8-2003

Design and Verification of the Data Encryption Standard for ASICs and FPGAs

Xiaoquan Fu
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Xiaoquan Fu entitled "Design and Verification of the Data Encryption Standard for ASICs and FPGAs." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin, Major Professor

We have read this thesis and recommend its acceptance:

Gregory D. Peterson, Daniel B. Koch

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Xiaoquan Fu entitled "Design and Verification of the Data Encryption Standard for ASICs and FPGAs". I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin, Major Professor

We have read this thesis and
recommend its acceptance:

Gregory D. Peterson
(Electrical Engineering)

Daniel B. Koch
(Electrical Engineering)

Accepted for the Council:

Dr. Anne Mayhew (Vice Provost and Dean of Graduate Studies)

(Original signatures are on file with official student records)

DESIGN AND VERIFICATION OF THE DATA ENCRYPTION STANDARD FOR ASICS AND FPGAS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Xiaoquan Fu
August, 2003

Acknowledgments

Many thanks are necessary because I have been so fortunate to meet so many nice and cordial people during my graduate study in the Department of Electrical Engineering.

First of all, I wish to thank my advisor, Dr. Donald Bouldin, for his particular patience, kindness and advice to make it possible for me to finish this thesis. I would also like to thank the members of my graduate committee, Dr. Gregory Peterson and Dr. Daniel Koch for their support during the course of this thesis work.

I would also like to thank my friends Dr. Chandra Tan and Fuat Karakaya in the Microelectronic Systems Lab. Their sincere support helped me out during some of my toughest time.

My greatest thanks go to my dead father, my mother and my wife. My knowledgeable, kindhearted and honest father is the person who affected me the greatest in the world. With his nonstop perseverance, he made me understand that one should keep his enterprising spirit for his entire life. From my firm and indomitable mother I learned how to face and overcome all obstacles with courage and persistence. My nice and sweet wife is a big spiritual support and comfort for me. Every bit of my success belongs to them.

Abstract

Encryption and decryption in a communication channel are used to provide security. In this thesis, the Data Encryption Standard (DES) is implemented with both FPGAs and ASICs. Different versions of FPGAs from different synthesis tools are compared. For ASIC implementation, the design space is explored to compile the design with different optimization targets like size, speed and power. Physical realizations of the design are obtained with Cadence tools. Simulations are made at each level to verify the implementations.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of this thesis was to demonstrate the design flow for developing both FPGA and ASIC implementations of an application initially described in a hardware description language. Exercising the design flow with a variety of tools and constraints for an application requiring tens of thousands of logic gates to implement is not a simple task. Documenting this effort can serve as a tutorial to others.

Many people wish to communicate privately. To prevent unauthorized persons from extracting information from the communication channel or injecting misinformation into the communication channel, messages need to be disguised by encryption. At the transmitter, the plaintext is encrypted to produce the ciphertext. The ciphertext is transmitted over an insecure channel to the receiver. The receiver then decrypts the ciphertext to obtain the original plaintext.

DES, which stands for Data Encryption Standard [17], is a block encryption algorithm adopted by the National Bureau of Standards. With this algorithm, a 64-bit plaintext and a 64-bit key are provided as input. By applying a sequence of initial permutation, switch, shift on the key and plaintext, the 64-bit ciphertext is generated at the output after 16 clock cycles.

The VHDL code to implement the DES algorithm in this thesis was downloaded from the website [4]. The original code could not be compiled with our toolset so the code was

modified and some parts were rewritten to make the code synthesizable. It should be noted that the resulting VHDL code can only be synthesized with the 1993 standard when using Synopsys synthesis tools.

A test bench for simulation is critically important for the final success of the whole work. The testbench code for DES was found and downloaded from the web to verify each step of the implementation. This test bench provides a sequence of keys and plaintext to the DES design, along with a sequence of expected ciphertext. After each 16 clock cycles, the generated ciphertext from the output of the design is compared with the expected ciphertext. If they are the same, the simulation succeeds and a success message is reported. Otherwise, the simulation fails and an error message is reported. In that case, it is necessary to go back to check the previous steps for errors.

With the test bench, the pre-synthesis simulation is then made using ModelSim. This is an RTL level simulation which verifies the logic functionality of the code without gate-level information involved. After the successful pre-synthesis simulation, Mentor Graphics's Leonardo Spectrum is used to synthesize the DES design with some compile constraints applied to generate the EDIF (Electronic Design Interchange Format) file, which is a FPGA-independent netlist file. The Xilinx Place & Route (P&R) tools take the EDIF file and implement it in the targeted Virtex V1000ebg560 device. With the gate-delay and wire-delay information obtained from the files generated during the P&R process, the post-layout simulation is made using ModelSim again.

Synopsys's FPGA COMPILER II (FC2) is used as another synthesis tool to compile the DES design to Vertex V1000ebg560 device. The two versions of implementation from Leonardo Spectrum and FC2 are compared in layout, speed, and size, etc.

The DES design is also synthesized to a testable Standard-Cell-Based ASIC with TSMC18 process. Synopsys's Design Compiler is used as the tool to explore the design space to compile the DES design to different versions that are respectively optimized to area, speed and

power. Cadence's tools are used to place and route the designs to get the physical layout. Finally, the post-layout simulation is made again to verify the implementation.

Chapter 2

Background

2.1 The Data Encryption Standard

From early times people have often wished to communicate privately. To prevent unauthorized persons from extracting information from the communication channel or injecting misinformation into the communication channel, messages need to be disguised. The term "encrypt" refers to the message transformation performed at the transmitter and the term "decrypt" refers to the inverse transformation performed at the receiver.

At the transmitter, the plaintext is encrypted by the use of an invertible transformation to produce the ciphertext. The ciphertext is transmitted over an insecure channel to the receiver. The authorized receiver obtains ciphertext and decrypts it with the inverse transformation to obtain the original plaintext message.

In 1977, the National Bureau of Standards adopted a modified Lucifer system as the national Data Encryption Standard (DES), which is regarded as a block encryption system. Figure 2.1 illustrates the system functions of DES in block diagram form. The encryption algorithm starts with an initial permutation (IP), which simply switches the position of the 64 plaintext bits. The subsequent 16 iterations using standard building blocks (SBB) consist of the core of DES. The standard building blocks use 48 bits of key to transform the 64 input data bits into 64 output data bits, designated as 32 left-half bits and 32 right-half

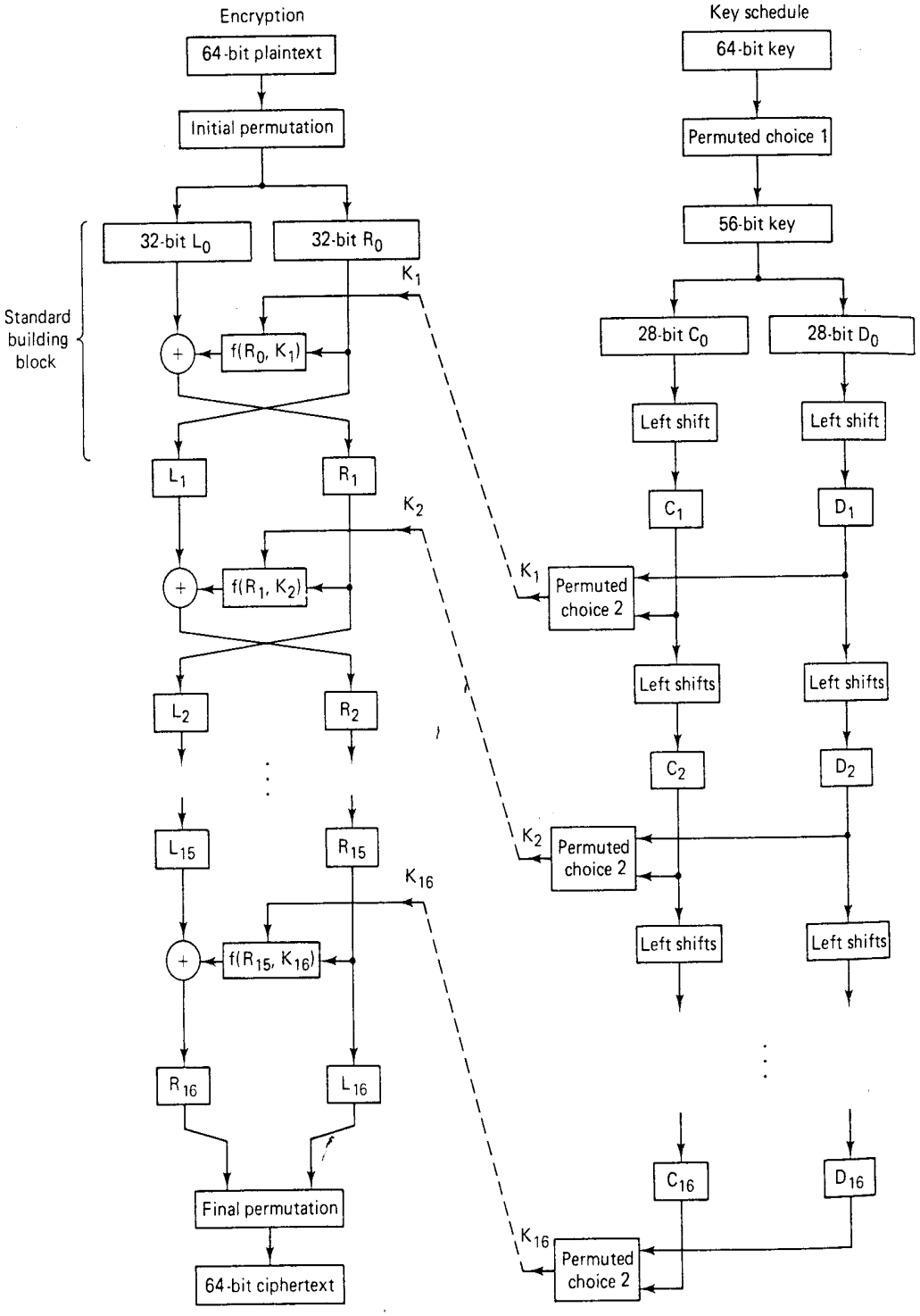

Figure 2.1: Data Encryption Standard (referenced from [17])

bits. The output of each building block becomes the input to the next building block.

Key selection also proceeds in 16 iterations, as seen in the key schedule portion of Figure 2.1. The input key consists of a 64-bit block with 8 parity bits. The permuted choice (PC-1) block discards the parity bits and permutes the remaining 56 bits. The output of PC-1 is split into two halves of 28 bits each. Key selection iterates 16 times to provide a different set of 48 key bits to each SBB encryption iteration.

2.2 Low-power VLSI circuits

The increasing prominence of portable systems and the need to limit power consumption (and hence, heat dissipation) in VLSI chips have led to rapid and innovative developments in low-power design during recent years. The driving forces behind these developments are portable applications requiring low power dissipation and high throughput, such as notebook computers, portable communication devices and personal digital assistants (PDAs) [5]. The low-power design of digital integrated circuits has emerged as a very active and rapidly developing field of CMOS design.

2.2.1 Overview of power consumption

The average power consumption in conventional CMOS digital circuits can be expressed as the sum of three main components, namely, the dynamic (switching) power consumption, the short-circuit power consumption, and the leakage power consumption.

2.2.1.1 Dynamic power dissipation

The dynamic dissipation represents the power dissipated during a switching event, i.e., when the output node voltage of a CMOS logic gate makes a power-consuming transition. In digital CMOS circuits, dynamic power is dissipated when energy is drawn from the power supply to charge up the output node capacitance. During the charge-up phase, the output

node voltage typically makes a full transition from 0 to VDD, and the energy used for the transition is relatively independent of the function performed by the circuit.

2.2.1.2 Short-Circuit power dissipation

If a CMOS logic gate is driven with input voltage waveforms with finite rise and fall times, both the nMOS and the pMOS transistors in the circuit may conduct simultaneously for a short amount of time during switching, forming a direct current path between the power supply and the ground. The dissipation due to this short-circuit is called short-circuit power dissipation. The total amount of energy dissipated in the short-circuit current is a function of the on-time of the transistors and the operation modes of the devices [16].

2.2.1.3 Leakage power dissipation

The nMOS and pMOS transistors used in a CMOS logic gate generally have nonzero reverse leakage and subthreshold currents. In a CMOS VLSI chip containing thousands of transistors, these currents can contribute to the overall power dissipation even when the transistors are not undergoing any switching event. The magnitude of the leakage currents is determined mainly by the processing parameters and the leakage power dissipation is the product of the device leakage current and the supply voltage [15].

2.2.2 Power optimization

In order to reduce the power consumption, designers must use power-saving techniques at every stage of IC design including register transfer-level (RTL) and gate-level implementation [1]. RTL power optimization techniques, normally including clock gating and sleep mode techniques, deliver the biggest return in reducing the power consumption. Gate-level power optimization offers capabilities such as pin-swapping, downsizing, buffer removal, gate merging, slew optimization, and power speed-up based restructuring [12].

Synopsys's Power Compiler automatically minimizes power consumption at the RTL and gate level. At the RTL level, during the design elaboration phase, Power Compiler performs automatic clock gating to reduce the power consumption. At the gate level, based on the user's timing, power and area constraints, Power Compiler measures trade-offs between positive timing slacks, area and power and then delivers the lowest power consuming design that meets timing constraints. It usually delivers an average of 10 to 20 percent reduction in power during gate-level optimization [10].

2.3 CAD tools

The most important CAD tools used in this work are Mentor Graphics's Modelsim SE and Leonardo Spectrum, Synopsys's FPGA Compiler II and Design Compiler, and Cadence's Silicon Ensemble and Virtuoso. The following sections give a brief description of these tools.

2.3.1 Modelsim SE

Modelsim SE ® is Mentor Graphics's Unix, Linux, and Windows-based simulator. It utilizes the Single Kernel Simulator (SKS) technology to enable VHDL, Verilog and mixed-language simulation. Its other major features include high-performance RTL and gate-level optimizations, Performance Analyzer for accelerating simulations and Waveform Compare advanced debugging feature [9].

2.3.2 Leonardo Spectrum

Mentor Graphics's Leonardo Spectrum ® provides a synthesis environment in which PLDs, FPGAs, or ASICs can be created in VHDL or Verilog [8]. It can mix VHDL, Verilog and DEIF to enable design reuse and instantiation of intellectual property.

2.3.3 Design Compiler

Design Compiler ® is the core of the Synopsys synthesis software products including Module Compiler, Power Compiler, Design Ware, Prime Time, and DFT Compiler [3]. It provides constraint-driven sequential optimization and supports a wide range of design styles. Its key features include critical path synthesis, register retiming, and basic timing closure capabilities.

2.3.4 FPGA Compiler II

Synopsys's FPGA Compiler II ® includes architecture-specific synthesis capabilities with features such as automatic register retiming and pipelining, Block-Level Incremental Synthesis (BLIS), DesignWare library support, Design Compiler shell script input/output and db file support. FPGA Compiler II offers the migration path between FPGA and ASIC technologies because FPGA Compiler II uses the same VHDL and Verilog synthesis subset as Synopsys' Design Compiler ® [13].

2.3.5 Silicon Ensemble

Silicon Ensemble ® is the cornerstone of the Cadence unified Synthesis/Place-and-Route (SP&R) solution. In addition to its conventional place-and-route features, Silicon Ensemble also includes Physically Knowledgeable Synthesis (PKS) concurrent optimization and signal integrity prevention, detection, and correction capabilities. These features make Silicon Ensemble ® a popular tool for physical implementation of integrated circuits [2].

2.3.6 Virtuoso Layout Editor

Cadence's Virtuoso ® Layout Editor supports both digital and analog custom layout editing [14]. It provides a streamlined command set which incorporates editing and layout techniques to support lots of methodologies and process technologies.

