_87/mult/mult/U3081/Z
(ENI)                                      0.52     8.48 r
  DUT_execute/mul_87/mult/mult/U2124/Z
(ND2I)                                     0.32     8.80 f
  DUT_execute/mul_87/mult/mult/U3094/Z
(AO3P)                                     0.75     9.54 r
  DUT_execute/mul_87/mult/mult/U3097/Z
(IVI)                                      0.23     9.77 f
  DUT_execute/mul_87/mult/mult/U3103/Z
(ENI)                                      0.44    10.22 f
  DUT_execute/mul_87/mult/mult/U2566/Z
(ENI)                                      0.44    10.66 f
  DUT_execute/mul_87/mult/mult/U457/Z
(ENI)                                      0.51    11.18 f
  DUT_execute/mul_87/mult/mult/U2832/Z
(ENI)                                      0.44    11.62 f
  DUT_execute/U816/Z (ENI)                 0.44    12.06 f
  DUT_execute/mul_87/mult/mult/U2772/Z
(ENI)                                      0.44    12.51 f
  DUT_execute/mul_87/mult/mult/U3000/Z
(ENI)                                      0.51    13.02 f
  DUT_execute/mul_87/mult/mult/U2407/Z
(IVI)                                      0.33    13.35 r
  DUT_execute/mul_87/mult/mult/U1895/Z
(ND2I)                                     0.15    13.51 f
  DUT_execute/mul_87/mult/mult/U1896/Z
(ND2I)                                     0.33    13.83 r
  DUT_execute/mul_87/mult/mult/U2418/Z
(ENI)                                      0.44    14.28 f
  DUT_execute/mul_87/mult/mult/U2533/Z
(ENI)                                      0.51    14.79 f
  DUT_execute/mul_87/mult/mult/U2714/Z
(ENI)                                      0.44    15.24 f
  DUT_execute/mul_87/mult/mult/U2712/Z
(ENI)                                      0.44    15.68 f
  DUT_execute/mul_87/mult/mult/U852/Z
(ENI)                                      0.51    16.20 f
```

```
  DUT_execute/mul_87/mult/mult/U3125/Z
(ENI)                                        0.51   16.71 f
  DUT_execute/U838/Z (ENI)                   0.51   17.22 f
  DUT_execute/U834/Z (ENI)                   0.44   17.67 f
  DUT_execute/U835/Z (ENI)                   0.51   18.18 f
  DUT_execute/mul_87/mult/mult/U3004/Z
(IVI)                                        0.26   18.45 r
  DUT_execute/mul_87/mult/mult/U3005/Z
(ND2I)                                       0.15   18.60 f
  DUT_execute/mul_87/mult/mult/U3006/Z
(ND2I)                                       0.27   18.87 r
  DUT_execute/mul_87/mult/mult/U3007/Z
(IVI)                                        0.15   19.01 f
  DUT_execute/mul_87/mult/mult/U3017/Z
(ND2I)                                       0.27   19.28 r
  DUT_execute/mul_87/mult/mult/U3018/Z
(IVI)                                        0.20   19.48 f
  DUT_execute/mul_87/mult/mult/U3032/Z
(MUX21LP)                                    0.57   20.06 r
  DUT_execute/mul_87/mult/mult/U3033/Z
(ND2I)                                       0.15   20.21 f
  DUT_execute/mul_87/mult/mult/U2814/Z
(ENI)                                        0.49   20.70 f
  DUT_execute/mul_87/mult/mult/U2813/Z
(ENI)                                        0.45   21.14 r
  DUT_execute/mul_87/mult/mult/FS/U289/Z
(ND2I)                                       0.15   21.30 f
  DUT_execute/mul_87/mult/mult/FS/U288/Z
(NR2I)                                       0.65   21.94 r
  DUT_execute/mul_87/mult/mult/FS/U287/Z
(ND2I)                                       0.46   22.40 f
  DUT_execute/U889/Z (ND2I)                  0.33   22.72 r
  DUT_execute/U858/Z (ND2I)                  0.35   23.08 f
  DUT_execute/U839/Z (ND2I)                  0.27   23.35 r
  DUT_execute/mul_87/mult/mult/FS/U319/Z
(ND2I)                                       0.22   23.57 f
  DUT_execute/mul_87/mult/mult/FS/U103/Z
(MUX21L)                                     0.88   24.45 r
  DUT_execute/U572/Z (ND2I)                  0.15   24.60 f
  DUT_execute/U249/Z (ND2I)                  0.27   24.87 r
DUT_execute/int_ex_data_reg[24]/D (FD1)      0.00   24.87 r
  data arrival time                                 24.87

  clock clock (rise edge)                   25.00   25.00
  clock network delay (ideal)                0.00   25.00
  DUT_execute/int_ex_data_reg[24]/CP
(FD1)                                          0.00   25.00 r
  library setup time                          -0.80   24.20
  data required time                                  24.20
  --------------------------------------------------------
  data required time                                  24.20
  data arrival time                                  -24.87
  --------------------------------------------------------
  slack (VIOLATED)                                    -0.67
```

Setup timing violation reduced to –0.67 ns. Command **balance_registers** is executed.

```
dc_shell> current_design = microc_ent
dc_shell> balance_registers
dc_shell> report_timing -path full -delay max -
max_paths 1 -nworst 1
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : microc_ent
Version: 1998.02-1
Date   : Wed Mar 17 19:19:51 1999
****************************************

Operating Conditions:
Wire Loading Model Mode: top

Design                Wire Loading Model         Library
---------------------------------------------------------
microc_ent            20x20                      class

  Startpoint: microc_ent_REG894_S12
             (rising edge-triggered flip-flop clocked by
clock)
  Endpoint: microc_ent_REG381_S14
            (rising edge-triggered flip-flop clocked by
clock)
```

```
  Path Group: clock
  Path Type: max

  Point                                          Incr     Path
  -----------------------------------------------------------------
  clock clock (rise edge)                        0.00     0.00
  clock network delay (ideal)                    0.00     0.00
  microc_ent_REG894_S12/CP (FD1)                 0.00     0.00 r
  microc_ent_REG894_S12/Q (FD1)                  4.97     4.97 r
  DUT_execute/U609/Z (ND2I)                      0.15     5.12 f
  DUT_execute/U237/Z (ND2I)                      0.39     5.51 r
  U137/Z (IVI)                                   0.28     5.79 f
  DUT_execute/U212/Z (MUX21LP)                   0.77     6.56 r
  DUT_execute/mul_87/mult/mult/U283/Z2
(B3IP)                                           0.50     7.06 r
  DUT_execute/mul_87/mult/mult/U1661/Z
(ND2I)                                           0.25     7.31 f
  DUT_execute/mul_87/mult/mult/U1662/Z
(ND2I)                                           0.27     7.58 r
  DUT_execute/mul_87/mult/mult/U359/Z
(ND2I)                                           0.15     7.73 f
  DUT_execute/mul_87/mult/mult/U41/Z
(IVI)                                            0.33     8.06 r
  DUT_execute/mul_87/mult/mult/U2801/Z
(ND2I)                                           0.25     8.32 f
  DUT_execute/mul_87/mult/mult/U2799/Z
(ENI)                                            0.44     8.76 f
  DUT_execute/mul_87/mult/mult/U413/Z
(ENI)                                            0.56     9.32 f
  DUT_execute/U965/Z (ENI)                       0.44     9.77 f
  DUT_execute/U953/Z (ENI)                       0.58    10.35 f
  DUT_execute/mul_87/mult/mult/U2519/Z
(ENI)                                            0.49    10.84 f
  DUT_execute/mul_87/mult/mult/U845/Z
(ENI)                                            0.51    11.35 f
  DUT_execute/mul_87/mult/mult/U2402/Z
(IVI)                                            0.33    11.69 r
  DUT_execute/mul_87/mult/mult/U1873/Z
(ND2I)                                           0.15    11.84 f
  DUT_execute/mul_87/mult/mult/U1874/Z
(ND2I)                                           0.33    12.17 r
  DUT_execute/mul_87/mult/mult/U449/Z
(ENI)                                            0.56    12.72 f
  DUT_execute/mul_87/mult/mult/U2532/Z
(ENI)                                            0.56    13.28 f
  DUT_execute/mul_87/mult/mult/U37/Z
(ENI)                                            0.51    13.80 f
```

```
  DUT_execute/mul_87/mult/mult/U2538/Z
(ENI)                                        0.49    14.29 f
  DUT_execute/mul_87/mult/mult/U2430/Z
(ENI)                                        0.45    14.73 r
  DUT_execute/mul_87/mult/mult/U2177/Z
(ND2I)                                       0.38    15.12 f
  DUT_execute/mul_87/mult/mult/U1981/Z
(ND3P)                                       0.87    15.99 r
  DUT_execute/mul_87/mult/mult/U74/Z
(IVI)                                        0.20    16.19 f
  DUT_execute/mul_87/mult/mult/U2676/Z
(ENI)                                        0.58    16.77 f
  DUT_execute/mul_87/mult/mult/U2546/Z
(ENI)                                        0.51    17.28 f
  DUT_execute/mul_87/mult/mult/U398/Z
(ENI)                                        0.51    17.80 f
  DUT_execute/mul_87/mult/mult/U2045/Z
(ND2I)                                       0.27    18.07 r
  DUT_execute/mul_87/mult/mult/U2046/Z
(ND2I)                                       0.25    18.32 f
  DUT_execute/mul_87/mult/mult/U2047/Z
(ND2I)                                       0.33    18.65 r
  DUT_execute/mul_87/mult/mult/U1104/Z
(ND2I)                                       0.38    19.03 f
  DUT_execute/mul_87/mult/mult/U3026/Z
(ND3P)                                       0.76    19.80 r
  microc_ent_REG381_S14/D (FD1)              0.00    19.80 r
  data arrival time                                  19.80

  clock clock (rise edge)                   25.00    25.00
  clock network delay (ideal)                0.00    25.00
  microc_ent_REG381_S14/CP (FD1)             0.00    25.00 r
  library setup time                        -0.80    24.20
  data required time                                 24.20
  -----------------------------------------------------------
  data required time                                 24.20
  data arrival time                                 -19.80
  -----------------------------------------------------------
  slack (MET)                                         4.40
```

Timing violation of –3.85 ns is now optimized to be a positive slack of 4.40 ns.

APPENDIX E

(EDIF FILE OF SYNTHESIZED MICROCONTROLLER EXAMPLE FROM CHAPTER 6)

A sample example of the EDIF file from the synthesized microcontroller example of Chapter 6.