Chapter 3

Implementation with FPGA

The DES design is prototyped to FPGA first. Figure 3.1 shows the basic implementation process. The Register Transfer Level (RTL) design is simulated first to verify the syntax and basic functionality (without timing information). Then the synthesis process implements and optimizes the RTL design into equivalent unit-delay (no timing information) primitive blocks (flip-flops, logic gates, etc.). The generated FPGA-independent netlist from synthesis can be simulated with functional simulation, which contains no timing information, just verifying that the design finished the synthesis process with the same functionality as at the RTL level. In the Place & Route step, the blocks of the FPGA-independent netlist are converted into their FPGA device-specific elements. After that, both the element delay and routing delay are known. The postlayout simulation can be applied with the delay information.

3.1 RTL level simulation

In this design, Mentor Graphics's ModelSim is chosen as the simulation tool. The testbench mentioned previously is used throughout the whole flow to verify the functionality of the design at each level (RTL, Functional Gate, Timing Gate). It provides the stimulus and response information that the design encounters. A sequence of keys and plaintexts are

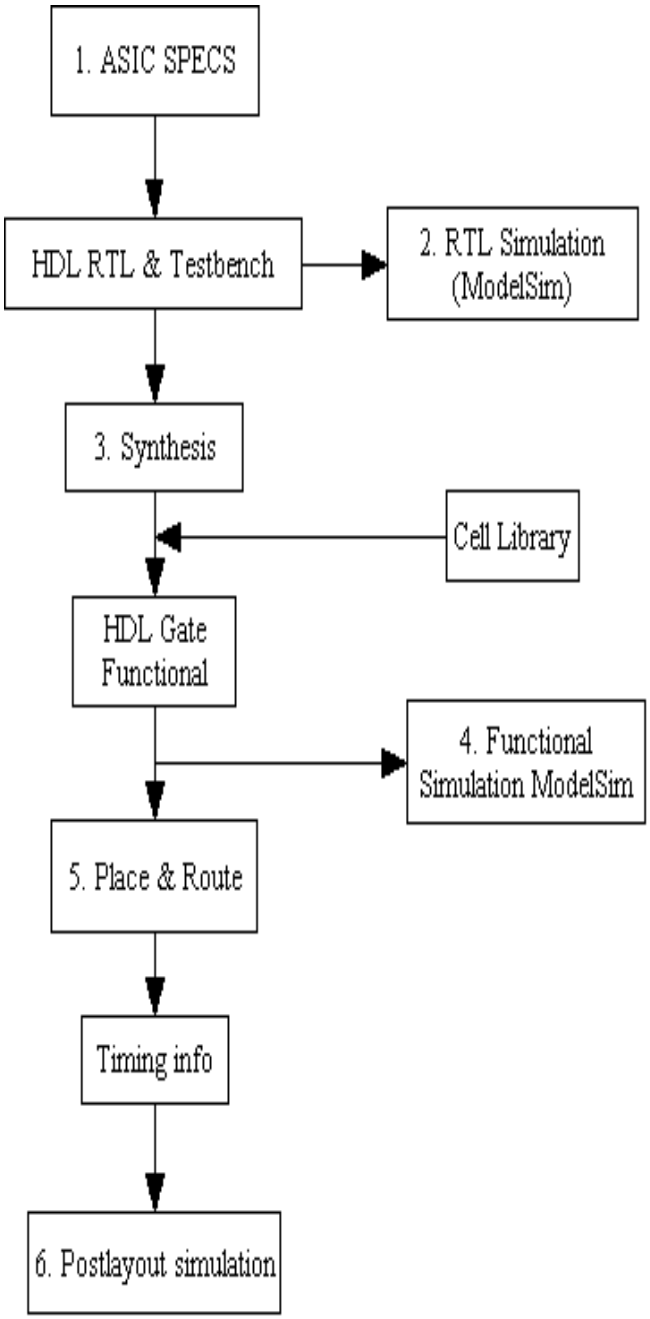

Figure 3.1: FPGA flow

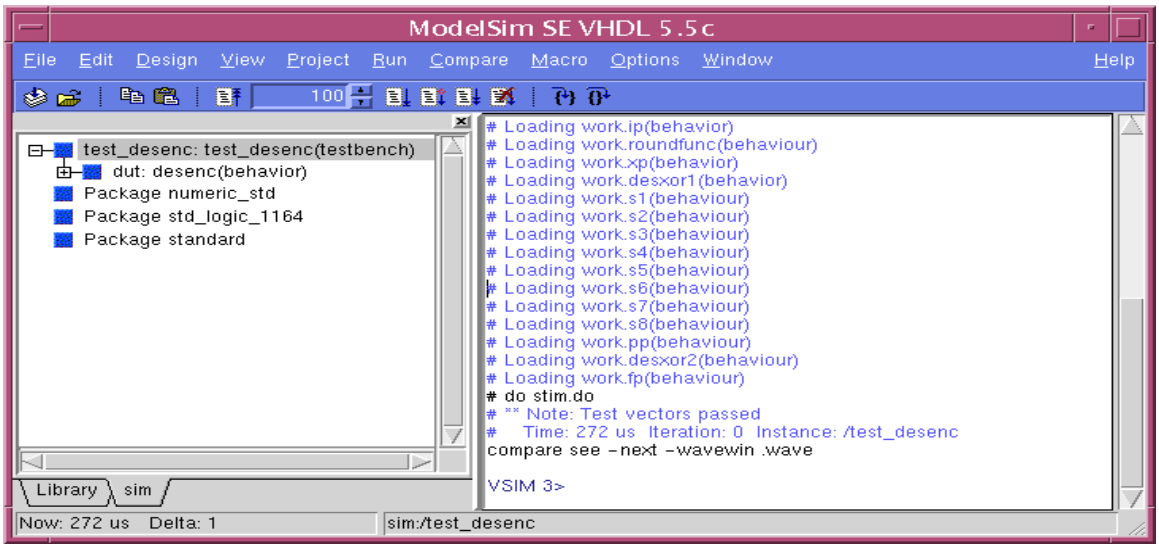

Figure 3.2: The main window of ModelSim for prelayout simulation of DES

provided to the DES design, along with a sequence of expected ciphertext. After each 16 clock cycles, the ciphertext generated by the DES design is compared with the expected ciphertext. If they are the same, the simulation passes. Otherwise, the simulation is not correct and an error message is reported. Figure 3.2 and figure 3.3 show the result of the successful RTL level simulation of the DES design.

From the ModelSim main window, it can be seen that "Test vectors passed." The wave window shows that when the key is "fedcba9876543210" and the plaintext is "ffffffffffffff", the ciphertext is "2a2bb008df97c2f2," which corresponds with the expected value from the testbench.

3.2 Synthesis with Leonardo Spectrum

First, Mentor Graphics's Leonardo Spectrum is used to synthesize the DES design. Some compile constraints are applied. The output file is an EDIF (Electronic Design Interchange Format) file, which is a FPGA-independent netlist file.

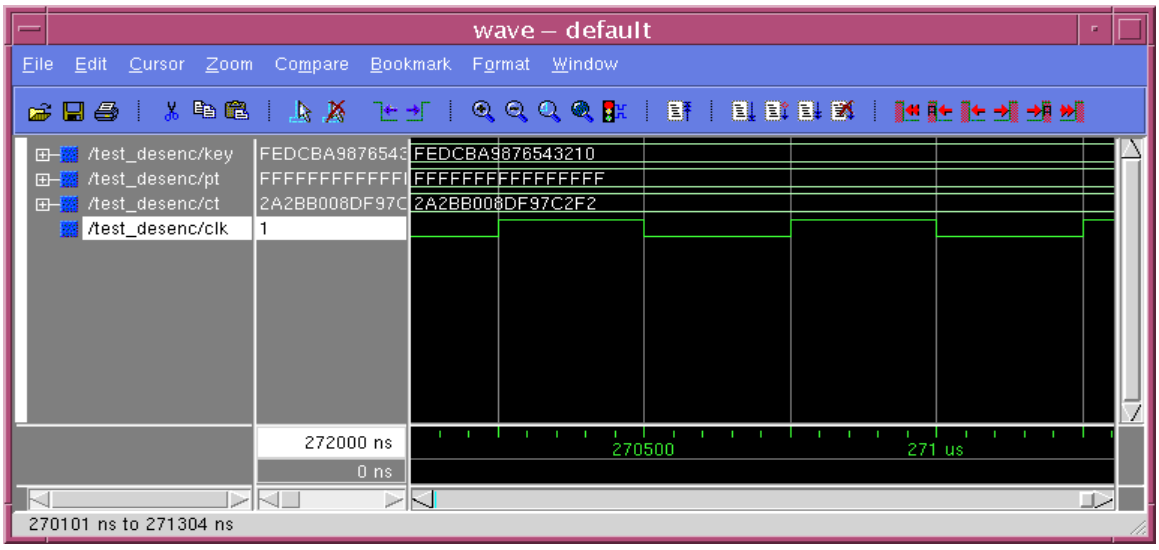

Figure 3.3: The wave window of ModelSim for prelayout simulation of DES

The targeted device is the Virtex V1000ebg560. Some other Virtex devices may also have enough slices to contain the DES design, but some of them do not have enough IOB and pins for the design (DES requires 185 pins). The Xilinx Place & Route (P&R) tools take the EDIF file and implement it in the targeted Virtex V1000ebg560 device. Finally, the Place & Route tools export a ".ncd" file, which is a device configuration file used to program the targeted FPGA device.

There are two important files generated by the P&R tools for gate-level simulation. One is "time_sim.vhd," which is a VHDL netlist file. This netlist is made up of gate-level primitive blocks that are specific to a particular FPGA device. The functionality of the blocks is represented by FPGA-vendor supplied libraries that contain the definition and characteristics of each primitive block for each FPGA device. Another file is "time_sim.sdf" which contains the delay information.

The ".mrp" file contains the design information. By checking that, it is found that 1,511 out of 12,288 available slices are used. The ".twr" file contains the timing information for this implementation. The minimum period is 20.063ns and therefore the maximum

Table 3.1: Leonardo Spectrum and FPGA Compiler II synthesis results

	Leonardo Spectrum	FPGA Compiler II
Slices used	1511	1538
Maximum frequency	49.843MHz	52.200MHz

frequency is 49.843MHz.

Figure 3.4 shows the layout of the DES design in the Virtex V1000ebg560.

The post-layout simulations with ModelSim are shown in Figure 3.5 and Figure 3.6. They look similar to the pre-layout simulation. However, by checking the zoom-in view of the wave window of the postlayout simulation, it can be found that the delay information obtained from the P&R process is included in the postlayout simulation. For example, in a zoom-in view shown in Figure 3.7, the "ct" signal changes with some ns delay after the clock switch.

3.3 Synthesis with FC2

Synopsys's FPGA Compiler II (FC2) is another synthesis tool used to compile the DES design to the Virtex V1000ebg560 device. After synthesis, placement and routing, the post-layout simulation is made again with the wave window shown in Figure 3.8.

The Figure 3.9 shows the layout of the design, which is, of course, different from that from Leonardo Spectrum synthesis.

The ".mrp" file and ".twr" file report that 1,538 out of 12,288 slices are used and the minimum period is 19.157ns, which determines the maximum frequency 52.200MHz.

Table 3.1 lists the comparison of the synthesis results between Leonardo Spectrum and FC2. For this DES case, the FC2 implementation takes a little bit more slices, but it is also a little bit faster than the Leonardo Spectrum implementation.

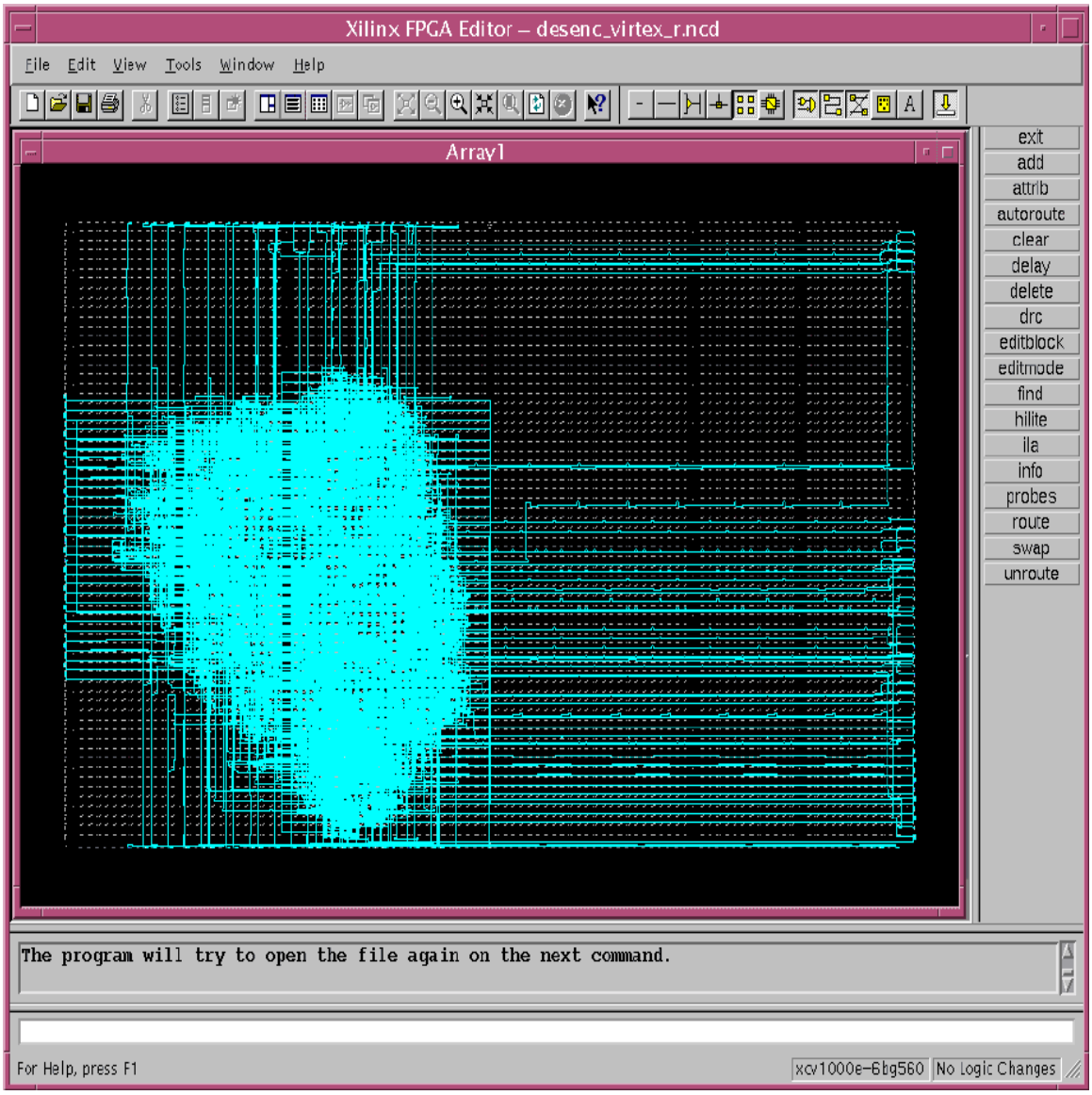

Figure 3.4: The layout of DES with Virtex V1000ebg560

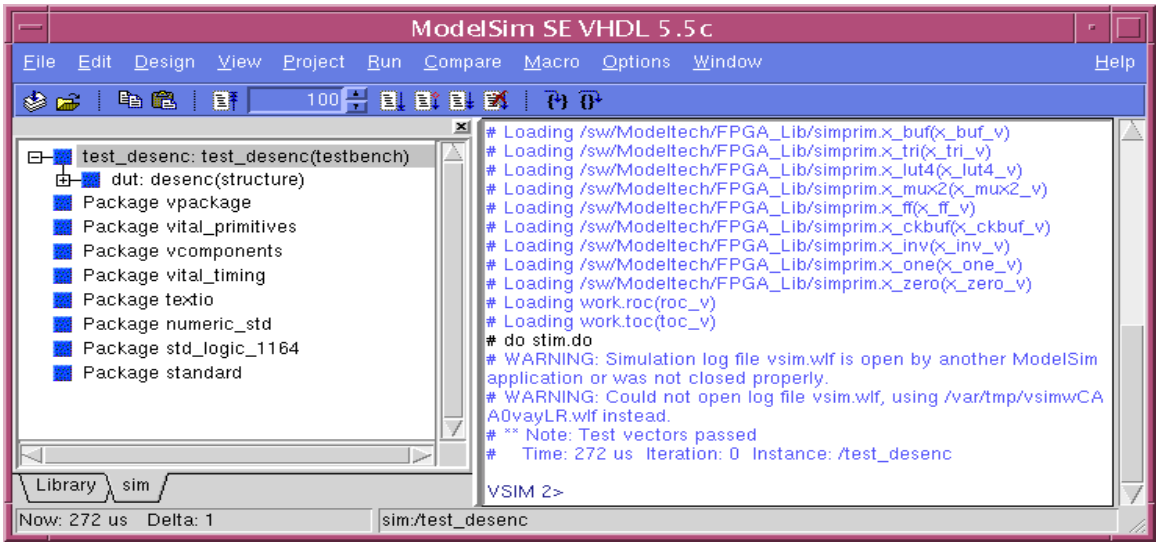

Figure 3.5: The main window of ModelSim for postlayout simulation of DES from Leonardo Spectrum synthesis