```
(edif Synopsys_edif (edifVersion 2 0 0) (edifLevel 0)
 (keywordMap (keywordLevel 0))
 (status
  (written (timeStamp 1999 4 5 20 11 34)
   (program "Synopsys Design Compiler" (Version "1998.02-1"))
   (dataOrigin "company") (author "designer")
  )
 )
 (external (rename generic_sdb "generic.sdb") (edifLevel 0)
  (technology (numberDefinition (scale 1 (e 2480469 -12) (unit DISTANCE)))
   (figureGroup default) (figureGroup text_layer (color 99 50 0))
   (figureGroup variable_layer (color 99 50 0))
   (figureGroup net_name_layer (color 100 100 100))
   (figureGroup constraint_layer (color 100 0 0))
   (figureGroup symbol_name_layer (color 100 100 100))
   (figureGroup designware_layer (color 100 0 0))
   (figureGroup bus_osc_layer (color 100 100 0))
   (figureGroup hierarchy_name_layer (color 100 100 100))
   (figureGroup bus_net_type_layer (color 99 50 0))
   (figureGroup bus_net_layer (color 0 59 100))
   (figureGroup template_layer (color 0 70 70))
   (figureGroup cell_layer (color 100 100 0))
   (figureGroup net_layer (color 0 100 100))
   (figureGroup osc_layer (color 100 100 0))
  (figureGroup hierarchy_layer (color 100 100 0))
  (figureGroup template_text_layer (color 100 100 100))
  (figureGroup fat_layer (color 99 0 0))
  (figureGroup port_layer (color 100 100 0))
  (figureGroup cell_name_layer (color 100 100 100))
  (figureGroup bus_cell_name_layer (color 100 100 100))
  (figureGroup designware_name_layer (color 100 100 100))
  (figureGroup cell_ref_name_layer (color 100 100 100))
  (figureGroup bus_port_width_layer (color 99 50 0))
  (figureGroup bus_net_name_layer (color 100 100 100))
  (figureGroup port_name_layer (color 100 100 100))
  (figureGroup bus_cell_layer (color 0 100 100))
  (figureGroup clock_layer (color 100 0 0))
  (figureGroup bit_mapper_name_layer (color 100 100 100))
  (figureGroup pin_layer (color 99 99 0))
  (figureGroup pin_name_layer (color 100 100 100))
  (figureGroup osc_name_layer (color 100 100 100))
  (figureGroup bus_port_name_layer (color 100 100 100))
  (figureGroup bus_pin_name_layer (color 100 100 100))
  (figureGroup bus_osc_name_layer (color 100 100 100))
  (figureGroup bus_ripper_type_layer (color 99 50 0))
  (figureGroup symbol_layer (color 100 100 0))
  (figureGroup type_mapper_name_layer (color 99 50 0))
  (figureGroup bus_compound_name_layer (color 100 100 100))
  (figureGroup bus_port_layer (color 100 100 0))
  (figureGroup bus_ripper_name_layer (color 100 100 100))
  (figureGroup bus_ripper_layer (color 100 100 0))
 )
 (cell ripper (cellType RIPPER)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port (array bus_end 32)) (port (array w 32))
    (joined (portRef bus_end) (portRef w))
   )
  )
 )
 (cell AN2I (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port B (direction INPUT))
    (port Z (direction OUTPUT))
   )
  )
 )
 (cell FD1S (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port D (direction INPUT)) (port CP (direction INPUT))
    (port TI (direction INPUT)) (port TE (direction INPUT))
    (port Q (direction OUTPUT)) (port QN (direction OUTPUT))
   )
  )
 )
 (cell IVDA (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port Y (direction OUTPUT))
    (port Z (direction OUTPUT))
   )
  )
 )
 (cell AN3P (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port B (direction INPUT))
    (port C (direction INPUT)) (port Z (direction OUTPUT))
   )
  )
 )
 (cell AO3 (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port B (direction INPUT))
    (port C (direction INPUT)) (port D (direction INPUT))
    (port Z (direction OUTPUT))
   )
  )
 )
 (cell IVI (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port Z (direction OUTPUT)))
  )
 )
 (cell EOI (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port B (direction INPUT))
    (port Z (direction OUTPUT))
   )
  )
 )
 (cell AO3P (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port A (direction INPUT)) (port B (direction INPUT))
    (port C (direction INPUT)) (port D (direction INPUT))
    (port Z (direction OUTPUT))
   )
  )
 )
```

```
(cell IV (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port Z (direction OUTPUT)))
 )
)
(cell ND3P (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port B (direction INPUT))
   (port C (direction INPUT)) (port Z (direction OUTPUT))
  )
 )
)
(cell IVDAP (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port Y (direction OUTPUT))
   (port Z (direction OUTPUT))
  )
 )
)
(cell OR2I (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port B (direction INPUT))
   (port Z (direction OUTPUT))
  )
 )
)
(cell AO7 (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port B (direction INPUT))
   (port C (direction INPUT)) (port Z (direction OUTPUT))
  )
 )
)
(cell AO7P (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port B (direction INPUT))
   (port C (direction INPUT)) (port Z (direction OUTPUT))
  )
 )
)
(cell B4IP (cellType GENERIC)
 (view Schematic_representation (viewType SCHEMATIC)
  (interface (port A (direction INPUT)) (port Z (direction OUTPUT)))
 )
)
(cell B5IP (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port Z (direction OUTPUT)))
   )
  )
  (cell ND3 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port C (direction INPUT)) (port Z (direction OUTPUT))
    )
   )
  )
  (cell ND4 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port C (direction INPUT)) (port D (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell NR2 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell ND4P (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port C (direction INPUT)) (port D (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell logic_0 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC) (interface (port a)))
  )
  (cell ENI (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell ND2 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell NR3 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port C (direction INPUT)) (port Z (direction OUTPUT))
    )
   )
  )
  (cell ND2I (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell OR2P (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell logic_1 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC) (interface (port a)))
  )
  (cell IVAP (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port Z (direction OUTPUT)))
   )
  )
  (cell NR2I (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port Z (direction OUTPUT))
    )
   )
  )
  (cell B4I (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port Z (direction OUTPUT)))
   )
  )
```