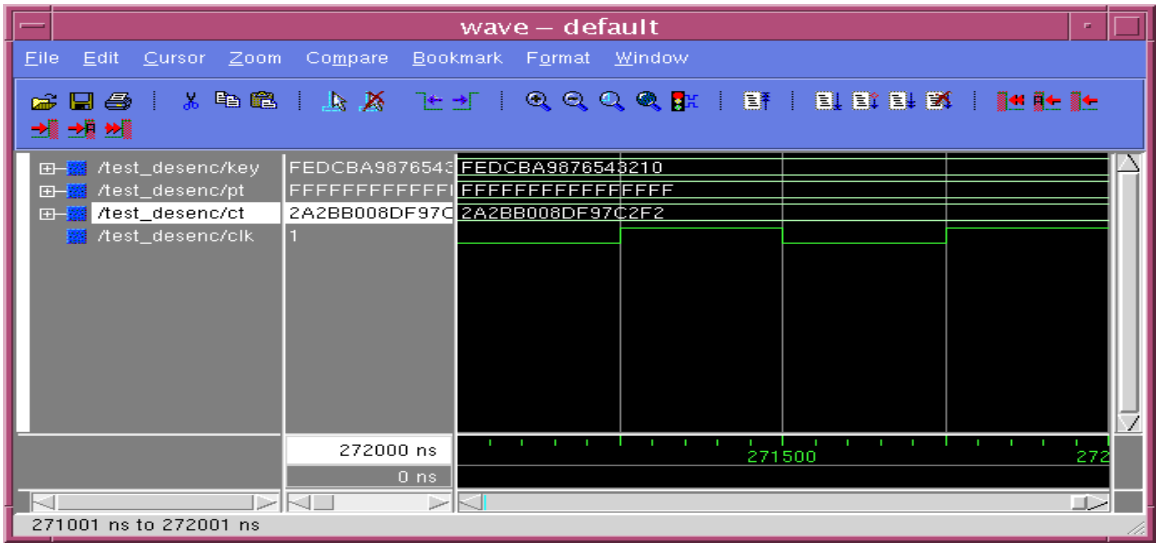

Figure 3.6: The wave window of ModelSim for postlayout simulation of DES from Leonardo Spectrum synthesis

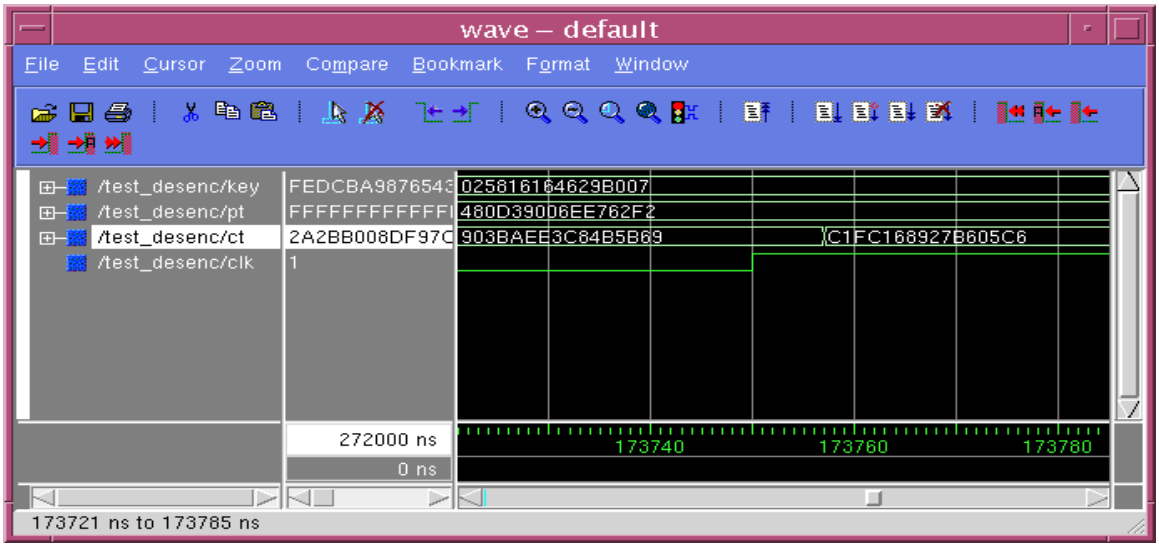

Figure 3.7: The zoom-in view of wave window for postlayout simulation from Leonardo Spectrum synthesis

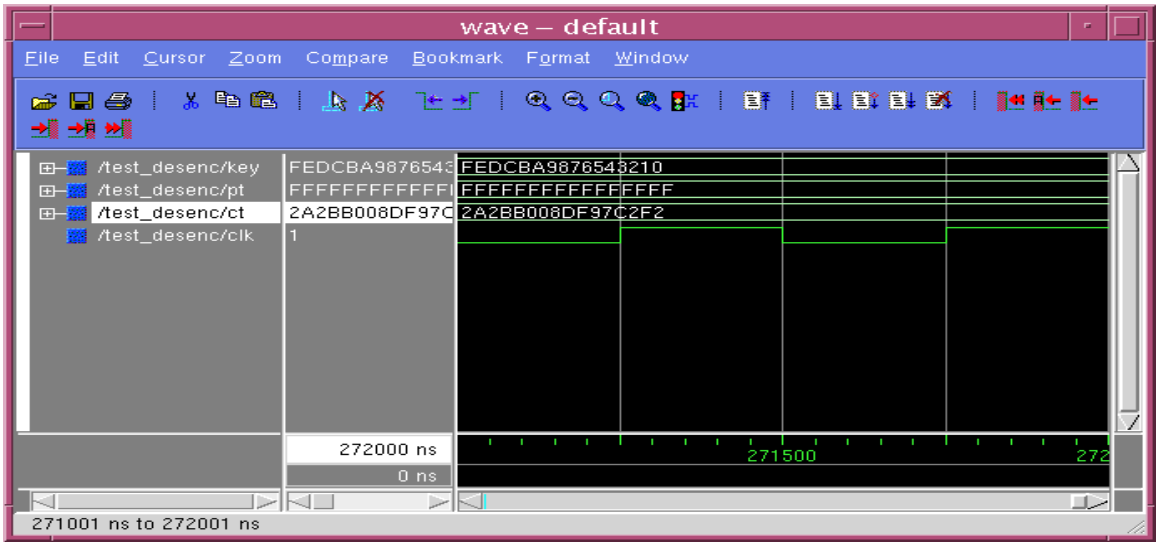

Figure 3.8: The wave window for postlayout simulation from FC2 synthesis

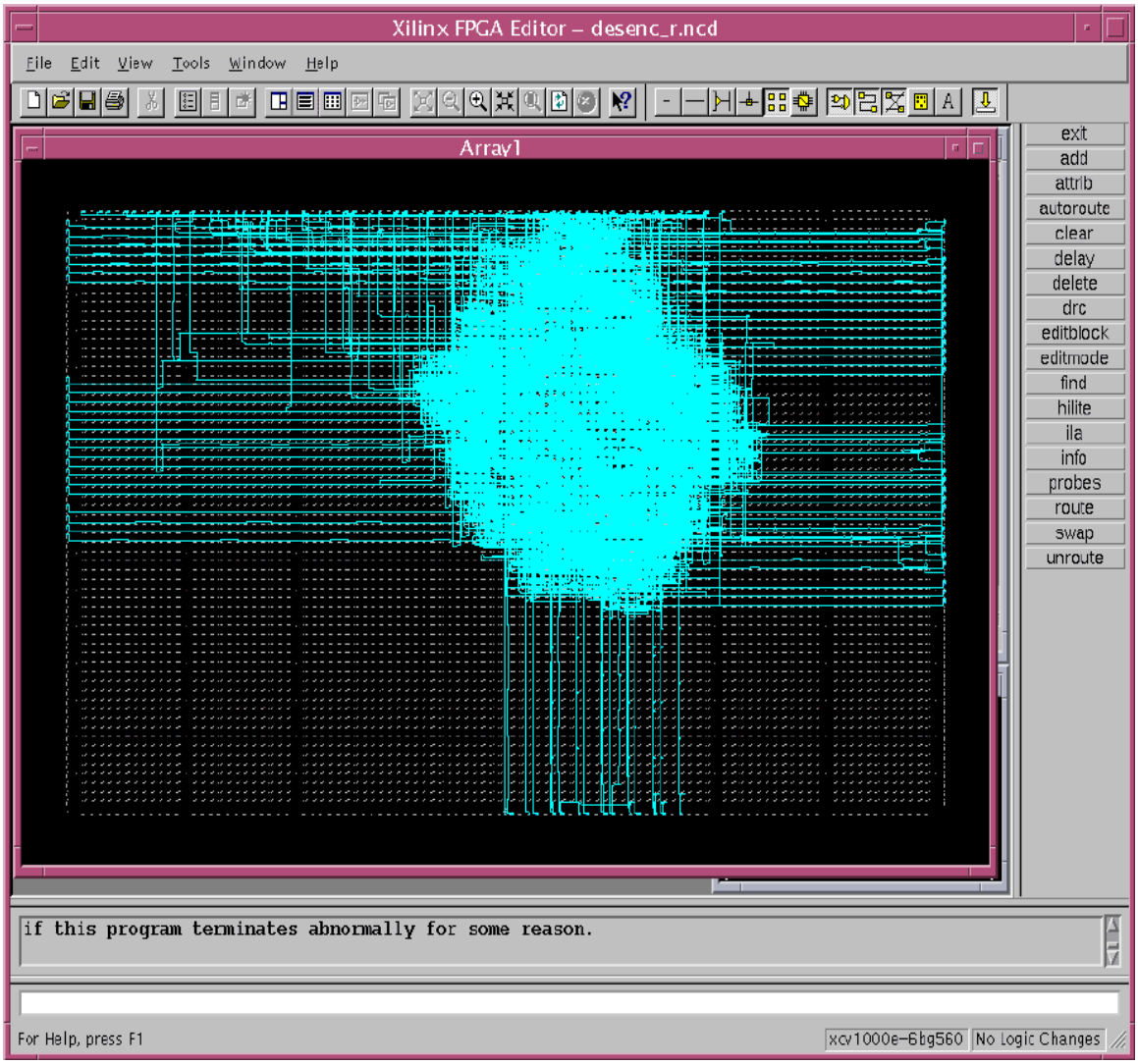

Figure 3.9: The layout of DES with Virtex V1000ebg560 from FC2 synthesis

Chapter 4

Implementation with Testable ASIC

The DES design is also synthesized to Standard-Cell-Based ASIC with TSMC18 process. Synopsys Design Compiler is used as the tool to explore the design space to compile the DES design to different versions that are respectively optimized for area, speed and power. The generated netlists are saved as structural Verilog files. Cadence's Silicon Ensemble takes the Verilog files as input and does the automatic placement and routing. Finally, Cadence's Virtuoso imports DEF files from Silicon Ensemble and generates the physical layout.

4.1 Explore design space using Synopsys's Design Compiler

Design Compiler is the core of the Synopsys synthesis software products. It consists of tools that synthesize the HDL designs into optimized technology-dependent, gate-level designs. As shown in Figure 4.1, various formats of RTL designs such as VHDL and Verilog are compiled to technology-dependent netlists by Design Compiler. The technology-dependent netlists can then be used as input for place and route tools later.

Design Compiler provides two user interfaces: the Design Compiler command-line interface, or shell, referred to as dc–shell, and the Design Compiler graphical user interfaces (GUI) referred to as Design Analyzer and Design Vision. In this thesis, dc–shell is mostly

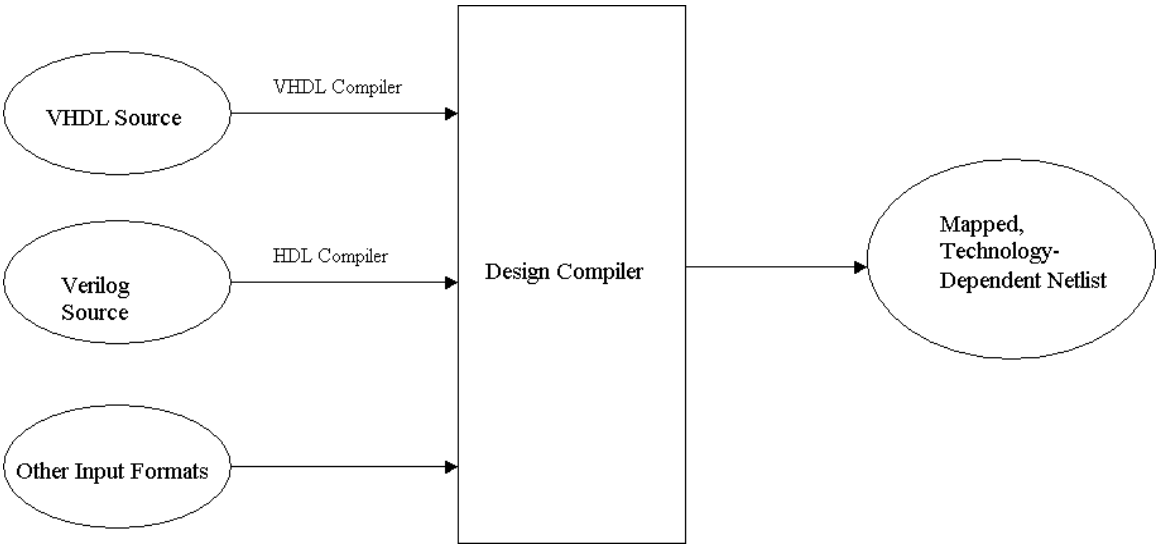

Figure 4.1: Design Compiler synthesis process overview

used.

With different constraints, the same design can be compiled to different implementations. For example, by setting speed, size or power constraints, the designer can get faster or smaller or less power-consuming versions. Some implementations are smaller, but consume more power. Some implementations consume less power, but have longer delay and therefore are slower. The designer can choose the implementation which satisfies the design requirement most. This is called exploring the design space. The three important design properties are area, speed and power. Figure 4.2 displays a design space curve.