```
)
(external (rename class_sdb "class.sdb") (edifLevel 0)
 (technology (numberDefinition (scale 1 (e 2480469 -12) (unit DISTANCE)))
  (figureGroup default) (figureGroup text_layer (color 99 50 0))
  (figureGroup variable_layer (color 99 50 0))
  (figureGroup net_name_layer (color 100 100 100))
  (figureGroup constraint_layer (color 100 0 0))
  (figureGroup symbol_name_layer (color 100 100 100))
  (figureGroup designware_layer (color 100 0 0))
  (figureGroup bus_osc_layer (color 100 100 0))
  (figureGroup hierarchy_name_layer (color 100 100 100))
  (figureGroup bus_net_type_layer (color 99 50 0))
  (figureGroup bus_net_layer (color 0 59 100))
  (figureGroup template_layer (color 0 70 70))
  (figureGroup cell_layer (color 100 100 0))
  (figureGroup net_layer (color 0 100 100))
  (figureGroup osc_layer (color 100 100 0))
  (figureGroup hierarchy_layer (color 100 100 0))
  (figureGroup template_text_layer (color 100 100 100))
  (figureGroup fat_layer (color 99 0 0))
  (figureGroup port_layer (color 100 100 0))
  (figureGroup cell_name_layer (color 100 100 100))
  (figureGroup bus_cell_name_layer (color 100 100 100))
  (figureGroup designware_name_layer (color 100 100 100))
  (figureGroup cell_ref_name_layer (color 100 100 100))
  (figureGroup bus_port_width_layer (color 99 50 0))
  (figureGroup bus_net_name_layer (color 100 100 100))
  (figureGroup port_name_layer (color 100 100 100))
  (figureGroup bus_cell_layer (color 0 100 100))
  (figureGroup clock_layer (color 100 0 0))
  (figureGroup bit_mapper_name_layer (color 100 100 100))
  (figureGroup pin_layer (color 99 99 0))
  (figureGroup pin_name_layer (color 100 100 100))
  (figureGroup osc_name_layer (color 100 100 100))
  (figureGroup bus_port_name_layer (color 100 100 100))
  (figureGroup bus_pin_name_layer (color 100 100 100))
  (figureGroup bus_osc_name_layer (color 100 100 100))
  (figureGroup bus_ripper_type_layer (color 99 50 0))
  (figureGroup symbol_layer (color 100 100 0))
  (figureGroup type_mapper_name_layer (color 99 50 0))
  (figureGroup bus_compound_name_layer (color 100 100 100))
  (figureGroup bus_port_layer (color 100 100 0))
  (figureGroup bus_ripper_name_layer (color 100 100 100))
  (figureGroup bus_ripper_layer (color 100 100 0))
 )
 (cell B2I (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port Z1 (direction OUTPUT))
     (port Z2 (direction OUTPUT))
    )
   )
  )
  (cell B2IP (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port Z1 (direction OUTPUT))
     (port Z2 (direction OUTPUT))
    )
   )
  )
  (cell B3IP (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port Z1 (direction OUTPUT))
     (port Z2 (direction OUTPUT))
    )
   )
  )
  (cell FD1 (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port D (direction INPUT)) (port CP (direction INPUT))
     (port Q (direction OUTPUT)) (port QN (direction OUTPUT))
    )
   )
  )
  (cell MUX21H (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port S (direction INPUT)) (port Z (direction OUTPUT))
    )
   )
  )
  (cell MUX21HP (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port S (direction INPUT)) (port Z (direction OUTPUT))
    )
   )
  )
  (cell MUX21L (cellType GENERIC)
   (view Schematic_representation (viewType SCHEMATIC)
    (interface (port A (direction INPUT)) (port B (direction INPUT))
     (port S (direction INPUT)) (port Z (direction OUTPUT))
    )
    )
   )
   (cell MUX21LP (cellType GENERIC)
    (view Schematic_representation (viewType SCHEMATIC)
     (interface (port A (direction INPUT)) (port B (direction INPUT))
      (port S (direction INPUT)) (port Z (direction OUTPUT))
     )
    )
   )
  )
  (library DESIGNS (edifLevel 0)
   (technology (numberDefinition (scale 1 (e 2480469 -12) (unit DISTANCE)))
    (figureGroup default) (figureGroup text_layer (color 99 50 0))
    (figureGroup variable_layer (color 99 50 0))
    (figureGroup net_name_layer (color 100 100 100))
    (figureGroup constraint_layer (color 100 0 0))
    (figureGroup symbol_name_layer (color 100 100 100))
    (figureGroup designware_layer (color 100 0 0))
    (figureGroup bus_osc_layer (color 100 100 0))
    (figureGroup hierarchy_name_layer (color 100 100 100))
    (figureGroup bus_net_type_layer (color 99 50 0))
    (figureGroup bus_net_layer (color 0 59 100))
    (figureGroup template_layer (color 0 70 70))
    (figureGroup cell_layer (color 100 100 0))
    (figureGroup net_layer (color 0 100 100))
    (figureGroup osc_layer (color 100 100 0))
    (figureGroup hierarchy_layer (color 100 100 0))
    (figureGroup template_text_layer (color 100 100 100))
    (figureGroup fat_layer (color 99 0 0))
    (figureGroup port_layer (color 100 100 0))
    (figureGroup cell_name_layer (color 100 100 100))
    (figureGroup bus_cell_name_layer (color 100 100 100))
    (figureGroup designware_name_layer (color 100 100 100))
    (figureGroup cell_ref_name_layer (color 100 100 100))
    (figureGroup bus_port_width_layer (color 99 50 0))
    (figureGroup bus_net_name_layer (color 100 100 100))
    (figureGroup port_name_layer (color 100 100 100))
    (figureGroup bus_cell_layer (color 0 100 100))
    (figureGroup clock_layer (color 100 0 0))
    (figureGroup bit_mapper_name_layer (color 100 100 100))
    (figureGroup pin_layer (color 99 99 0))
    (figureGroup pin_name_layer (color 100 100 100))
    (figureGroup osc_name_layer (color 100 100 100))
    (figureGroup bus_port_name_layer (color 100 100 100))
    (figureGroup bus_pin_name_layer (color 100 100 100))
    (figureGroup bus_osc_name_layer (color 100 100 100))
  (figureGroup bus_ripper_type_layer (color 99 50 0))
  (figureGroup symbol_layer (color 100 100 0))
  (figureGroup type_mapper_name_layer (color 99 50 0))
  (figureGroup bus_compound_name_layer (color 100 100 100))
  (figureGroup bus_port_layer (color 100 100 0))
  (figureGroup bus_ripper_name_layer (color 100 100 100))
  (figureGroup bus_ripper_layer (color 100 100 0))
 )
 (cell microc_ent (cellType GENERIC)
  (view Schematic_representation (viewType SCHEMATIC)
   (interface (port clock (direction INPUT))
    (port (array (rename data_31_0_ "data[31:0]") 32) (direction INPUT))
    (port (array (rename destination_3_0_ "destination[3:0]") 4)
     (direction INPUT)
    )
    (port (array (rename inst_2_0_ "inst[2:0]") 3) (direction INPUT))
    (port jump (direction OUTPUT))
    (port (array (rename output_31_0_ "output[31:0]") 32) (direction OUTPUT))
    (port (array (rename source1_3_0_ "source1[3:0]") 4) (direction INPUT))
    (port (array (rename source2_3_0_ "source2[3:0]") 4) (direction INPUT))
    (symbol (boundingBox (rectangle (pt -3072 -6144) (pt 3072 6144)))
     (portImplementation clock
      (connectLocation (figure cell_layer (dot (pt -3072 5120))))
     )
     (portImplementation data_31_0_
      (connectLocation (figure cell_layer (dot (pt -3072 3072))))
     )
     (portImplementation destination_3_0_
      (connectLocation (figure cell_layer (dot (pt -3072 1024))))
     )
     (portImplementation inst_2_0_
      (connectLocation (figure cell_layer (dot (pt -3072 -1024))))
     )
     (portImplementation source1_3_0_
      (connectLocation (figure cell_layer (dot (pt -3072 -3072))))
     )
     (portImplementation source2_3_0_
      (connectLocation (figure cell_layer (dot (pt -3072 -5120))))
     )
     (portImplementation jump
      (connectLocation (figure cell_layer (dot (pt 3072 1024))))
     )
     (portImplementation output_31_0_
      (connectLocation (figure cell_layer (dot (pt 3072 -1024))))
     )
     (figure cell_layer (path (pointList (pt -3072 -6144) (pt -3072 6144))))
     (figure cell_layer (path (pointList (pt 3072 -6144) (pt 3072 6144))))
     (figure cell_layer (path (pointList (pt -3072 -6144) (pt 3072 -6144))))
     (figure cell_layer (path (pointList (pt -3072 6144) (pt 3072 6144))))
    )
   )
   (contents
    (page &1 (pageSize (rectangle (pt -11909424 -7680) (pt 11876 6322432)))
     (portImplementation clock
      (connectLocation (figure port_layer (dot (pt -11888640 6321152))))
      (figure port_layer
       (path (pointList (pt -11891200 6320512) (pt -11891200 6321792)))
      )
      (figure port_layer
       (path (pointList (pt -11891200 6321792) (pt -11889920 6321792)))
      )
      (figure port_layer
       (path (pointList (pt -11891200 6320512) (pt -11889920 6320512)))
      )
      (figure port_layer
       (path (pointList (pt -11889920 6320512) (pt -11888640 6321152)))
      )
      (figure port_layer
       (path (pointList (pt -11889920 6321792) (pt -11888640 6321152)))
      )
     )
     (portImplementation data_31_0_
      (connectLocation (figure bus_port_layer (dot (pt -11888640 6024192))))
      (figure bus_port_layer
       (path (pointList (pt -11891200 6023552) (pt -11891200 6024832)))
      )
      (figure bus_port_layer
       (path (pointList (pt -11891200 6024832) (pt -11889920 6024832)))
      )
      (figure bus_port_layer
       (path (pointList (pt -11891200 6023552) (pt -11889920 6023552)))
      )
      (figure bus_port_layer
       (path (pointList (pt -11889920 6023552) (pt -11888640 6024192)))
      )
      (figure bus_port_layer
       (path (pointList (pt -11889920 6024832) (pt -11888640 6024192)))
      )
     )
     (portImplementation destination_3_0_
      (connectLocation (figure bus_port_layer (dot (pt -11888640 2507776))))
      (figure bus_port_layer
   (path (pointList (pt -11891200 2507136) (pt -11891200 2508416)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11891200 2508416) (pt -11889920 2508416)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11891200 2507136) (pt -11889920 2507136)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11889920 2507136) (pt -11888640 2507776)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11889920 2508416) (pt -11888640 2507776)))
  )
 )
 (portImplementation inst_2_0_
  (connectLocation (figure bus_port_layer (dot (pt -11888640 2321408))))
  (figure bus_port_layer
   (path (pointList (pt -11891200 2320768) (pt -11891200 2322048)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11891200 2322048) (pt -11889920 2322048)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11891200 2320768) (pt -11889920 2320768)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11889920 2320768) (pt -11888640 2321408)))
  )
  (figure bus_port_layer
   (path (pointList (pt -11889920 2322048) (pt -11888640 2321408)))
  )
 )
 (portImplementation jump
  (connectLocation (figure port_layer (dot (pt -6144 2652160))))
  (figure port_layer
   (path (pointList (pt -6144 2651520) (pt -6144 2652800)))
  )
  (figure port_layer
   (path (pointList (pt -6144 2652800) (pt -4864 2652800)))
  )
  (figure port_layer
   (path (pointList (pt -6144 2651520) (pt -4864 2651520)))
  )
  (figure port_layer
   (path (pointList (pt -4864 2651520) (pt -3584 2652160)))
   )
   (figure port_layer
    (path (pointList (pt -4864 2652800) (pt -3584 2652160)))
   )
  )
  (portImplementation output_31_0_
   (connectLocation (figure bus_port_layer (dot (pt -6144 6309888))))
   (figure bus_port_layer
    (path (pointList (pt -6144 6309248) (pt -6144 6310528)))
   )
   (figure bus_port_layer
    (path (pointList (pt -6144 6310528) (pt -4864 6310528)))
   )
   (figure bus_port_layer
    (path (pointList (pt -6144 6309248) (pt -4864 6309248)))
   )
   (figure bus_port_layer
    (path (pointList (pt -4864 6309248) (pt -3584 6309888)))
   )
   (figure bus_port_layer
    (path (pointList (pt -4864 6310528) (pt -3584 6309888)))
   )
  )
  (portImplementation source1_3_0_
   (connectLocation (figure bus_port_layer (dot (pt -11888640 2688000))))
   (figure bus_port_layer
    (path (pointList (pt -11891200 2687360) (pt -11891200 2688640)))
   )
   (figure bus_port_layer
    (path (pointList (pt -11891200 2688640) (pt -11889920 2688640)))
   )
   (figure bus_port_layer
    (path (pointList (pt -11891200 2687360) (pt -11889920 2687360)))
   )
   (figure bus_port_layer
    (path (pointList (pt -11889920 2687360) (pt -11888640 2688000)))
   )
   (figure bus_port_layer
    (path (pointList (pt -11889920 2688640) (pt -11888640 2688000)))
   )
  )
  (portImplementation source2_3_0_
   (connectLocation (figure bus_port_layer (dot (pt -11888640 2706432))))
   (figure bus_port_layer
    (path (pointList (pt -11891200 2705792) (pt -11891200 2707072)))
   )
```