The shape of the curve demonstrates the trade-off between area-efficient and speed-efficient circuits. Usually, the power is proportional to the area, so the actual trade-off is between speed and power. In this thesis, three implementations are obtained with different design constraints. The first one uses the library default constraint, i.e, it doesn't have any additional time or power constraint. This implementation turns out to be the smallest one of the three. The second version achieves higher speed by setting time constraints on critical paths. However, it is also the one that consumes the most power of the three

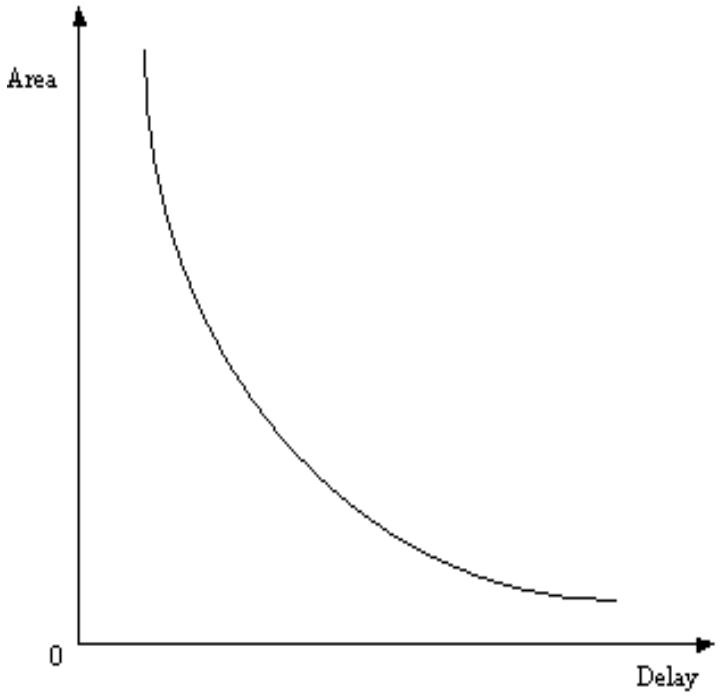

Figure 4.2: Design space curve

implementations. The last implementation consumes the least power, but is slower than the other two.

4.1.1 Optimize with library default constraints

To use dc–shell, the ".synopsys–dc.setup" setup file is created in the working directory. It contains the definitions of the Design Compiler variables, which determine the Design Compiler working environment. The following is the content of the ".synopsys–dc.setup" file:

```
search_path = {.}
link_library = {typical.db};
target_library = {typical.db};
symbol_library = {typical.db};
define_design_lib WORK -path WORK;
```

The file "typical.db," which is the library file for this project, is also copied to the working directory. Then the design is compiled without any additional constraints set on it. After invoking dc–shell, the following commands are used to compile the design:

```
dc_shell> analyze -format vhdl -lib WORK ./desenc.vhd
dc_shell> elaborate desenc
dc_shell> uniquify
dc_shell> compile
dc_shell> write -format db -hierarchy -output "./desenc.db"
```

The analyze command reads the VHDL file, checks for proper syntax and synthesizable logic, and stores the design in an intermediate format. The elaborate command creates the GTECH design, which is a technology independent design, from the intermediate format

produced by the analyze process. The elaborate command replaces the HDL operators in the design with synthetic operators and determines the correct bus sizes.

DES is a hierarchical design in which some subdesigns such as s1 and s2 are referenced more than once. In order to resolve multiple instances in DES, the uniquify command is used. The uniquify command creates one copy of the subdesign each time it is referenced in the design and assigns a unique design name to the copy. During compilation, the Design Compiler optimization maps each instance to its unique environment.

Optimization is the step in the synthesis process that attempts to implement a combination of library cells that meets the functional, area and speed requirements of the design. The compile command invokes optimization. The compile process modifies and optimizes the design as it attempts to create a circuit that meets the specified constraints.

Finally, Design Compiler does not save the design automatically. It is necessary to use the write command to save design. The above write command saves the design in "db" format.

The report–area command is used to obtain the area of the design as follows:

```
****************************************
Report : area
Design : desenc
Version: 2000.11
Date   : Wed Mar 13 16:39:35 2002
****************************************

Library(s) Used:

    typical (File: /home/xfu/thesis/tsmc18/opt_nothg/typical.db)

Number of ports:              193
Number of nets:              2049
Number of cells:               19
Number of references:          19
```

```
Combinational area:       183351.171875
Noncombinational area:    28952.992188
Net Interconnect area:     187.700317

Total cell area:          212304.156250
Total area:               212491.859375
```

It is seen that the area of this implementation is estimated to be 212491 micron square. The speed and power of this implementation will be discussed later.

4.1.2 Optimize for speed

In sequential circuits, the clock period must be greater than the longest path delay of the combinational logic between latch outputs and inputs. So the key to optimize the speed of the design is to locate the flip-flops and the combinational logic between them. If the longest path in the combinational logic can be shortened, the required minimum clock period will be decreased due to the decreased longest path delay. In other words, the clock frequency can be increased, which means faster circuits.

Figure 2.1 is revisited and studied here. The right half generates key signals K1 to K16 to provide for the standard building blocks. The key-generation part is purely combinational without any flip-flops. The encryption part on the left consists of 16 rounds of standard building blocks. Figure 4.3 shows the structure of a standard building block.

In Figure 4.3, XP and PP boxes only contain a signal order switch. S boxes contain combinational mapping logic followed by registers. Therefore S boxes are where most path delays exist. If the path delay of the combinational part of S boxes can be decreased, the speed of the circuit will be increased.

In order to decrease the path delay of S boxes, s1.vhd to s8.vhd, which are the vhdl codes to describe the functionality of S boxes, are extracted from the original desenc.vhd and are

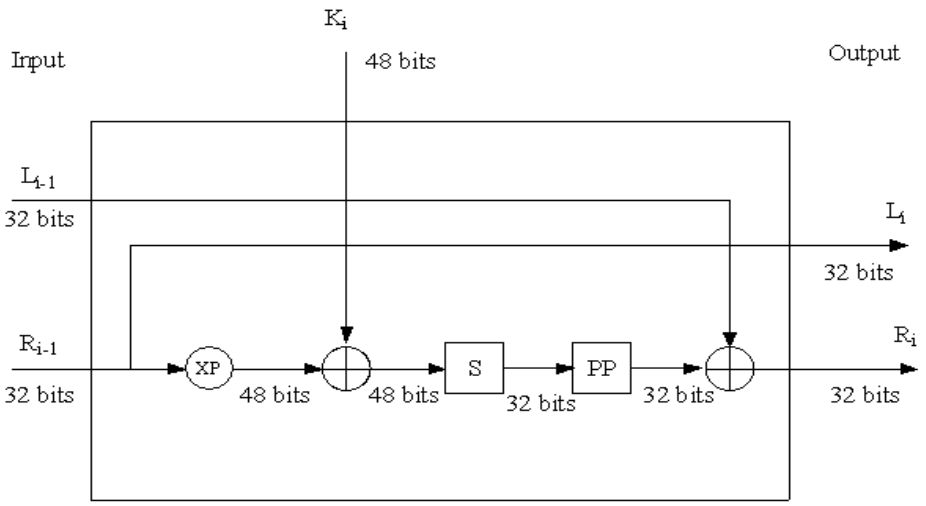

Figure 4.3: Standard building block (SBB)

compiled separately with tight time constraints. For example, after s1.vhd is compiled for the first time, the following dc_shell command is executed to apply constraint:

```
dc_shell> set_max_delay 1.2 -from {b[1] b[2] b[3] b[4] b[5] b[6]} -to {so_reg[1]/D so_reg[2]/D
so_reg[3]/D so_reg[4]/D}
```

In the above command, b[1] to b[6] are the input ports of S boxes. So_reg[1]/D to so_reg[4]/D are the D ports of the flip-flops that follow the combinational logic. They can also be viewed as the "output" of the combinational logic. The above command sets the maximum path delay from any b port to any D port of flip-flops to be 1.2 ns. Then s1.vhd is compiled again with the constraint applied. The report_timing command is executed to find the path delay as following:

```
dc_shell> report_timing
```

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : s1
Version: 2000.11
Date   : Fri Mar 15 16:48:30 2002
****************************************

Operating Conditions: typical   Library: typical
Wire Load Model Mode: segmented

  Startpoint: b[1] (input port)
  Endpoint: so_reg[2] (rising edge-triggered flip-flop)
  Path Group: default
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  s1                 TSMC18_Conservative   typical

  Point                                    Incr       Path
  -----------------------------------------------------------
  input external delay                     0.00       0.00 r
  b[1] (in)                                0.00       0.00 r
  U189/Y (XOR2X4)                          0.12       0.12 f
  U163/Y (AND3X2)                          0.16       0.28 f
  U153/Y (NAND2X2)                         0.09       0.37 r
  U197/Y (AOI31X2)                         0.05       0.42 f
  U170/Y (OR4X2)                           0.20       0.62 f
  U220/Y (OR4X2)                           0.24       0.86 f
  U201/Y (OR4X2)                           0.24       1.09 f
  so_reg[2]/D (DFFXL)                      0.00       1.09 f
  data arrival time                                   1.09

  max_delay                                1.20       1.20
  library setup time                      -0.11       1.09
```

```
data required time                                  1.09
-----------------------------------------------------------
data required time                                  1.09
data arrival time                                  -1.09
-----------------------------------------------------------
slack (MET)                                        0.00
```

It can be seen from above report that the longest path of the combinational part, which starts from b[1] and ends at so_reg[2]/D, satisfies the maximum delay constraint. After all the S boxes are compiled with the maximum delay constraint in the same way with s1, the "set_dont_touch" command is applied on them so that these subdesigns will remain a fixed internal structure and will not be affected by later compilation. For example,

```
dc_shell> set_dont_touch s2
```

sets the constraint on s2. Then the top level desenc.vhd is compiled. Because of the "set_dont_touch" command, the s boxes are not changed during the optimization of the whole design. This method is called the compile-once-don't-touch method.

After all the design is compiled, the report_timing command is used to find the path delay of the combinational logic between the output of a round of flip-flop outputs and the input of the next round of flip-flops. The following command

```
dc_shell>  report_timing -from {round10/us1/so_reg[1]/Q round10/us1/so_reg[2]/Q round10/us1/so_reg[3]/Q
round10/us1/so_reg[4]/Q }  -to  {round11/us3/so_reg[4]/D round11/us3/so_reg[3]/D round11/us3/so_reg[2]/D
round11/us3/so_reg[1]/D }
```

reports the longest path delay from any Q port of the four flip-flops of us1 of round10 to any D port of the four flip-flops of us3 of round11. The following report is obtained:

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : desenc
Version: 2000.11
Date   : Sun Mar 17 14:35:19 2002
****************************************

Operating Conditions: typical   Library: typical
Wire Load Model Mode: segmented

  Startpoint: round10/us1/so_reg[1]
              (rising edge-triggered flip-flop)
  Endpoint: round11/us3/so_reg[2]
            (rising edge-triggered flip-flop)
  Path Group: (none)
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  s7                 TSMC18_Conservative   typical
  s8                 TSMC18_Conservative   typical
  s6                 TSMC18_Conservative   typical
  roundfunc_2        TSMC18_Conservative   typical
  roundfunc_5        TSMC18_Conservative   typical
  roundfunc_10       TSMC18_Conservative   typical
  xp_8               TSMC18_Conservative   typical
  s1                 TSMC18_Conservative   typical
  roundfunc_3        TSMC18_Conservative   typical
  desxor1_8          TSMC18_Conservative   typical
  roundfunc_11       TSMC18_Conservative   typical
  desenc             TSMC18_Conservative   typical
  s5                 TSMC18_Conservative   typical
  roundfunc_4        TSMC18_Conservative   typical
  s4                 TSMC18_Conservative   typical
```

```
s3                TSMC18_Conservative   typical
s2                TSMC18_Conservative   typical
roundfunc_1       TSMC18_Conservative   typical
roundfunc_6       TSMC18_Conservative   typical
roundfunc_13      TSMC18_Conservative   typical
roundfunc_8       TSMC18_Conservative   typical
roundfunc_14      TSMC18_Conservative   typical
desxor2_13        TSMC18_Conservative   typical
roundfunc_0       TSMC18_Conservative   typical
roundfunc_9       TSMC18_Conservative   typical
roundfunc_12      TSMC18_Conservative   typical
roundfunc_15      TSMC18_Conservative   typical
pp_13             TSMC18_Conservative   typical
roundfunc_7       TSMC18_Conservative   typical
desxor2_0         TSMC18_Conservative   typical

Point                                    Incr       Path
-----------------------------------------------------------
round10/us1/so_reg[1]/CK (DFFX1)         0.00       0.00 r
round10/us1/so_reg[1]/Q (DFFX1) <-       0.30       0.30 r
round10/us1/so[1] (s1)                   0.00       0.30 r
round10/upp/so1x[1] (pp_13)              0.00       0.30 r
round10/upp/ppo[9] (pp_13)               0.00       0.30 r
round10/udesxor2/d[9] (desxor2_13)       0.00       0.30 r
round10/udesxor2/U7/Y (XOR2X1)           0.47       0.77 r
round10/udesxor2/q[9] (desxor2_13)       0.00       0.77 r
round10/ro[9] (roundfunc_13)             0.00       0.77 r
round11/ri[9] (roundfunc_8)              0.00       0.77 r
round11/uxp/ri[9] (xp_8)                 0.00       0.77 r
round11/uxp/e[14] (xp_8)                 0.00       0.77 r
round11/udesxor1/e[14] (desxor1_8)       0.00       0.77 r
round11/udesxor1/U41/Y (XOR2X1)          0.87       1.64 r
round11/udesxor1/b3x[2] (desxor1_8)      0.00       1.64 r
round11/us3/b[2] (s3)                    0.00       1.64 r
round11/us3/U229/Y (CLKINVX3)            0.22       1.86 f
round11/us3/U171/Y (NAND2X2)             0.24       2.09 r
round11/us3/U153/Y (AOI211X1)            0.09       2.18 f
round11/us3/U152/Y (OR4X2)               0.26       2.44 f
```

```
round11/us3/U261/Y (OR4X2)              0.24       2.68 f
round11/us3/U238/Y (OR4X2)              0.24       2.91 f
round11/us3/so_reg[2]/D (DFFXL)         0.00       2.91 f
data arrival time                                  2.91
-----------------------------------------------------------
(Path is unconstrained)
```

The longest path is from the Q port of round10/us1/so_reg[1] to the D port of round11/us3/so_reg[2] with delay 2.91 ns. The DES design contains 16 rounds of standard building blocks, and therefore 16 rounds of flip-flop banks. By applying the report_timing commands repeatedly, the longest combinational path delay between rounds of flip-flop banks is found to be 3.45 ns. Therefore, the highest clock frequency of the circuit is 289855 Hz. The total area is found to be 331204 micron square with report_area command.

The implementation with default compile constraint discussed in the last section is also checked with report_timing command. The longest combinational path delay is 3.66 ns and therefore the allowed highest clock frequency is 273224 Hz. The default constraint version is slower and smaller than the speed-optimized version, which illustrates the tradeoff between speed and area.

4.1.3 Optimize for power

4.1.3.1 Power flow

As one creates a design, it moves from a high level of abstraction to its final implementation at the gate level. Synopsys's Power Compiler offers analysis and optimization throughout the design cycle, from RTL to the gate level. As shown in Figure 4.4, simulation, analysis and optimization are used at each level of abstraction to refine the design before moving to the next lower level of design abstraction. Simulation and the resultant switching activity give analysis and optimization the necessary information to refine the design.

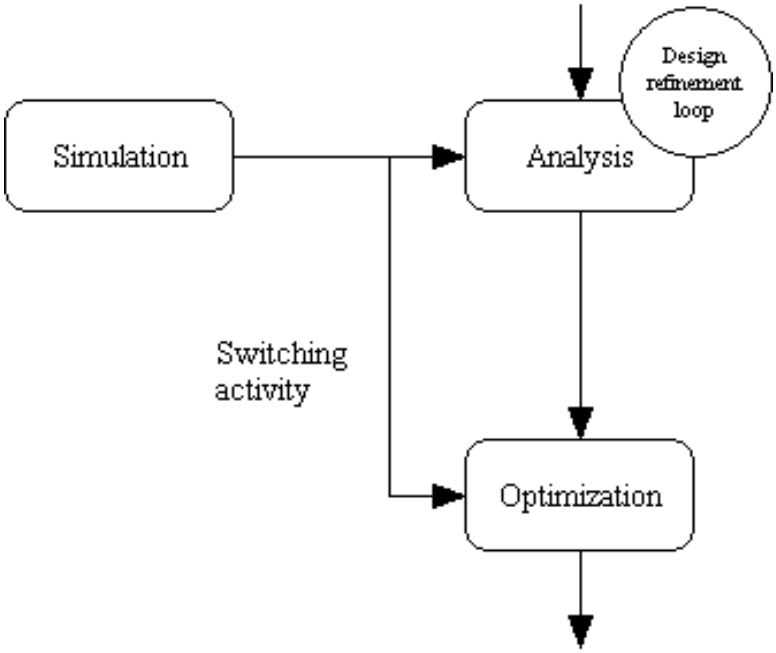

Figure 4.4: Power flow

4.1.3.2 Switching activity

Cell internal power and net toggling have a direct effect on the dynamic power of a circuit design. To report or optimize dynamic power, Power Compiler needs toggle information for the design. This information is called switching activity. A SAIF (switching activity interchange format) forward-annotation file directs simulation to monitor primary inputs and other synthesis-invariant elements and a SAIF back-annotation file contains the resultant switching activity of the elements monitored during RTL simulation. Synopsys power tools can read the information in the back-annotation file and annotate it on the compiled design. Below is part of the back-annotation SAIF file of DES.

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Thu Mar  7 18:35:41 2002")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VSS-SAIFAPI")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 272000.00)
(INSTANCE test_desenc
    (INSTANCE DUT
       (PORT
          (pt\[1\] (T1 24000)(T0 248000)(TX 0)(TC 5))
          (pt\[2\] (T1 80000)(T0 192000)(TX 0)(TC 15))
          (pt\[3\] (T1 64000)(T0 208000)(TX 0)(TC 11))
          (pt\[4\] (T1 104000)(T0 168000)(TX 0)(TC 15))
          (pt\[5\] (T1 64000)(T0 208000)(TX 0)(TC 15))
          (pt\[6\] (T1 88000)(T0 184000)(TX 0)(TC 15))
          (pt\[7\] (T1 112000)(T0 160000)(TX 0)(TC 13))
         (pt\[8\] (T1 152000)(T0 120000)(TX 0)(TC 21))
         (pt\[9\] (T1 88000)(T0 184000)(TX 0)(TC 13))
```

```
(pt\[10\] (T1 112000)(T0 160000)(TX 0)(TC 21))
(pt\[11\] (T1 112000)(T0 160000)(TX 0)(TC 15))
(pt\[12\] (T1 120000)(T0 152000)(TX 0)(TC 17))
(pt\[13\] (T1 104000)(T0 168000)(TX 0)(TC 17))
```

In a SAIF back-annotation file like the one above, T0 is the duration of the time found in the logic 0 state. T1 is the duration of time found in the logic 1 state. TC is the sum of the rise (0 to 1) and fall (1 to 0) transitions that are captured during monitoring.

4.1.3.3 Optimize design with RTL simulation and SAIF files

Figure 4.5 shows a general RTL methodology for capturing and annotating switching activity using a SAIF-compliant simulator.

When one analyzes and elaborates an RTL design, HDL Compiler creates a technology-independent design called GTECH design. Using information from the GTECH design, HDL Compiler creates the SAIF forward-annotation file. The SAIF back-annotation file is created from the SAIF forward-annotation file after RTL simulation. The switching activity in the back-annotation file is annotated onto a gate-level design (a technology-specific format) and power compiler can use that information to optimize the design to achieve the desired power goal.

4.1.3.4 Procedure

The key procedures of the optimization are briefly described here. First, analyze and elaborate the design with

```
dc_shell> analyze -f vhdl -lib WORK desenc.vhd
dc_shell> elaborate desenc
```

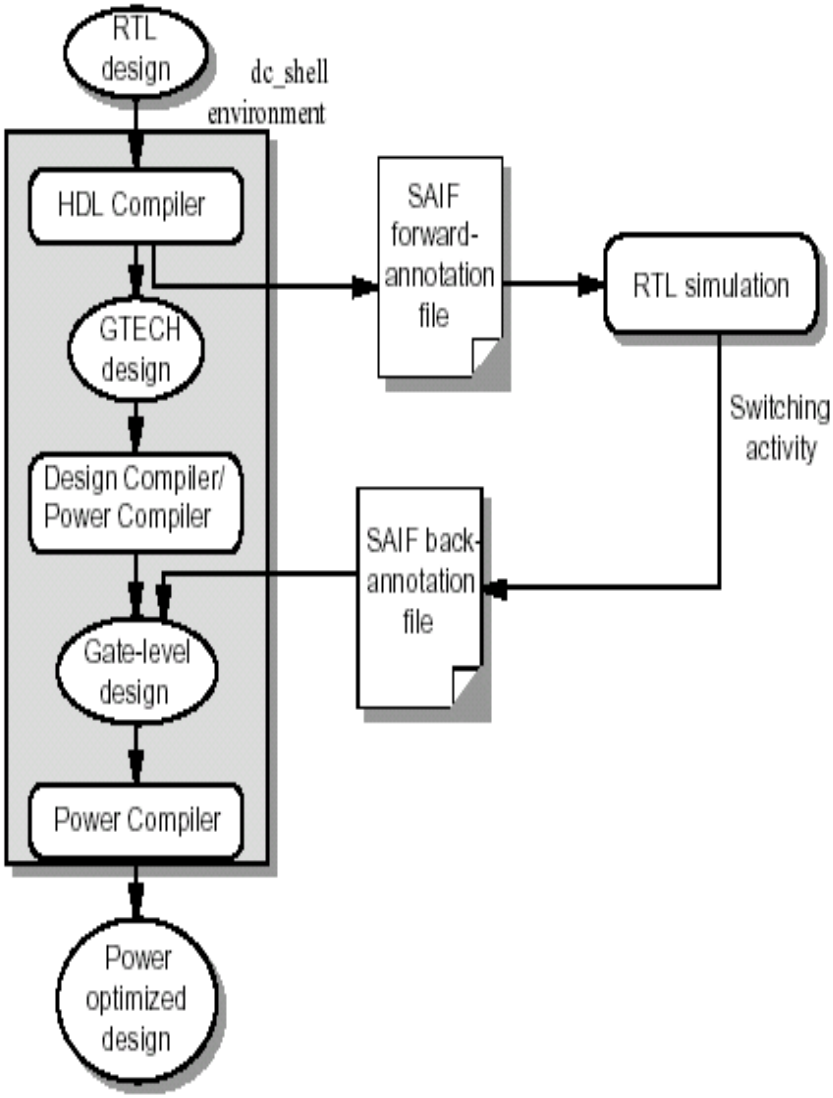

Figure 4.5: Methodology using RTL simulation and SAIF files (referenced from [11])

and resolve multiple instances with uniquify command and then compile the design as below:

```
dc_shell> uniquify
dc_shell> check_design
dc_shell> compile
```

Then set parameter as

```
dc_shell> power_preserve_rtl_hier_names = true
```

After that, use rtl2saif command to generate the SAIF forward-annotation file

```
dc_shell> rtl2saif -o desenc_fwd.saif
```

Then under Unix prompt, vhdlan command is used to analyze design file "desenc.vhd" and testbench "test_desenc.vhd" to generate configuration CFG_TEST_DESENC_BEHAVIORAL for VSS simulation. VSS simulator is invoked with vhdlsim command as following:

```
vhdlsim -nc -i capture_saif.vss CFG_TEST_DESENC_BEHAVIORAL &
```

"capture_saif.vss" is the VSS file with the following content:

```
monitor -SAIF -name mntr -hier test_desenc/DUT -input desenc_fwd.saif -output desenc_bwd.saif
run 272000
quit
```

The VSS monitor command creates, monitors, and reads the SAIF forward-annotation file and writes the SAIF back-annotation file which contains switching activity results from simulation. Now the switching activity can be annotated back to the design with the "read_saif" command. After setting power constraints, an incremental compile is performed on the design, trying to achieve the desired power goal. Some of the important commands during this process are listed below:

```
dc_shell> find_ignore_case = true
dc_shell> read_saif -input desenc_bwd.saif -instance test_desenc/DUT -unit ns
dc_shell> find_ignore_case = false
dc_shell> report_power
dc_shell> set_max_dynamic_power 200 uW
dc_shell> compile -incremental
dc_shell> report_power
dc_shell> write -format db desenc.db
```

In the above commands, the find_ignore_case command is required because VSS simulation creates the SAIF file with all the object names in uppercase. Report_power command is used to get an idea how much power the original compilation consumes so that the following power constraint can be chosen realistically. Only the dynamic power constraint is set because leakage power is much smaller than the dynamic power and can be ignored here.

4.1.3.5 Results

The power report for the power-optimized implementation is obtained as following:

```
Global Operating Voltage = 1.8
Power-specific unit information :
     Voltage Units = 1V
     Capacitance Units = 1.000000pf
     Time Units = 1ns
```

Table 4.1: Area, speed and power results

	Area (micron square)	Highest clock frequency (Hz)	Power (uW)
default-constraint version	212491	273224	490
speed-optimized version	331204	289855	753
power-optimized version	250078	169779	421

```
     Dynamic Power Units = 1mW    (derived from V,C,T units)
     Leakage Power Units = 1nW


   Cell Internal Power  = 190.3801 uW   (45%)
   Net Switching Power  = 229.5871 uW   (55%)
                          ---------
Total Dynamic Power    = 419.9673 uW  (100%)

Cell Leakage Power     =   1.2754 uW
```

The power reports of the default-constraint version and speed-optimized version are also generated by back-annotating the SAIF back-annotation file to the design using read_saif command and report_power command. It is necessary to point out that these power reports are based on the test time and test vectors of the testbench and therefore are not necessarily accurate for all the working scenarios.

4.1.4 Comparison

Some important parameters of the three different implementations discussed above are listed in Table 4.1. The tradeoff between area, speed and power is clearly illustrated in the above table. The power-optimized version consumes the least power, but is the slowest of the

three. The speed-optimized version is the fastest, but takes the biggest area. The default-constraint version can be viewed as "area-optimized"; its speed and power are between those of the speed-optimized and power-optimized implementations.

4.2 Implement with testable ASIC

4.2.1 Manufacturing testing

After chips are manufactured, there might be manufacturing defects including problems such as power or ground shorts, open interconnect on the die caused by dust particles and short-circuited source or drain on a transistor caused by metal spike-through [6]. Manufacturing defects might remain undetected by functional testing yet cause undesirable behavior during circuit operation. Manufacturing testing verifies that circuits do not have manufacturing defects by focusing on circuit structure rather than functional behavior and therefore can screen devices for manufacturing defects.

4.2.2 Design-for-Test flow

Synopsys's DFT (Design-for-Test) compiler, along with other Synopsys tools, can be used to create a testable design with design-for-test methods. Scan synthesis, which includes scan replacement and scan assembly, automates the process of inserting scan structures into designs. The scan replacement process inserts scan cells into designs by replacing nonscan sequential cells with their scan equivalents. When starting with a gate-level netlist, scan replacement occurs as an independent process. If one starts with an HDL description of the design, scan replacement occurs during the initial mapping of the design to gates. Figure 4.6 shows the Design-for-Test flow from an HDL design (unmapped design).

The test methodology is selected based on testability, area, and performance requirements. Either a full-scan test methodology or a partial-scan methodology can be used.

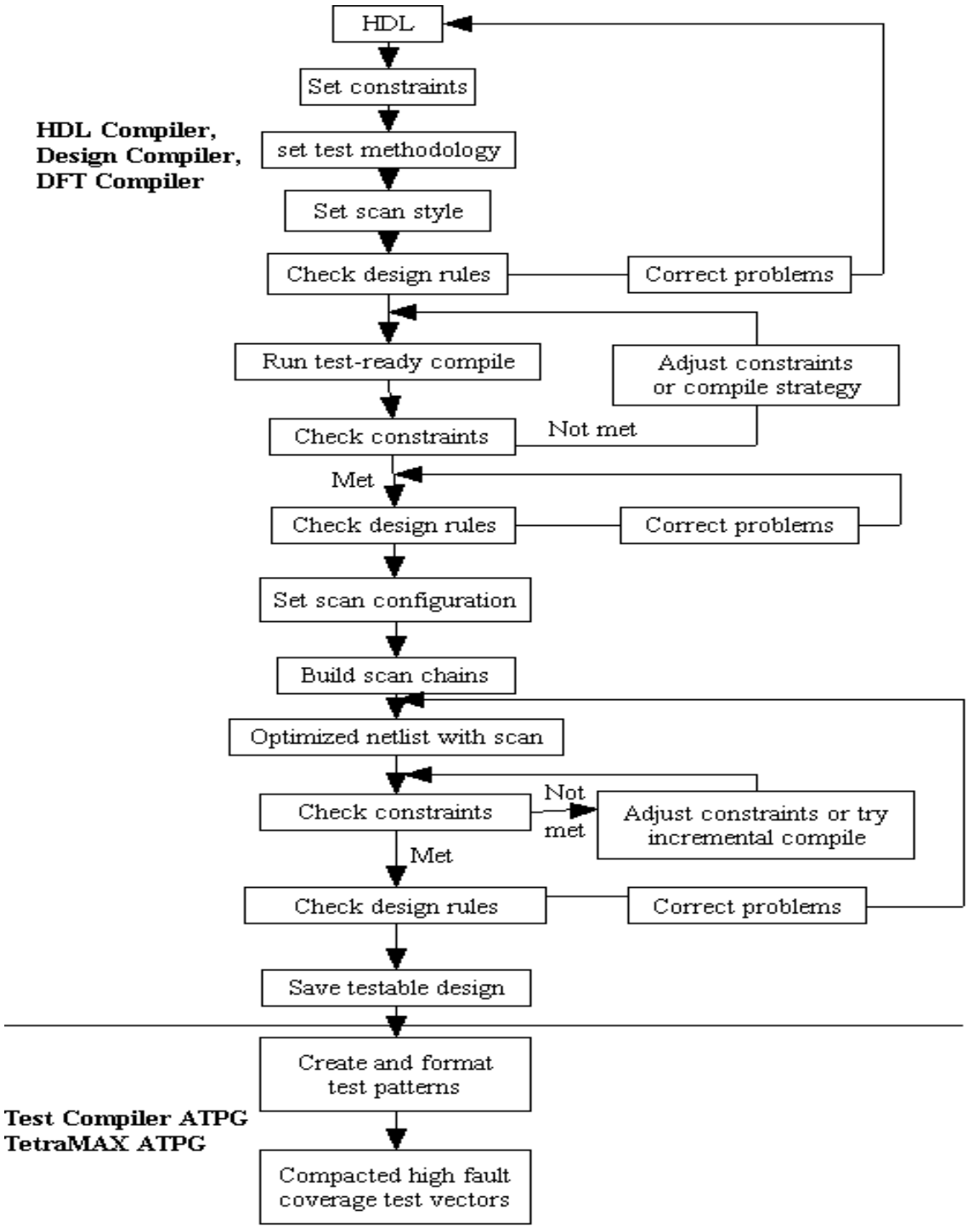

Figure 4.6: Design-for-Test flow from an unmapped design (referenced from [7])

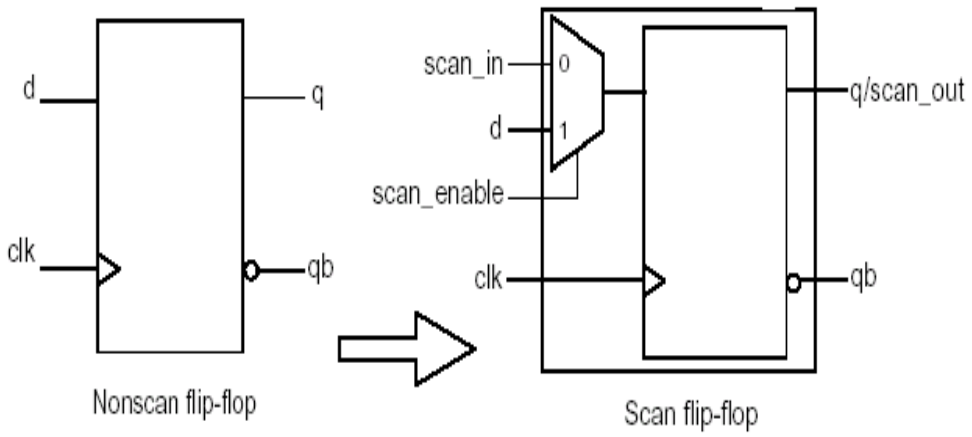

Figure 4.7: Nonscan flip-flop and multiplexed scan flip-flop (referenced from [7])

With full-scan test methodology, DFT Compiler replaces all sequential cells in the design with their scannable equivalents during scan insertion. With partial scan, some, but not all, of the sequential cells are replaced. Full scan typically provides higher fault coverage than partial scan with the price of bigger design area and decreased design performance.

DFT Compiler supports four scan styles: Multiplexed flip-flop, Clocked scan, Level-sensitive scan design (LSSD) and Auxiliary-clock LSSD. DFT Compiler requires a scan style to perform scan synthesis. The scan style dictates the appropriate scan cells to insert during optimization. Multiplexed flip-flop is the scan style most commonly supported in technology libraries. Figure 4.7 shows a D flip-flop modified to support internal scan by the addition of a multiplexer.

In Figure 4.7, the data input of the flip-flop (d) and the scan input signal (scan_in)

are the inputs to the multiplexer. The scan-enable signal (scan_enable) controls which multiplexer input is active. During functional mode, the scan_enable signal selects the system data input (d). During scan shift, the scan_enable signal selects the scan data input (scan_in). The data output of the flip-flop (q) is used as the scan output signal (scan_out). The scan_out signal is connected to the scan_in signal of the next scan cell to form a serial scan (shift) capability.

Test-ready compile maps all sequential cells directly to scan cells. During optimization, DFT Compiler considers the design constraints and the impact of both the scan cells themselves and the additional loading due to scan chain routing to minimize the impact of the scan structure on the design.

The scan sequential cells are chained together to form one or more large shift registers, which are called scan chains or scan paths. Because scan cells are larger than the nonscan cells they replace, and additional area is needed for the nets for the scan signals, usually the size and power for the design will increase after adding scan circuitry. Also, speed might decrease marginally because of changes in the electrical characteristics of the scan cells.

4.2.3 Design Compiler code

The following is a Design Compiler script which generates the Verilog file of the testable design and the test pattern.