```
    (figure bus_port_layer
     (path (pointList (pt -11891200 2707072) (pt -11889920 2707072)))
    )
    (figure bus_port_layer
     (path (pointList (pt -11891200 2705792) (pt -11889920 2705792)))
    )
    (figure bus_port_layer
     (path (pointList (pt -11889920 2705792) (pt -11888640 2706432)))
    )
    (figure bus_port_layer
     (path (pointList (pt -11889920 2707072) (pt -11888640 2706432)))
    )
   )
   (instance Ripper_1
    (viewRef Schematic_representation
     (cellRef ripper (libraryRef generic_sdb))
    )
    (transform (orientation R180) (origin (pt -16384 5926912)))
   )
   (instance Ripper_2
    (viewRef Schematic_representation
     (cellRef ripper (libraryRef generic_sdb))
    )
    (transform (orientation R180) (origin (pt -16384 5939200)))
   )
   (instance Ripper_3
    (viewRef Schematic_representation
     (cellRef ripper (libraryRef generic_sdb))
    )
    (transform (orientation R180) (origin (pt -16384 5951488)))
   )
   (instance Ripper_4
    (viewRef Schematic_representation
     (cellRef ripper (libraryRef generic_sdb))
    )
    (transform (orientation R180) (origin (pt -16384 5963776)))
   )
   (instance Ripper_5
    (viewRef Schematic_representation
     (cellRef ripper (libraryRef generic_sdb))
    )
    (transform (orientation R180) (origin (pt -16384 5976064)))
   )
   (instance Ripper_6
    (viewRef Schematic_representation
     (cellRef ripper (libraryRef generic_sdb))
```

```
         )
         (transform (orientation R180) (origin (pt -16384 5988352)))
        )
        (instance Ripper_7
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
         (transform (orientation R180) (origin (pt -16384 6000640)))
        )
        (instance Ripper_8
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
         (transform (orientation R180) (origin (pt -16384 6012928)))
        )
        (instance Ripper_9
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
         (transform (orientation R180) (origin (pt -16384 6025216)))
        )
        (instance Ripper_10
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
         (transform (orientation R180) (origin (pt -16384 6037504)))
        )
        (instance Ripper_11
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
         (transform (orientation R180) (origin (pt -16384 6049792)))
        )
        (instance Ripper_12
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
         (transform (orientation R180) (origin (pt -16384 6062080)))
        )
        (instance Ripper_13
         (viewRef Schematic_representation
          (cellRef ripper (libraryRef generic_sdb))
         )
.........
.........
(net (rename DUT_execute_net15859 "DUT_execute/net15859")
  (joined (portRef A (instanceRef DUT_execute_U788))
   (portRef QN (instanceRef microc_ent_REG956_S10))
  )
  (figure net_layer
   (path (pointList (pt -11825152 2284544) (pt -11462656 2284544)))
   (path (pointList (pt -11462656 2284544) (pt -11462656 2702336)))
   (path (pointList (pt -11462656 2702336) (pt -11444224 2702336)))
  )
 )
 (net (rename DUT_register_file_n5676 "DUT_register_file/n5676")
  (joined (portRef TE (instanceRef microc_ent_REG883_S11))
   (portRef TE (instanceRef microc_ent_REG946_S11))
   (portRef TE (instanceRef microc_ent_REG944_S11))
   (portRef TE (instanceRef microc_ent_REG942_S11))
   (portRef QN (instanceRef microc_ent_REG18_S9))
   (portRef TE (instanceRef microc_ent_REG940_S11))
   (portRef B (instanceRef DUT_register_file_U3936))
   (portRef B (instanceRef DUT_register_file_U2596))
   (portRef TE (instanceRef microc_ent_REG954_S11))
   (portRef TE (instanceRef microc_ent_REG952_S11))
   (portRef TE (instanceRef microc_ent_REG950_S11))
   (portRef TE (instanceRef microc_ent_REG931_S11))
   (portRef TE (instanceRef microc_ent_REG948_S11))
   (portRef TE (instanceRef microc_ent_REG937_S11))
   (portRef TE (instanceRef microc_ent_REG934_S11))
   (portRef TE (instanceRef microc_ent_REG929_S11))
   (portRef B (instanceRef DUT_register_file_U2621))
  )
  (figure net_layer
   (path
    (pointList (pt -11833344 4096) (pt -11795456 4096)
     (pt -11773952 4096) (pt -11649024 4096) (pt -11619328 4096)
     (pt -11425792 4096) (pt -11410432 4096) (pt -8192 4096)
    )
   )
   (path (pointList (pt -8192 4096) (pt -8192 2643968)))
   (path (pointList (pt -10240 2643968) (pt -8192 2643968)))
   (path (pointList (pt -11410432 4096) (pt -11410432 2539520)))
   (path (pointList (pt -11410432 2539520) (pt -11409408 2539520)))
   (path (pointList (pt -11425792 4096) (pt -11425792 2540544)))
   (path (pointList (pt -11425792 2540544) (pt -11424768 2540544)))
   (path
    (pointList (pt -11619328 4096) (pt -11619328 2624512)
     (pt -11619328 2642944) (pt -11619328 2661376) (pt -11619328 2679808)
     (pt -11619328 2698240)
    )
```

```
          )
          (path (pointList (pt -11619328 2698240) (pt -11612160 2698240)))
          (path (pointList (pt -11619328 2624512) (pt -11612160 2624512)))
          (path (pointList (pt -11619328 2679808) (pt -11612160 2679808)))
          (path (pointList (pt -11619328 2661376) (pt -11612160 2661376)))
          (path (pointList (pt -11619328 2642944) (pt -11612160 2642944)))
          (path
           (pointList (pt -11649024 4096) (pt -11649024 2318336)
            (pt -11649024 2572288) (pt -11649024 2598912) (pt -11649024 2611200)
           )
          )
          (path (pointList (pt -11649024 2572288) (pt -11642880 2572288)))
          (path (pointList (pt -11649024 2611200) (pt -11642880 2611200)))
          (path (pointList (pt -11649024 2598912) (pt -11642880 2598912)))
          (path (pointList (pt -11649024 2318336) (pt -11642880 2318336)))
          (path
           (pointList (pt -11773952 4096) (pt -11773952 2272256)
            (pt -11773952 2499584)
           )
          )
          (path (pointList (pt -11773952 2499584) (pt -11771904 2499584)))
          (path (pointList (pt -11773952 2272256) (pt -11771904 2272256)))
          (path
           (pointList (pt -11795456 4096) (pt -11795456 2342912)
            (pt -11795456 2435072)
           )
          )
          (path (pointList (pt -11795456 2435072) (pt -11793408 2435072)))
          (path (pointList (pt -11795456 2342912) (pt -11793408 2342912)))
          (path (pointList (pt -11833344 4096) (pt -11833344 2298880)))
          (path (pointList (pt -11833344 2298880) (pt -11832320 2298880)))
         )
        )
        (net n184
         (joined (portRef B (instanceRef DUT_decode_U43))
          (portRef B (instanceRef DUT_decode_U40))
          (portRef A (instanceRef DUT_decode_U42))
          (portRef C (instanceRef DUT_decode_U28))
          (portRef Q (instanceRef microc_ent_REG0_S1))
         )
         (figure net_layer
          (path
           (pointList (pt -11835392 2185216) (pt -11703296 2185216)
            (pt -11672576 2185216) (pt -11209728 2185216)
           )
```