```
dc_shell> analyze  -f vhdl ../vhdl/desenc_testable.vhd
dc_shell> elaborate desenc_testable
dc_shell> link
dc_shell> current_design desenc_testable
dc_shell> create_clock -name "clk" -period 25 -waveform {0 12.5} {"clk"}
dc_shell> uniquify
dc_shell> compile
dc_shell> current_design desenc_testable
dc_shell> set_test_methodology full_scan
```

```
dc_shell> set_scan_style multiplexed_flip_flop
dc_shell> check_test
dc_shell> set_scan_configuration -chain_count 2
dc_shell> insert_scan
dc_shell> current_design desenc_testable
dc_shell> create_test_clock clk -period 100 -waveform{35 65}
dc_shell> create_test_patterns -output desenc_testable.vdb -compaction_effort low -check_contention true -check_float t
dc_shell> write_test -input desenc_testable.vdb -format vhdl -output desenc_testable
dc_shell> write -format Verilog -hierarchy -output desenc_testable.v
dc_shell> quit
```

The generated Verilog file contains the gate-level netlist. Cadence's Silicon Ensemble can take this Verilog file as input to generate the physical-level layout.

The results generated below indicate that a fault coverage of near 100 percent is achieved.

```
Combinational Test Pattern Generation starts:

    Start random pattern generation...

       83.06% faults processed ; cumulative fault coverage =  83.06%
       93.51% faults processed ; cumulative fault coverage =  93.51%
       97.77% faults processed ; cumulative fault coverage =  97.77%
       99.18% faults processed ; cumulative fault coverage =  99.18%
       99.69% faults processed ; cumulative fault coverage =  99.69%
       99.86% faults processed ; cumulative fault coverage =  99.86%
       99.95% faults processed ; cumulative fault coverage =  99.95%
       99.99% faults processed ; cumulative fault coverage =  99.99%
       99.99% faults processed ; cumulative fault coverage =  99.99%

    ...End random pattern generation

    Start deterministic pattern generation...

      100.00% faults processed ; cumulative fault coverage = 100.00%
```

```
...End deterministic pattern generation


                          Non-collapsed     Collapsed
No. of detected faults    103508            59218
No. of abandoned faults   0                 0
No. of tied faults        0                 0
No. of redundant faults   0                 0
No. of untested faults    0                 0
Total no. of faults       103508            59218
Fault coverage            100.00            100.00

No. of test patterns       188

Test Generation Time (CPU)  12.47 sec

Start compaction...

...End compaction

No. of compacted patterns   188

Compaction Time (CPU)       5.53 sec
```

4.3 Placement and routing with Silicon Ensemble

Cadence's Silicon Ensemble is a multiengine automatic place and route platform for custom designs. It can take Verilog netlist as input and generate def files as layout files. Figure 4.8 shows a typical design flow with Silicon Ensemble [18].

4.3.1 Placement and routing for testable design

The general steps to do placement and routing with Silicon Ensemble usually include: import LEF files, import Verilog netlist, initialize floorplan, place IOs, place cells, add filler

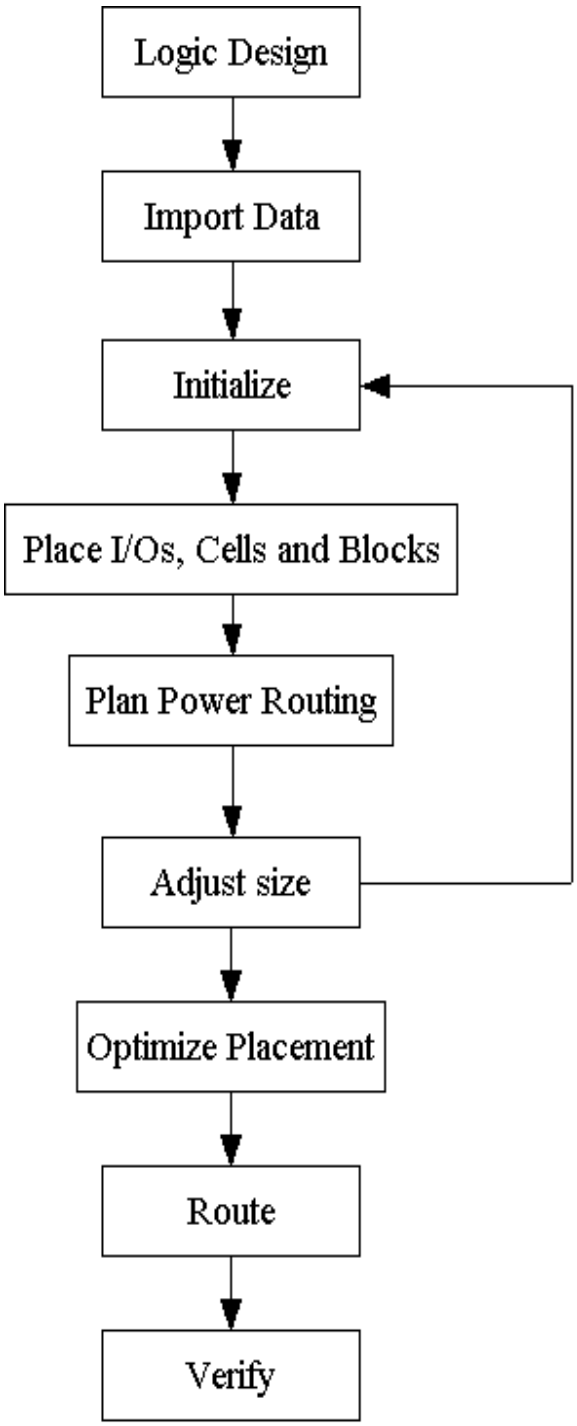

Figure 4.8: Typical Silicon Ensemble design flow

cells, add power and ground rings, and route. The following sections describe the procedure to place and route the "desenc_testable" design.

4.3.1.1 Import data

After starting Silicon Ensemble, the first step is to import the Library Exchange Format (LEF) file with command