```
    (path (pointList (pt -11209728 2185216) (pt -11209728 2700288)))
    (path (pointList (pt -11209728 2700288) (pt -11206656 2700288)))
    (path (pointList (pt -11672576 2185216) (pt -11672576 2593792)))
    (path (pointList (pt -11672576 2593792) (pt -11669504 2593792)))
    (path (pointList (pt -11703296 2185216) (pt -11703296 2583552)))
    (path (pointList (pt -11703296 2583552) (pt -11695104 2583552)))
    (path
     (pointList (pt -11835392 2185216) (pt -11835392 2319360)
      (pt -11835392 2329600)
     )
    )
    (path (pointList (pt -11847680 2319360) (pt -11835392 2319360)))
    (path (pointList (pt -11835392 2329600) (pt -11832320 2329600)))
   )
  )
  (net (rename DUT_predecode_net20 "DUT_predecode/net20")
   (joined (portRef QN (instanceRef microc_ent_REG928_S10))
    (portRef B (instanceRef DUT_predecode_U41))
   )
   (figure net_layer
    (path (pointList (pt -11836416 2274304) (pt -11802624 2274304)))
    (path (pointList (pt -11802624 2274304) (pt -11802624 2341888)))
    (path (pointList (pt -11808768 2341888) (pt -11802624 2341888)))
    (path (pointList (pt -11836416 2274304) (pt -11836416 2349056)))
    (path (pointList (pt -11836416 2349056) (pt -11832320 2349056)))
   )
  )
  (net (rename DUT_predecode_net21 "DUT_predecode/net21")
   (joined (portRef QN (instanceRef microc_ent_REG930_S10))
    (portRef B (instanceRef DUT_predecode_U40))
   )
   (figure net_layer
    (path (pointList (pt -11838464 2273280) (pt -11801600 2273280)))
    (path (pointList (pt -11801600 2273280) (pt -11801600 2434048)))
    (path (pointList (pt -11808768 2434048) (pt -11801600 2434048)))
    (path (pointList (pt -11838464 2273280) (pt -11838464 2441216)))
    (path (pointList (pt -11838464 2441216) (pt -11832320 2441216)))
   )
  )
  (net (rename destination_3_ "destination[3]")
   (joined (portRef A (instanceRef DUT_predecode_U56))
    (portRef TI (instanceRef microc_ent_REG937_S11))
    (portRef (member w 0) (instanceRef Ripper_68))
   )
   (figure net_layer
    (path (pointList (pt -11840512 2267136) (pt -11774976 2267136)))
```

```
      (path (pointList (pt -11774976 2267136) (pt -11774976 2501632)))
      (path (pointList (pt -11774976 2501632) (pt -11771904 2501632)))
      (path (pointList (pt -11840512 2267136) (pt -11840512 2507776)))
      (path
       (pointList (pt -11852800 2507776) (pt -11840512 2507776)
        (pt -11840512 2507776) (pt -11832320 2507776)
       )
      )
     )
    )
    (net n183
     (joined (portRef TI (instanceRef microc_ent_REG883_S11))
      (portRef B (instanceRef DUT_decode_U42))
      (portRef B (instanceRef DUT_decode_U28))
      (portRef Q (instanceRef microc_ent_REG840_S10))
      (portRef A (instanceRef DUT_decode_U43))
      (portRef A (instanceRef DUT_decode_U40))
     )
     (figure net_layer
      (path
       (pointList (pt -11841536 2184192) (pt -11704320 2184192)
        (pt -11671552 2184192) (pt -11210752 2184192)
       )
      )
      (path (pointList (pt -11210752 2184192) (pt -11210752 2702336)))
      (path (pointList (pt -11210752 2702336) (pt -11206656 2702336)))
      (path (pointList (pt -11671552 2184192) (pt -11671552 2591744)))
      (path (pointList (pt -11671552 2591744) (pt -11669504 2591744)))
      (path (pointList (pt -11704320 2184192) (pt -11704320 2585600)))
      (path (pointList (pt -11704320 2585600) (pt -11701248 2585600)))
      (path
       (pointList (pt -11841536 2184192) (pt -11841536 2300928)
        (pt -11841536 2331648)
       )
      )
      (path
       (pointList (pt -11847680 2331648) (pt -11841536 2331648)
        (pt -11841536 2331648) (pt -11832320 2331648)
       )
      )
      (path (pointList (pt -11841536 2300928) (pt -11832320 2300928)))
     )
    )
    (net (rename destination_2_ "destination[2]")
     (joined (portRef TI (instanceRef microc_ent_REG934_S11))
      (portRef A (instanceRef DUT_predecode_U55))
      (portRef (member w 1) (instanceRef Ripper_67))
     )
     (figure net_layer
      (path (pointList (pt -11842560 2266112) (pt -11772928 2266112)))
      (path (pointList (pt -11772928 2266112) (pt -11772928 2274304)))
      (path (pointList (pt -11772928 2274304) (pt -11771904 2274304)))
      (path (pointList (pt -11842560 2266112) (pt -11842560 2280448)))
      (path
       (pointList (pt -11852800 2280448) (pt -11842560 2280448)
        (pt -11842560 2280448) (pt -11832320 2280448)
       )
      )
     )
    )
    (net (rename DUT_decode_n7351 "DUT_decode/n7351")
     (joined (portRef A (instanceRef DUT_decode_U29))
      (portRef B (instanceRef U243))
      (portRef B (instanceRef DUT_decode_U46))
      (portRef QN (instanceRef microc_ent_REG957_S10))
     )
     (figure net_layer
      (path
       (pointList (pt -11854848 2232320) (pt -11817984 2232320)
        (pt -11646976 2232320)
       )
      )
      (path (pointList (pt -11646976 2232320) (pt -11646976 2590720)))
      (path (pointList (pt -11646976 2590720) (pt -11642880 2590720)))
      (path (pointList (pt -11817984 2232320) (pt -11817984 2332672)))
      (path (pointList (pt -11817984 2332672) (pt -11815936 2332672)))
      (path
       (pointList (pt -11854848 2232320) (pt -11854848 2281472)
        (pt -11854848 2291712)
       )
      )
      (path (pointList (pt -11866112 2281472) (pt -11854848 2281472)))
      (path (pointList (pt -11854848 2291712) (pt -11853824 2291712)))
     )
    )
    (net (rename destination_1_ "destination[1]")
     (joined (portRef TI (instanceRef microc_ent_REG931_S11))
      (portRef A (instanceRef DUT_predecode_U54))
      (portRef (member w 2) (instanceRef Ripper_66))
     )
     (figure net_layer
      (path (pointList (pt -11858944 2272256) (pt -11796480 2272256)))
    (path (pointList (pt -11796480 2272256) (pt -11796480 2437120)))
    (path (pointList (pt -11796480 2437120) (pt -11793408 2437120)))
    (path (pointList (pt -11858944 2272256) (pt -11858944 2443264)))
    (path
     (pointList (pt -11872256 2443264) (pt -11858944 2443264)
      (pt -11858944 2443264) (pt -11853824 2443264)
     )
    )
   )
  )
  (net (rename destination_0_ "destination[0]")
   (joined (portRef A (instanceRef DUT_predecode_U53))
    (portRef TI (instanceRef microc_ent_REG929_S11))
    (portRef (member w 3) (instanceRef Ripper_65))
   )
   (figure net_layer
    (path (pointList (pt -11859968 2270208) (pt -11794432 2270208)))
    (path (pointList (pt -11794432 2270208) (pt -11794432 2344960)))
    (path (pointList (pt -11794432 2344960) (pt -11793408 2344960)))
    (path (pointList (pt -11859968 2270208) (pt -11859968 2351104)))
    (path
     (pointList (pt -11872256 2351104) (pt -11859968 2351104)
      (pt -11859968 2351104) (pt -11853824 2351104)
     )
    )
   )
  )
  (net (rename sig_command_0_ "sig_command[0]")
   (joined (portRef D (instanceRef microc_ent_REG840_S10))
    (portRef B (instanceRef DUT_predecode_U27))
    (portRef Z (instanceRef DUT_predecode_U26))
   )
   (figure net_layer
    (path (pointList (pt -11857920 2549760) (pt -11494400 2549760)))
    (path (pointList (pt -11857920 2331648) (pt -11857920 2549760)))
    (path
     (pointList (pt -11867136 2331648) (pt -11857920 2331648)
      (pt -11857920 2331648) (pt -11853824 2331648)
     )
    )
   )
  )
  (net (rename sig_command_2_ "sig_command[2]")
   (joined (portRef D (instanceRef microc_ent_REG0_S1))
    (portRef A (instanceRef DUT_predecode_U27))
    (portRef Z (instanceRef DUT_predecode_U25))
```