```
FINPUT LEF FILENAME "/sw/CDS/ARTISAN/TSMC18/aci/sc/lef/tsmc18_6lm.lef"  REPORTFILE "importlef.rpt";
```

The "tsmc18_6lm.lef" file, which is provided by the cell-library provider, contains technology and cell information for the design. It tells Silicon Ensemble where the pins of each cell are located as well as cell abstract view information. Then the library Verilog file "tsmc18.v" and the design Verilog netlist "desenc_testable.v" are also read in.

4.3.1.2 Initialize

Initialization sets up the physical design based on the data read in. It creates a core area with rows or columns, I/O rows around the core area, and sets up the Gcell and Rgrid tracks.

Figure 4.9 shows the Initialize Floorplan window of the desenc_testable design. As seen in that window, there are altogether 106 rows for standard cells and 85 percent of core row is utilized. "Flip Every Other Row" button is selected so that the ground of two adjacent rows is overlapped. The total area is therefore smaller because some space between rows is saved.

4.3.1.3 Place I/Os, Blocks and Cells

After initializing the floorplan, the I/Os are placed with random placement mode and evenly spaced. Since there is no block in this design, "Place Block" is not used here. But it is used later for hierarchical placement and routing.

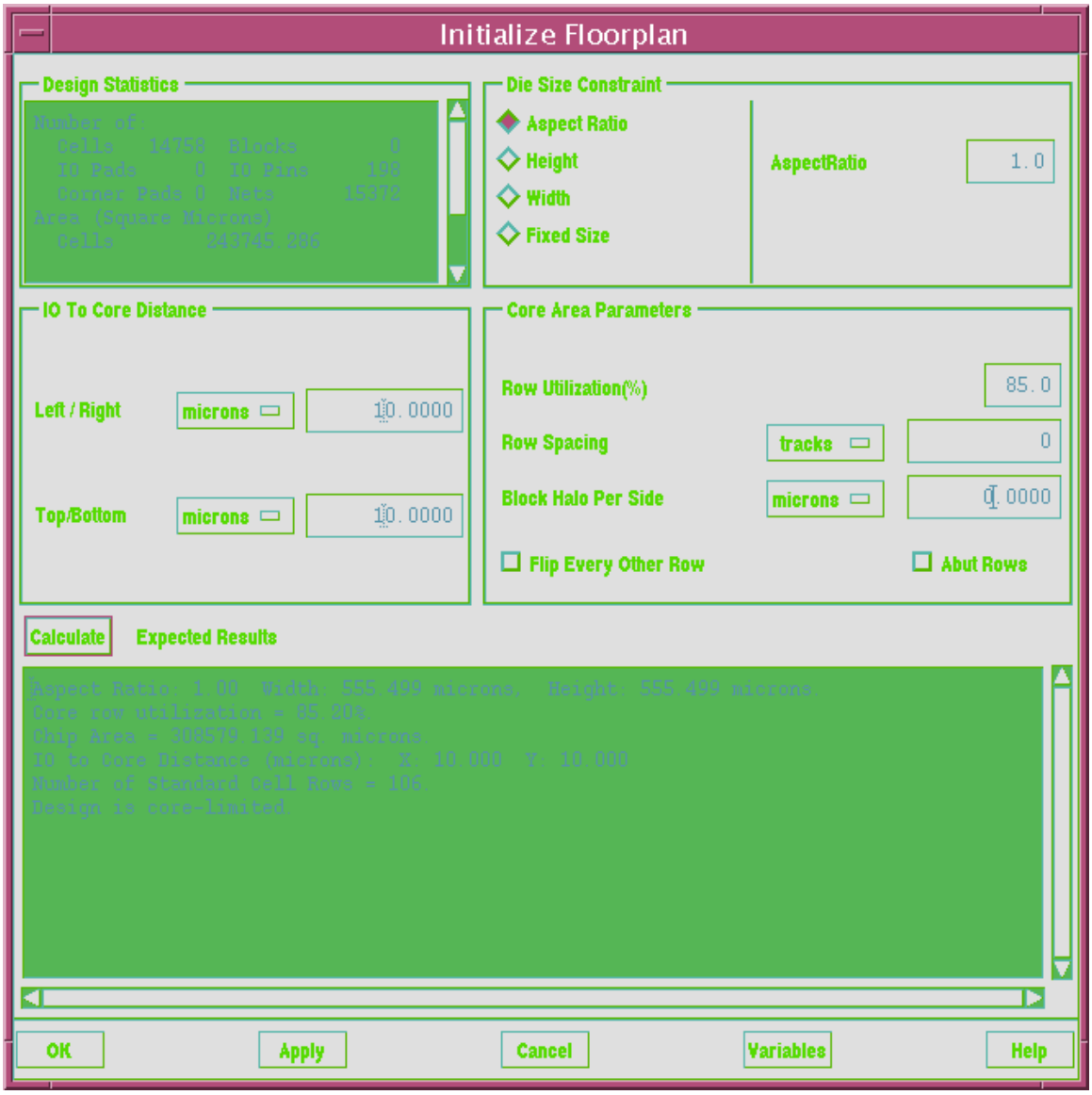

Figure 4.9: The Initialize Floorplan window

Finally, all the 14758 standard cells are placed from "Place Cells" window. Since there are still gaps between cells, filler cells, which are standard cells without transistors, have to be added. In order to decrease the number of cells, the biggest filler cell model FILL64 is first selected and the smaller models like FILL32, FILL16, FILL8, FILL4, FILL2 and FILL1 are subsequently used.

4.3.1.4 Plan power routing and final routing

The last step before routing is to add power and ground rings. After that, the global and final routing is run with auto search and repair mode. Finally the routed design is saved. Figure 4.10 shows the picture of the routed design in Silicon Ensemble window.

4.3.1.5 Postlayout simulation

After the routed design is available, the Standard Delay Format (SDF) file "desenc_testable.sdf" is extracted from it. With that delay information, the postlayout simulation is done with Modelsim. Figure 4.11 and figure 4.12 show the simulation result.

4.3.1.6 Virtuoso layout

In Silicon Ensemble the routed design is saved as def file — "desenc_testable.def". This def file is opened with Virtuoso Layout Editor to get the layout as shown in Figure 4.13. Figure 4.14 shows the zoomed-in view.

4.3.2 Hierarchical placement and routing

The placement and routing procedure described in the last section can be used in normal cases. However, it is not very suitable to process big designs because the routing would be very complex and require a huge amount of computer running time. In that case, hierarchical placement and routing would be a better option. It can speed up the P&R

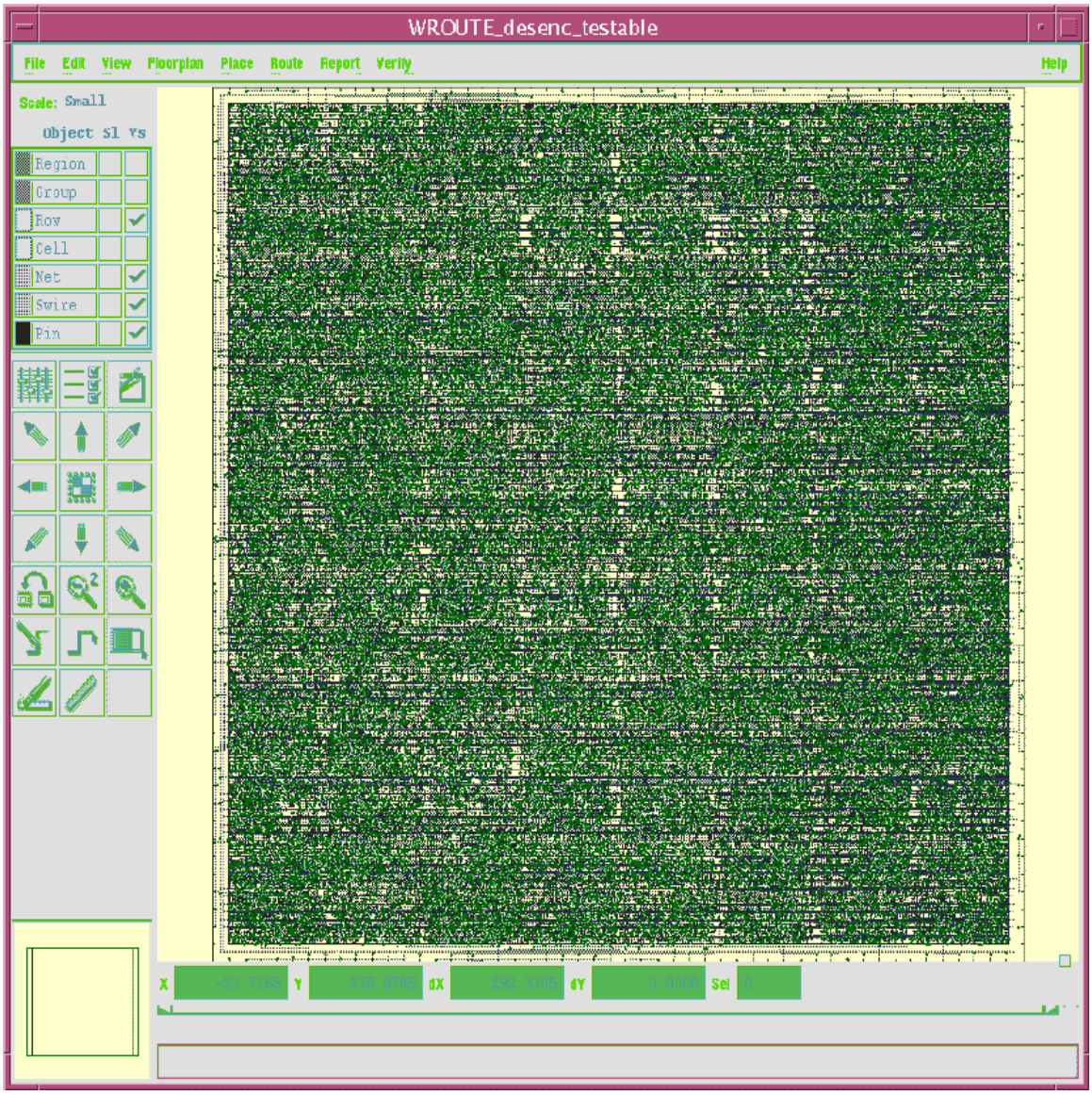

Figure 4.10: The final routed design

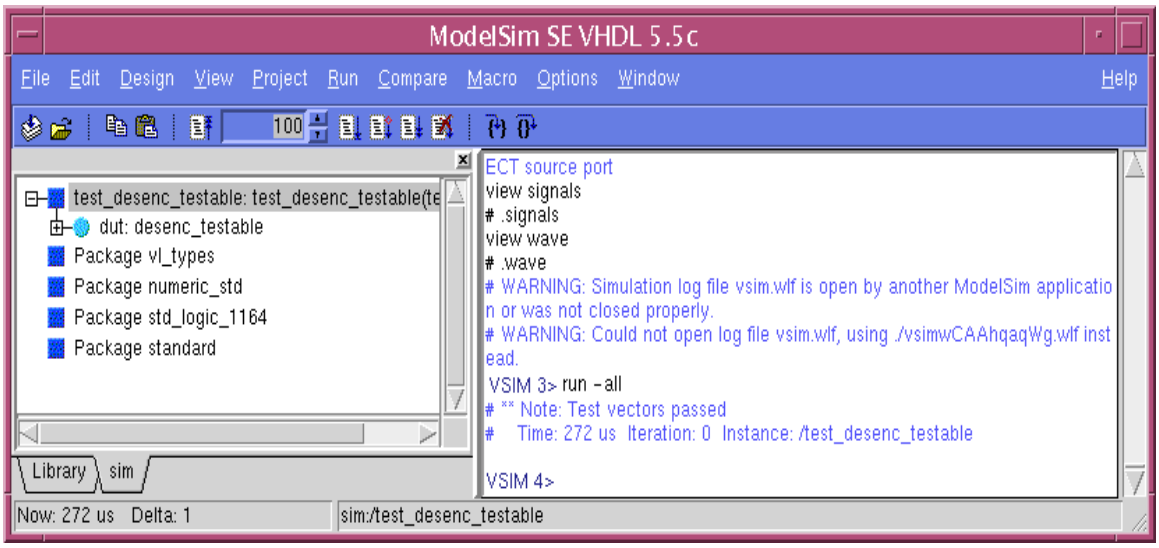

Figure 4.11: The main window of Modelsim for postlayout simulation

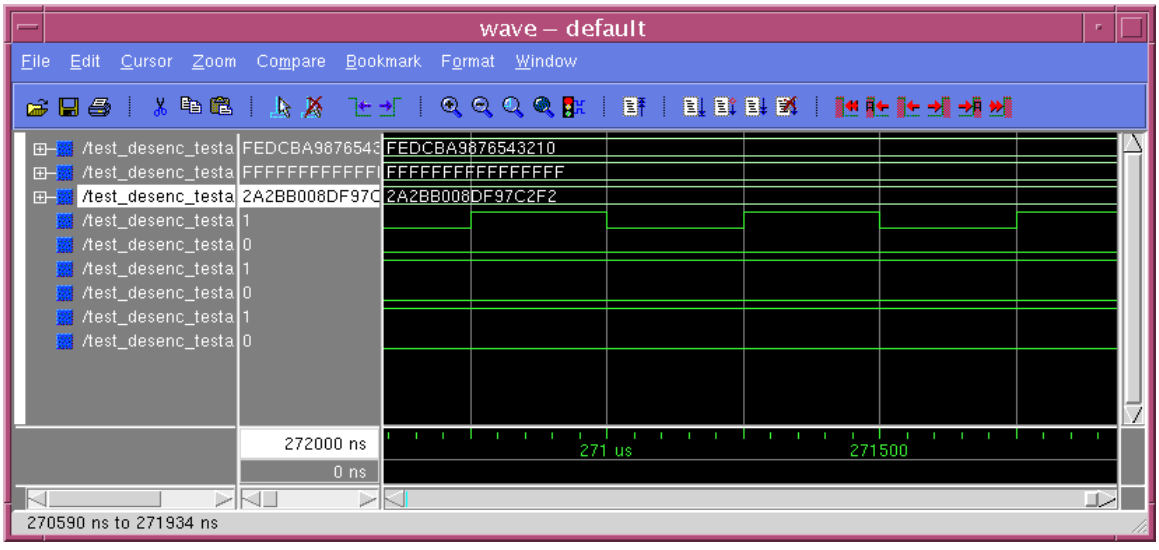

Figure 4.12: The wave window of Modelsim for postlayout simulation

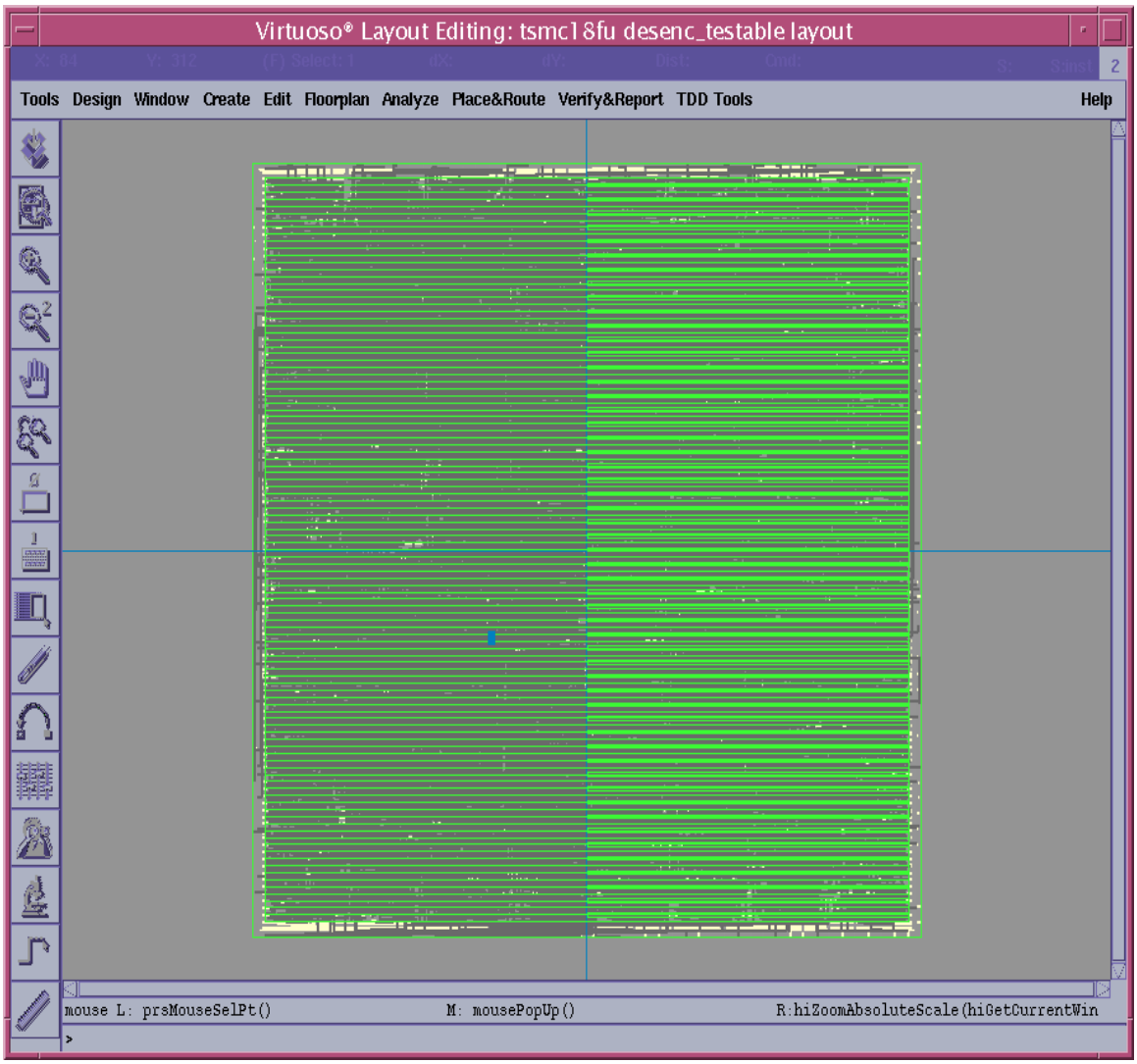

Figure 4.13: The layout in Virtuoso Layout Editor

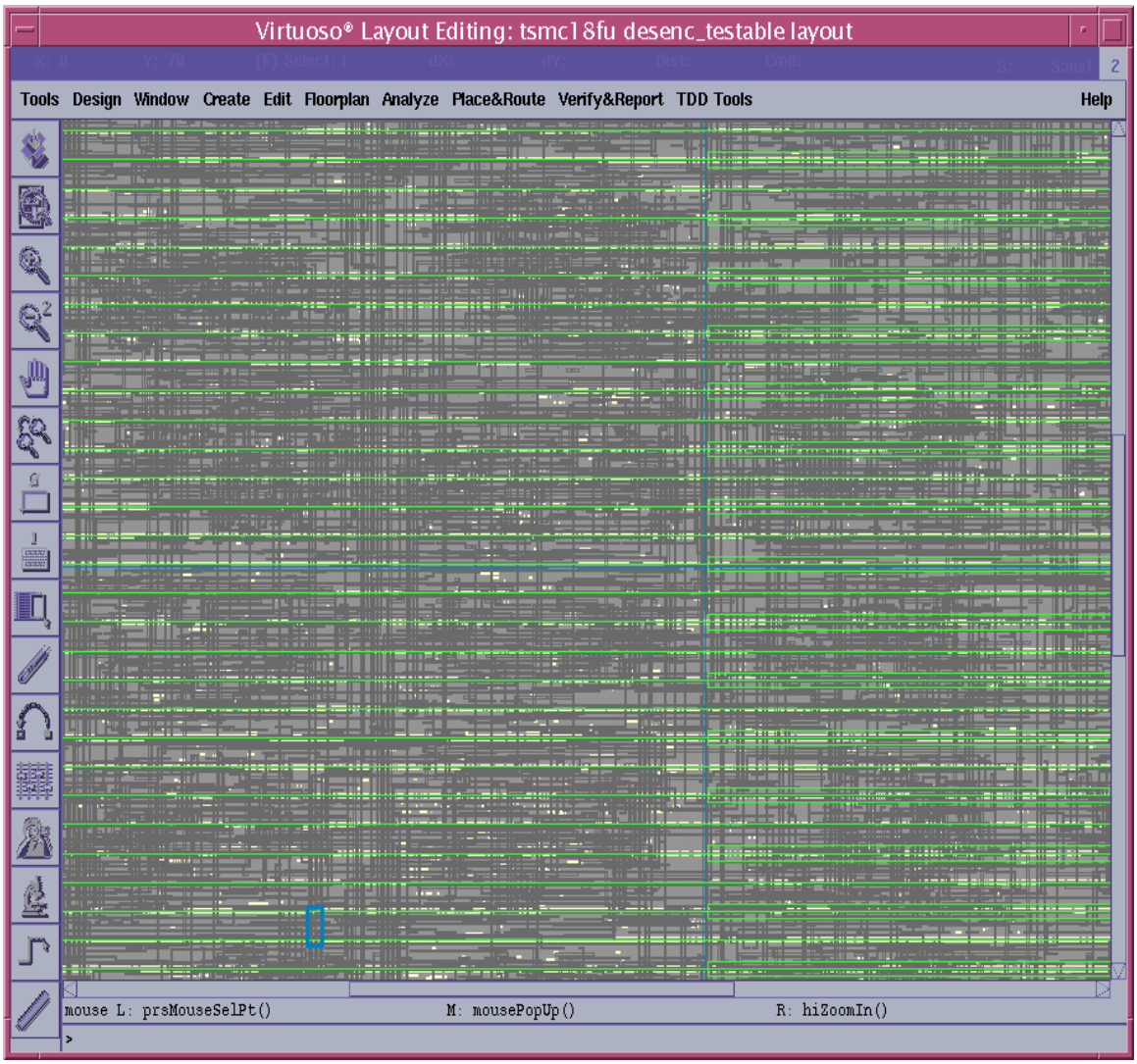

Figure 4.14: Zoomed-in layout in Virtuoso Layout Editor

process and make the debug work easier, although with the price of bigger area usually.

The "desenc" design consists of a "keysched" block, an "ip" block, a "fp" block and 16 "roundfunc" blocks. This makes it possible to decompose the whole design into pieces. Especially the "roundfunc" blocks can be compiled separately and integrated into the top design later. With the above consideration, the original design vhdl file "desenc.vhd" is decomposed into two vhdl files – the new "desenc.vhd" and "roundfunc.vhd", and these two files are synthesized into two Verilog netlists: "roundfunc.v" and "desenc.v". "Roundfunc.v" is then placed and routed and saved as a block and called directly during the placement and routing process of the top design.

The original "desenc.vhd" is also rewritten to include an unnecessary OR gate in the design. This seemingly ridiculous step is performed only because Silicon Ensemble needs some core component during the process of placement and routing.

After going through all the normal steps mentioned in last section, such as place I/Os, blocks and cells and wroute, the final layout is obtained as shown in Figure 4.15. The layout of "roundfunc" block is shown in Figure 4.16.

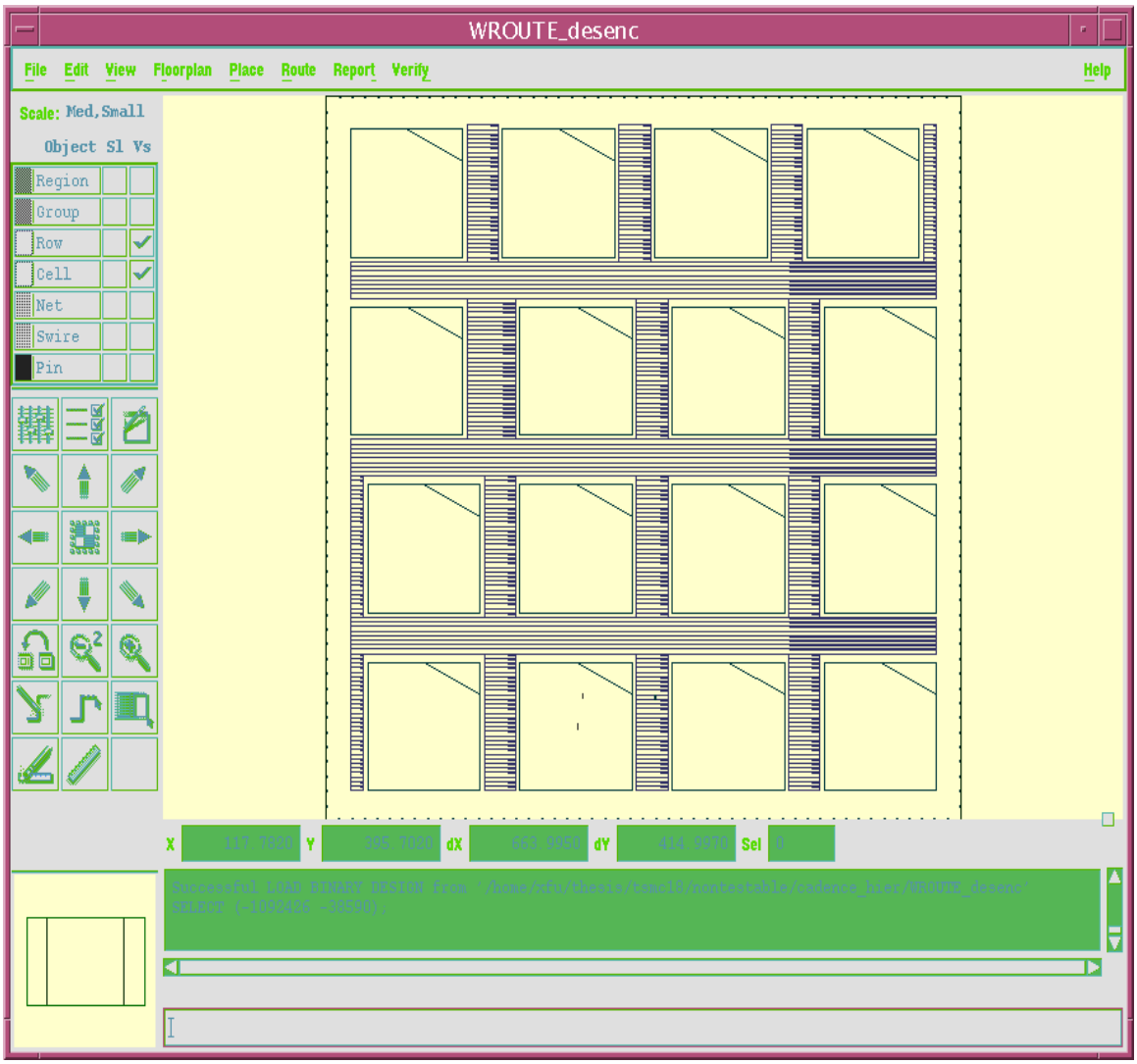

Figure 4.15: The layout from hierarchical Placement and Routing

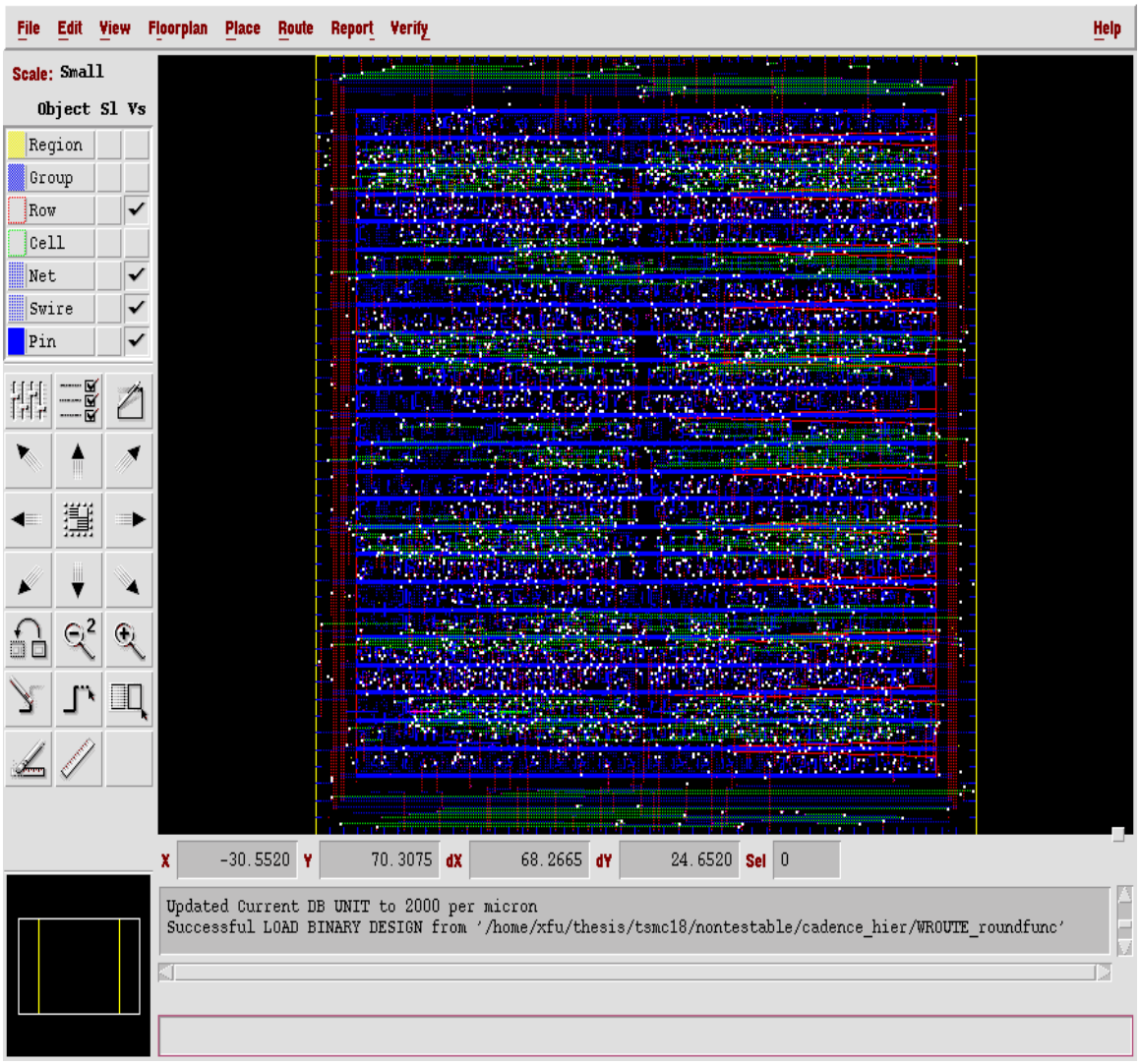

Figure 4.16: The layout of "roundfunc" block

Chapter 5

Summary, conclusions and future work

To protect people's privacy, cryptography technology is becoming more and more important in the communication area. The rapid progress of VLSI technology benefits the hardware realization of encryption and decryption a lot, making the devices smaller, faster, and less power-consuming.

In this thesis, the national Data Encryption Standard is implemented in various FPGA and ASIC technology with Synopsys, Cadence and Mentor Graphics tools. Implementations with different tools are compared. Design space is explored to get different versions of ASIC optimized respectively to different optimization goals. The comparison of those versions illustrated the tradeoff between area, speed and power. Thereafter, the design was made testable by inserting scan chains with DFT compiler. The report shows that 100 percent fault coverage is achieved. Finally, the Cadence tools were used to place and route the design to get the physical layout. The simulation made at each level verified the success of the implementations.

This thesis demonstrated the design flow for developing both FPGA and ASIC implementations of a big application. It can be used as a tutorial of basic FPGA and ASIC implementations and introductory use of some synthesis, simulation, placement & routing

and layout tools. In the future, the use of the tools needs to be explored more to utilize the tools more efficiently. All the implementations can be optimized further to their optimizing goals. Various implementations of DES can be integrated to the real application environment to test all the parameters.

Bibliography

Bibliography

[1] Cadence low-power synthesis option for buildgates and pks. Available: http://www.cadence.com/datasheets/low_power_synth_option.html.

[2] Cadence Silicon Ensemble Datasheet. Available: http://www.cadence.com/products/sepks.html.

[3] DC Ultra - Design Compiler at its Best. Available: http://www.synopsys.com/products/logic/dc_ultra_ds.html.

[4] DES vhdl code. Available: http://www.cse.cuhk.edu.hk/čeg5010/des_core.tar.gz.

[5] Design of VLSI systems, ch.7. Available: http://vlsi.wpi.edu/webcourse/ch07/ch07.html#7.1.

[6] DFT compiler overview. Synopsys Inc.

[7] DFT compiler user guide. Synopsys Inc.

[8] Leonardo Spectrum Datasheet. Available: http://www.mentor.com/leonardospectrum.

[9] ModelSim Product Literature. Available: http://www.model.com/products/literature.asp.

[10] Power Compiler Datasheet. Available: http://www.synopsys.com/products/power/power_bkg.html.

[11] Power compiler user guide. Synopsys Inc.

[12] Synopsys DesignPower and Power Compiler Technology Backgrounder. Available: http://www.synopsys.com.tw/taiwan/tech/compiler.html.

[13] Synopsys Product Line Guide. Available: http://www.synopsys.com/products/product_lineguide.html#17.

[14] Virtuoso layout editor. Available: http://www.cadence.com/products/veditor.html.

[15] N. H.E.Weste and K. Eshraghian. Principles of CMOS VLSI design, A systems perspective. Addison-Wesley, first edition, 1991.

[16] J. M. Rabaey. Digital integrated circuits, a design perspective. Addison-Wesley, first edition, 1991.

[17] B. Sklar. Digital Communications, Fundamentals and Applications. Addison-Wesley, first edition, 1991.

[18] K. Thirunagari. Silicon Ensemble flow and example. Available: http://www.utdallas.edu/g̃rinnell/eda.

Appendices

Appendix A

Selected Software Listings

A.1 desenc.vhd