```
      )
      (figure net_layer
       (path (pointList (pt -11856896 2551808) (pt -11494400 2551808)))
       (path (pointList (pt -11856896 2319360) (pt -11856896 2551808)))
       (path
        (pointList (pt -11867136 2319360) (pt -11856896 2319360)
         (pt -11856896 2319360) (pt -11853824 2319360)
        )
       )
      )
     )
     (net n536
      (joined (portRef A (instanceRef U243)) (portRef A (instanceRef U244))
       (portRef Z (instanceRef U245))
      )
      (figure net_layer
       (path (pointList (pt -11860992 2278400) (pt -11834368 2278400)))
       (path (pointList (pt -11834368 2278400) (pt -11834368 2313216)))
       (path (pointList (pt -11834368 2313216) (pt -11832320 2313216)))
       (path (pointList (pt -11860992 2278400) (pt -11860992 2293760)))
       (path
        (pointList (pt -11869184 2293760) (pt -11860992 2293760)
         (pt -11860992 2293760) (pt -11853824 2293760)
        )
       )
      )
     )
     (net n438
      (joined (portRef B (instanceRef DUT_decode_U41))
       (portRef A (instanceRef DUT_decode_U28))
       (portRef Q (instanceRef microc_ent_REG957_S10))
      )
      (figure net_layer
       (path
        (pointList (pt -11862016 2183168) (pt -11705344 2183168)
         (pt -11211776 2183168)
        )
       )
       (path (pointList (pt -11211776 2183168) (pt -11211776 2705408)))
       (path (pointList (pt -11211776 2705408) (pt -11206656 2705408)))
       (path (pointList (pt -11705344 2183168) (pt -11705344 2587648)))
       (path (pointList (pt -11705344 2587648) (pt -11701248 2587648)))
       (path (pointList (pt -11862016 2183168) (pt -11862016 2289664)))
       (path (pointList (pt -11867136 2289664) (pt -11862016 2289664)))
      )
     )
```

```
(net (rename inst_1_ "inst[1]")
 (joined (portRef A (instanceRef DUT_predecode_U42))
  (portRef TE (instanceRef microc_ent_REG957_S10))
  (portRef (member w 1) (instanceRef Ripper_70))
 )
 (figure net_layer
  (path (pointList (pt -11876352 2182144) (pt -11213824 2182144)))
  (path (pointList (pt -11213824 2182144) (pt -11213824 2713600)))
  (path (pointList (pt -11213824 2713600) (pt -11206656 2713600)))
  (path (pointList (pt -11876352 2182144) (pt -11876352 2281472)))
  (path
   (pointList (pt -11880448 2281472) (pt -11876352 2281472)
    (pt -11876352 2281472) (pt -11873280 2281472)
   )
  )
 )
)
(net jump
 (joined (portRef jump) (portRef S (instanceRef DUT_predecode_U31))
  (portRef A (instanceRef DUT_decode_U44))
  (portRef S (instanceRef DUT_predecode_U32))
  (portRef S (instanceRef DUT_predecode_U39))
  (portRef A (instanceRef DUT_decode_U41))
  (portRef Q (instanceRef microc_ent_REG18_S9))
  (portRef S (instanceRef DUT_predecode_U38))
  (portRef S (instanceRef DUT_predecode_U36))
  (portRef S (instanceRef DUT_predecode_U40))
  (portRef S (instanceRef DUT_predecode_U33))
  (portRef D (instanceRef microc_ent_REG957_S10))
  (portRef A (instanceRef DUT_predecode_U44))
  (portRef S (instanceRef DUT_predecode_U35))
  (portRef S (instanceRef DUT_predecode_U30))
  (portRef B (instanceRef DUT_predecode_U42))
  (portRef S (instanceRef DUT_predecode_U41))
  (portRef S (instanceRef DUT_predecode_U37))
  (portRef S (instanceRef DUT_predecode_U34))
  (portRef A (instanceRef U245))
 )
 (figure net_layer
  (path
   (pointList (pt -11882496 3072) (pt -11874304 3072)
    (pt -11837440 3072) (pt -11819008 3072) (pt -11707392 3072)
    (pt -11677696 3072) (pt -11212800 3072) (pt -7168 3072)
   )
  )
  (path (pointList (pt -7168 3072) (pt -7168 2652160)))
```

```
(path
 (pointList (pt -11264 2652160) (pt -7168 2652160) (pt -7168 2652160)
  (pt -6144 2652160)
 )
)
(path
 (pointList (pt -11212800 3072) (pt -11212800 2707456)
  (pt -11212800 2711552)
 )
)
(path (pointList (pt -11212800 2711552) (pt -11206656 2711552)))
(path (pointList (pt -11212800 2707456) (pt -11206656 2707456)))
(path
 (pointList (pt -11677696 3072) (pt -11677696 2628608)
  (pt -11677696 2647040) (pt -11677696 2665472) (pt -11677696 2683904)
  (pt -11677696 2702336)
 )
)
(path (pointList (pt -11677696 2647040) (pt -11668480 2647040)))
(path (pointList (pt -11668480 2647040) (pt -11668480 2648064)))
(path (pointList (pt -11677696 2665472) (pt -11668480 2665472)))
(path (pointList (pt -11668480 2665472) (pt -11668480 2666496)))
(path (pointList (pt -11677696 2683904) (pt -11668480 2683904)))
(path (pointList (pt -11668480 2683904) (pt -11668480 2684928)))
(path (pointList (pt -11677696 2628608) (pt -11668480 2628608)))
(path (pointList (pt -11668480 2628608) (pt -11668480 2629632)))
(path (pointList (pt -11677696 2702336) (pt -11668480 2702336)))
(path (pointList (pt -11668480 2702336) (pt -11668480 2703360)))
(path
 (pointList (pt -11707392 3072) (pt -11707392 2576384)
  (pt -11707392 2603008) (pt -11707392 2615296)
 )
)
(path (pointList (pt -11707392 2603008) (pt -11700224 2603008)))
(path (pointList (pt -11700224 2603008) (pt -11700224 2604032)))
(path (pointList (pt -11707392 2615296) (pt -11700224 2615296)))
(path (pointList (pt -11700224 2615296) (pt -11700224 2616320)))
(path (pointList (pt -11707392 2576384) (pt -11700224 2576384)))
(path (pointList (pt -11700224 2576384) (pt -11700224 2577408)))
(path
 (pointList (pt -11819008 3072) (pt -11819008 2276352)
  (pt -11819008 2337792) (pt -11819008 2503680)
 )
)
(path (pointList (pt -11819008 2337792) (pt -11815936 2337792)))
(path (pointList (pt -11819008 2276352) (pt -11814912 2276352)))
```

```
        (path (pointList (pt -11814912 2276352) (pt -11814912 2277376)))
        (path (pointList (pt -11819008 2503680) (pt -11814912 2503680)))
        (path (pointList (pt -11814912 2503680) (pt -11814912 2504704)))
        (path
         (pointList (pt -11837440 3072) (pt -11837440 2347008)
          (pt -11837440 2439168)
         )
        )
        (path (pointList (pt -11837440 2347008) (pt -11831296 2347008)))
        (path (pointList (pt -11831296 2347008) (pt -11831296 2348032)))
        (path (pointList (pt -11837440 2439168) (pt -11831296 2439168)))
        (path (pointList (pt -11831296 2439168) (pt -11831296 2440192)))
        (path
         (pointList (pt -11874304 3072) (pt -11874304 2289664)
          (pt -11874304 2293760)
         )
        )
        (path (pointList (pt -11874304 2293760) (pt -11873280 2293760)))
        (path (pointList (pt -11874304 2289664) (pt -11873280 2289664)))
        (path (pointList (pt -11882496 3072) (pt -11882496 2314240)))
        (path (pointList (pt -11882496 2314240) (pt -11881472 2314240)))
       )
      )
      (net (rename inst_0_ "inst[0]")
       (joined (portRef A (instanceRef DUT_predecode_U28))
        (portRef B (instanceRef DUT_predecode_U43))
        (portRef (member w 2) (instanceRef Ripper_69))
       )
       (figure net_layer
        (path (pointList (pt -11883520 2181120) (pt -11214848 2181120)))
        (path (pointList (pt -11214848 2181120) (pt -11214848 2717696)))
        (path (pointList (pt -11214848 2717696) (pt -11206656 2717696)))
        (path (pointList (pt -11883520 2181120) (pt -11883520 2332672)))
        (path
         (pointList (pt -11885568 2332672) (pt -11883520 2332672)
          (pt -11883520 2332672) (pt -11881472 2332672)
         )
        )
       )
      )
      (net (rename inst_2_ "inst[2]")
       (joined (portRef A (instanceRef DUT_predecode_U29))
        (portRef A (instanceRef DUT_predecode_U43))
        (portRef (member w 0) (instanceRef Ripper_71))
       )
       (figure net_layer
        (path (pointList (pt -11884544 2180096) (pt -11215872 2180096)))
        (path (pointList (pt -11215872 2180096) (pt -11215872 2719744)))
        (path (pointList (pt -11215872 2719744) (pt -11206656 2719744)))
        (path (pointList (pt -11884544 2180096) (pt -11884544 2320384)))
        (path
         (pointList (pt -11885568 2320384) (pt -11884544 2320384)
          (pt -11884544 2320384) (pt -11881472 2320384)
         )
        )
       )
      )
     )
    )
   )
  )
 )
(design Synopsys_edif (cellRef microc_ent (libraryRef DESIGNS))))
```

APPENDIX F

(SDF FILE FROM SYNTHESIZED MICROCONTROLLER EXAMPLE OF CHAPTER 6)

A sample of the SDF file obtained from the synthesized microcontroller example of Chapter 6.

```
(DELAYFILE
(SDFVERSION "OVI 1.0")
(DESIGN "microc_ent")
(DATE "Thu Apr  8 20:06:25 1999")
(VENDOR "class")
(PROGRAM "Synopsys Design Compiler cmos")
(VERSION "1998.02-1")
(DIVIDER /)
(VOLTAGE 5.00:5.00:5.00)
(PROCESS)
(TEMPERATURE 25.00:25.00:25.00)
(TIMESCALE 1ns)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U56)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U55)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U54)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U53)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U52)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U51)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U50)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U49)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U48)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U47)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U46)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U45)
  (DELAY
    (ABSOLUTE
      (IOPATH A Z (0.307:0.307:0.307) (0.200:0.200:0.200))
      )
    )
  )
  (CELL
    (CELLTYPE "IVI")
    (INSTANCE DUT_predecode\/U44)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.331:0.331:0.331) (0.228:0.228:0.228))
      )
    )
  )
  (CELL
    (CELLTYPE "AN2I")
    (INSTANCE DUT_predecode\/U43)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.362:0.362:0.362) (0.617:0.617:0.617))
      (IOPATH B Z (0.362:0.362:0.362) (0.617:0.617:0.617))
      )
    )
  )
  (CELL
    (CELLTYPE "NR2I")
    (INSTANCE DUT_predecode\/U42)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.646:0.646:0.646) (0.222:0.222:0.222))
      (IOPATH B Z (0.646:0.646:0.646) (0.222:0.222:0.222))
      )
    )
  )
  (CELL
    (CELLTYPE "MUX21L")
    (INSTANCE DUT_predecode\/U41)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
      (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
      (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
      )
    )
  )
  (CELL
    (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U40)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U39)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U38)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U37)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U36)
  (DELAY
        (ABSOLUTE
        (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
        )
      )
    )
    (CELL
      (CELLTYPE "MUX21L")
      (INSTANCE DUT_predecode\/U35)
      (DELAY
        (ABSOLUTE
        (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
        )
      )
    )
    (CELL
      (CELLTYPE "MUX21L")
      (INSTANCE DUT_predecode\/U34)
      (DELAY
        (ABSOLUTE
        (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
        )
      )
    )
    (CELL
      (CELLTYPE "MUX21L")
      (INSTANCE DUT_predecode\/U33)
      (DELAY
        (ABSOLUTE
        (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
        (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
        )
      )
    )
    (CELL
      (CELLTYPE "MUX21L")
      (INSTANCE DUT_predecode\/U32)
      (DELAY
        (ABSOLUTE
        (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U31)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "MUX21L")
  (INSTANCE DUT_predecode\/U30)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH B Z (0.481:0.481:0.481) (0.467:0.467:0.467))
    (IOPATH S Z (0.881:0.881:0.881) (0.667:0.667:0.667))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U29)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.262:0.262:0.262) (0.147:0.147:0.147))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_predecode\/U28)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.262:0.262:0.262) (0.147:0.147:0.147))
    )
  )
)
(CELL
    (CELLTYPE "ND2I")
    (INSTANCE DUT_predecode\/U27)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
      (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
      )
    )
  )
  (CELL
    (CELLTYPE "ND2I")
    (INSTANCE DUT_predecode\/U26)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.328:0.328:0.328) (0.253:0.253:0.253))
      (IOPATH B Z (0.328:0.328:0.328) (0.253:0.253:0.253))
      )
    )
  )
  (CELL
    (CELLTYPE "ND2I")
    (INSTANCE DUT_predecode\/U25)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.328:0.328:0.328) (0.253:0.253:0.253))
      (IOPATH B Z (0.328:0.328:0.328) (0.253:0.253:0.253))
      )
    )
  )
  (CELL
    (CELLTYPE "ND2I")
    (INSTANCE DUT_predecode\/U24)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
      (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
      )
    )
  )
  (CELL
    (CELLTYPE "IVI")
    (INSTANCE DUT_predecode\/U21)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (2.387:2.387:2.387) (2.655:2.655:2.655))
      )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_decode\/U46)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_decode\/U45)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.537:0.537:0.537) (0.471:0.471:0.471))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_decode\/U44)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.331:0.331:0.331) (0.228:0.228:0.228))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_decode\/U43)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "NR2I")
  (INSTANCE DUT_decode\/U42)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.646:0.646:0.646) (0.222:0.222:0.222))
```

```
      (IOPATH B Z (0.646:0.646:0.646) (0.222:0.222:0.222))
      )
    )
  )
  (CELL
    (CELLTYPE "NR2I")
    (INSTANCE DUT_decode\/U41)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.646:0.646:0.646) (0.222:0.222:0.222))
      (IOPATH B Z (0.646:0.646:0.646) (0.222:0.222:0.222))
      )
    )
  )
  (CELL
    (CELLTYPE "AN2I")
    (INSTANCE DUT_decode\/U40)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.362:0.362:0.362) (0.617:0.617:0.617))
      (IOPATH B Z (0.362:0.362:0.362) (0.617:0.617:0.617))
      )
    )
  )
  (CELL
    (CELLTYPE "ND2I")
    (INSTANCE DUT_decode\/U39)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.445:0.445:0.445) (0.455:0.455:0.455))
      (IOPATH B Z (0.445:0.445:0.445) (0.455:0.455:0.455))
      )
    )
  )
  (CELL
    (CELLTYPE "ND2I")
    (INSTANCE DUT_decode\/U38)
    (DELAY
      (ABSOLUTE
      (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
      (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
      )
    )
  )
  (CELL
    (CELLTYPE "ND2I")
  (INSTANCE DUT_decode\/U37)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.445:0.445:0.445) (0.455:0.455:0.455))
    (IOPATH B Z (0.445:0.445:0.445) (0.455:0.455:0.455))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_decode\/U34)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_decode\/U31)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (2.387:2.387:2.387) (2.655:2.655:2.655))
    )
  )
)
(CELL
  (CELLTYPE "IVI")
  (INSTANCE DUT_decode\/U30)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.262:0.262:0.262) (0.147:0.147:0.147))
    )
  )
)
(CELL
  (CELLTYPE "OR2I")
  (INSTANCE DUT_decode\/U29)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.595:0.595:0.595) (0.665:0.665:0.665))
    (IOPATH B Z (0.595:0.595:0.595) (0.665:0.665:0.665))
    )
  )
)
```

```
(CELL
  (CELLTYPE "AO7")
  (INSTANCE DUT_decode\/U28)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (1.006:1.006:1.006) (0.523:0.523:0.523))
    (IOPATH B Z (1.006:1.006:1.006) (0.523:0.523:0.523))
    (IOPATH C Z (1.006:1.006:1.006) (0.523:0.523:0.523))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_register_file\/U5600)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_register_file\/U5599)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_register_file\/U5598)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_register_file\/U5597)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
)
(CELL
  (CELLTYPE "ND2I")
  (INSTANCE DUT_register_file\/U5596)
  (DELAY
    (ABSOLUTE
    (IOPATH A Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    (IOPATH B Z (0.270:0.270:0.270) (0.152:0.152:0.152))
    )
  )
...............
...............
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG33_S17)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG29_S16)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
```

```
        (SETUP TI (posedge CP) (1.300:1.300:1.300))
        (HOLD TI (posedge CP) (0.300:0.300:0.300))
        (SETUP TE (posedge CP) (1.300:1.300:1.300))
        (HOLD TE (posedge CP) (0.300:0.300:0.300))
      )
    )
    (CELL
      (CELLTYPE "FD1S")
      (INSTANCE microc_ent_REG25_S15)
      (DELAY
        (ABSOLUTE
        (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
        (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
        )
      )
      (TIMINGCHECK
        (SETUP D (posedge CP) (1.300:1.300:1.300))
        (HOLD D (posedge CP) (0.300:0.300:0.300))
        (SETUP TI (posedge CP) (1.300:1.300:1.300))
        (HOLD TI (posedge CP) (0.300:0.300:0.300))
        (SETUP TE (posedge CP) (1.300:1.300:1.300))
        (HOLD TE (posedge CP) (0.300:0.300:0.300))
      )
    )
    (CELL
      (CELLTYPE "FD1S")
      (INSTANCE microc_ent_REG755_S11)
      (DELAY
        (ABSOLUTE
        (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
        (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
        )
      )
      (TIMINGCHECK
        (SETUP D (posedge CP) (1.300:1.300:1.300))
        (HOLD D (posedge CP) (0.300:0.300:0.300))
        (SETUP TI (posedge CP) (1.300:1.300:1.300))
        (HOLD TI (posedge CP) (0.300:0.300:0.300))
        (SETUP TE (posedge CP) (1.300:1.300:1.300))
        (HOLD TE (posedge CP) (0.300:0.300:0.300))
      )
    )
    (CELL
      (CELLTYPE "FD1S")
      (INSTANCE microc_ent_REG15_S8)
      (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG11_S7)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG8_S6)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG164_S16)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG4_S5)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG709_S14)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
```

```
      (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
      (SETUP TI (posedge CP) (1.300:1.300:1.300))
      (HOLD TI (posedge CP) (0.300:0.300:0.300))
      (SETUP TE (posedge CP) (1.300:1.300:1.300))
      (HOLD TE (posedge CP) (0.300:0.300:0.300))
    )
  )
  (CELL
    (CELLTYPE "FD1S")
    (INSTANCE microc_ent_REG185_S15)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
      (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
      (SETUP TI (posedge CP) (1.300:1.300:1.300))
      (HOLD TI (posedge CP) (0.300:0.300:0.300))
      (SETUP TE (posedge CP) (1.300:1.300:1.300))
      (HOLD TE (posedge CP) (0.300:0.300:0.300))
    )
  )
  (CELL
    (CELLTYPE "FD1S")
    (INSTANCE microc_ent_REG205_S15)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
      (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
      (SETUP TI (posedge CP) (1.300:1.300:1.300))
      (HOLD TI (posedge CP) (0.300:0.300:0.300))
      (SETUP TE (posedge CP) (1.300:1.300:1.300))
      (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG229_S15)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG251_S15)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG354_S15)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG385_S15)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG436_S15)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
```

```
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG488_S15)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG508_S14)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG528_S14)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG20_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.215:1.215:1.215) (1.415:1.415:1.415))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
  (INSTANCE microc_ent_REG882_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.507:1.507:1.507) (1.520:1.520:1.520))
    (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (1.300:1.300:1.300))
    (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1S")
```

```
    (INSTANCE microc_ent_REG881_S11)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.507:1.507:1.507) (1.520:1.520:1.520))
      (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
      (SETUP TI (posedge CP) (1.300:1.300:1.300))
      (HOLD TI (posedge CP) (0.300:0.300:0.300))
      (SETUP TE (posedge CP) (1.300:1.300:1.300))
      (HOLD TE (posedge CP) (0.300:0.300:0.300))
    )
  )
  (CELL
    (CELLTYPE "FD1S")
    (INSTANCE microc_ent_REG880_S11)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.507:1.507:1.507) (1.520:1.520:1.520))
      (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
      (SETUP TI (posedge CP) (1.300:1.300:1.300))
      (HOLD TI (posedge CP) (0.300:0.300:0.300))
      (SETUP TE (posedge CP) (1.300:1.300:1.300))
      (HOLD TE (posedge CP) (0.300:0.300:0.300))
    )
  )
  (CELL
    (CELLTYPE "FD1S")
    (INSTANCE microc_ent_REG879_S11)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.507:1.507:1.507) (1.520:1.520:1.520))
      (IOPATH CP QN (1.590:1.590:1.590) (1.570:1.570:1.570))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
    (SETUP TI (posedge CP) (1.300:1.300:1.300))
    (HOLD TI (posedge CP) (0.300:0.300:0.300))
    (SETUP TE (posedge CP) (1.300:1.300:1.300))
    (HOLD TE (posedge CP) (0.300:0.300:0.300))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG938_S12)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (2.007:2.007:2.007) (1.520:1.520:1.520))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG935_S12)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (2.007:2.007:2.007) (1.520:1.520:1.520))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG932_S12)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (2.007:2.007:2.007) (1.520:1.520:1.520))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG1_S2)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (2.007:2.007:2.007) (1.520:1.520:1.520))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG869_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG870_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG871_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG872_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG873_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG874_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG875_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG876_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG852_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG877_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG863_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG864_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
```

```
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG865_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG866_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG867_S11)
  (DELAY
    (ABSOLUTE
    (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
    (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
    )
  )
  (TIMINGCHECK
    (SETUP D (posedge CP) (0.800:0.800:0.800))
    (HOLD D (posedge CP) (0.400:0.400:0.400))
  )
)
(CELL
  (CELLTYPE "FD1")
  (INSTANCE microc_ent_REG868_S11)
  (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
      (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (0.800:0.800:0.800))
      (HOLD D (posedge CP) (0.400:0.400:0.400))
    )
  )
  (CELL
    (CELLTYPE "FD1")
    (INSTANCE microc_ent_REG878_S11)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
      (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (0.800:0.800:0.800))
      (HOLD D (posedge CP) (0.400:0.400:0.400))
    )
  )
  (CELL
    (CELLTYPE "FD1")
    (INSTANCE microc_ent_REG956_S10)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.361:1.361:1.361) (1.467:1.467:1.467))
      (IOPATH CP QN (1.861:1.861:1.861) (1.467:1.467:1.467))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (0.800:0.800:0.800))
      (HOLD D (posedge CP) (0.400:0.400:0.400))
    )
  )
  (CELL
    (CELLTYPE "FD1S")
    (INSTANCE microc_ent_REG883_S11)
    (DELAY
      (ABSOLUTE
      (IOPATH CP Q (1.090:1.090:1.090) (1.370:1.370:1.370))
      (IOPATH CP QN (2.458:2.458:2.458) (1.881:1.881:1.881))
      )
    )
    (TIMINGCHECK
      (SETUP D (posedge CP) (1.300:1.300:1.300))
      (HOLD D (posedge CP) (0.300:0.300:0.300))
      (SETUP TI (posedge CP) (1.300:1.300:1.300))
      (HOLD TI (posedge CP) (0.300:0.300:0.300))
      (SETUP TE (posedge CP) (1.300:1.300:1.300))
      (HOLD TE (posedge CP) (0.300:0.300:0.300))
    )
  )
  )
```

GLOSSARY

Architecture Portion of VHDL code whereby the implementation of a design is specified

ASIC Application Specific Integrated Circuit

behavioral VHDL A form of VHDL that is commonly used for simulation purposes

BIT A signal or port type that can have only values of 1 and 0

black box A design in which only the interface characteristics of the design are known; the internal circuitry is unknown

Boolean A signal or port type that can have only values of TRUE or FALSE

CAD Computer-Aided Design

chips Generic name for integrated circuits

component Logic block of a design

component inference A concept of inferring a component during synthesis

configuration Portion of VHDL code that declares usage of architecture for multiarchitectural design; also used to specify declaration of instantiated components in the architecture portion

`CONV_INTEGER` A function from IEEE library

`CONV_STD_LOGIC_VECTOR` A function from IEEE library

descriptive VHDL A form of VHDL that describes the functions of a logic block; this form of VHDL code is synthesizable

Design Compiler Name of synthesis tool from the Synopsys company

design constraint A set of constraints set on a design; outcome on the design synthesis depends widely upon design constraints

EDA Electronic Design Automation

entity Portion of VHDL code to declare interface ports of a design

FPGA Field Programmable Gate Array

FPGA Compiler Synopsys's FPGA Compiler for compilation into FPGA database

functional block Designs are always partitioned according to functional blocks whereby each block has different functionality that the block has to perform

Hold time The time required for a signal to be held valid after clock changes

Hold time violation A violation whereby the hold time is not met

IEEE Institute of Electrical and Electronic Engineers

instantiate Concept of usage of precompiled logic components within a design

Integer A signal type that can have value range from $-(2^{31} - 1)$ to $+(2^{31} - 1)$

Karnaugh Map A method that can be used to optimize a design

Leonardo Mentor Graphics' synthesis related tool

memory module Module with internal circuitry that functions as a memory unit

Mentor Graphics An EDA tool company that has a wide range of EDA tools

microcontroller An integrated circuit that has functions of a microprocessor but on a much smaller scale; internally the microcontroller also has peripheral modules that each has its own functionality to perform

multiple architecture Design with more than one architecture specified

pipeline An architectural design concept whereby a design is divided into stages; at each stage some form of functionality is performed; the design would seem to function like a pipeline where inputs are passed from one stage to another

precompiled library An existing library with design components that have been compiled

schematic capture A concept on which designs are manually hand-drawn using some CAD tools

Setup time The time required for a signal to be held valid before clock changes

Setup time violation A violation whereby the setup time is not met

signal A VHDL concept which is equivalent to that of "wire" in Verilog

simulation To inject a certain set of input stimulus and check for output waveform to assure correct functionality

`Std_logic` A signal or port type that is the resolved version of `std_ulogic`

`std_logic_1164` A library from IEEE

`std_logic_arith` A library from IEEE

`Std_ulogic` A signal or port type that can have 9 different values of 1, 0, H, L, X, U, Z, – and W

stimulus A set of input patterns injected into a design

structural VHDL A form of VHDL that is commonly used for netlist purposes

submodule Module within module of a design

Summit Design Name of an EDA company that has a wide range of design tools

Synopsys Name of a company whose synthesis tool is most popular in the market today; it also carries a wide range of other design tools

synthesis Concept of using EDA tools to convert HDL code into logic circuit

synthesis flow Design flow that involves synthesis

synthesizable code HDL code that is written for synthesis; not all HDL codes are synthesizable

synthesizable VHDL A form of VHDL that is synthesized by synthesis tools into logic circuits

Test Compiler Synopsys's Test Compiler for testability synthesis

testbench A wraparound of a design to enable simulation of the design

VHDL Very High Speed Integrated Circuit Hardware Description Language or short for VHSIC HDL

Visual HDL Graphical design tool from Summit Design

BIBLIOGRAPHY

DesignWare Developer Guide

Devadas, S., Ghash, A., and Keutzer, K. (1993). *Logic Synthesis.* New York: McGraw-Hill.

Hennessy, J.L. and Patterson, D.A. (1996). *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publication.

Kurup, P. and Abbasi, T. (1997). *Logic Synthesis Using Synopsys.* New York: Kluwer Academic Publication.

Patterson, D.A. and Hennessy, J.L. (1997). *Computer Organization & Design: The Hardware/Software Interface.* Morgan Kaufmann Publication.

Synopsys Design Compiler Manual 1998

Xilinx XC4000 Series Design Methodology Using FPGA Compiler Application Note 1994

This Page Intentionally Left Blank

INDEX