```
--- downloaded from the website: http://www.cse.cuhk.edu.hk/~ceg5010
--- and modified
library ieee;

use ieee.std_logic_1164.all;

entity pc1 is port
(
key : in std_logic_vector(1 TO 64);
c0x,d0x : out std_logic_vector(1 TO 28)
);
end pc1;

architecture behavior of pc1 is
signal XX : std_logic_vector(1 to 56);
begin
XX(1)<=key(57); XX(2)<=key(49); XX(3)<=key(41); XX(4)<=key(33); XX(5)<=key(25); XX(6)<=key(17); XX(7)<=key(9);
XX(8)<=key(1); XX(9)<=key(58); XX(10)<=key(50); XX(11)<=key(42); XX(12)<=key(34); XX(13)<=key(26); XX(14)<=key(18);
XX(15)<=key(10); XX(16)<=key(2); XX(17)<=key(59); XX(18)<=key(51); XX(19)<=key(43); XX(20)<=key(35); XX(21)<=key(27);
XX(22)<=key(19); XX(23)<=key(11); XX(24)<=key(3); XX(25)<=key(60); XX(26)<=key(52); XX(27)<=key(44); XX(28)<=key(36);
XX(29)<=key(63); XX(30)<=key(55); XX(31)<=key(47); XX(32)<=key(39); XX(33)<=key(31); XX(34)<=key(23); XX(35)<=key(15);
XX(36)<=key(7); XX(37)<=key(62); XX(38)<=key(54); XX(39)<=key(46); XX(40)<=key(38); XX(41)<=key(30); XX(42)<=key(22);
XX(43)<=key(14); XX(44)<=key(6); XX(45)<=key(61); XX(46)<=key(53); XX(47)<=key(45); XX(48)<=key(37); XX(49)<=key(29);
XX(50)<=key(21); XX(51)<=key(13); XX(52)<=key(5); XX(53)<=key(28); XX(54)<=key(20); XX(55)<=key(12); XX(56)<=key(4);

c0x<=XX(1 to 28); d0x<=XX(29 to 56);
end behavior;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity pc2 is port
(
c,d : in std_logic_vector(1 TO 28);
k : out std_logic_vector(1 TO 48)
);
end pc2;
 architecture behavior of pc2 is
signal YY : std_logic_vector(1 to 56);
begin
YY(1 to 28)<=c; YY(29 to 56)<=d;

k(1)<=YY(14); k(2)<=YY(17); k(3)<=YY(11); k(4)<=YY(24); k(5)<=YY(1); k(6)<=YY(5);
k(7)<=YY(3); k(8)<=YY(28); k(9)<=YY(15); k(10)<=YY(6); k(11)<=YY(21); k(12)<=YY(10);
k(13)<=YY(23); k(14)<=YY(19); k(15)<=YY(12); k(16)<=YY(4); k(17)<=YY(26); k(18)<=YY(8);
k(19)<=YY(16); k(20)<=YY(7); k(21)<=YY(27); k(22)<=YY(20); k(23)<=YY(13); k(24)<=YY(2);
k(25)<=YY(41); k(26)<=YY(52); k(27)<=YY(31); k(28)<=YY(37); k(29)<=YY(47); k(30)<=YY(55);
k(31)<=YY(30); k(32)<=YY(40); k(33)<=YY(51); k(34)<=YY(45); k(35)<=YY(33); k(36)<=YY(48);
k(37)<=YY(44); k(38)<=YY(49); k(39)<=YY(39); k(40)<=YY(56); k(41)<=YY(34); k(42)<=YY(53);
k(43)<=YY(46); k(44)<=YY(42); k(45)<=YY(50); k(46)<=YY(36); k(47)<=YY(29); k(48)<=YY(32);
end behavior;
```

```
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity keysched is port
(
key : in std_logic_vector(1 to 64);
k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
: out std_logic_vector(1 to 48)
);
end keysched;
 architecture behaviour of keysched is

COMPONENT pc1 port
(
key : in std_logic_vector(1 TO 64);
c0x,d0x : out std_logic_vector(1 TO 28)
);
end COMPONENT;

COMPONENT pc2 port
(
c,d : in std_logic_vector(1 TO 28);
k : out std_logic_vector(1 TO 48)
);
end COMPONENT;

signal c0x,c1x,c2x,c3x,c4x,c5x,c6x,c7x,c8x,c9x,c10x,c11x,c12x,c13x,c14x,c15x,c16x : std_logic_vector(1 to 28);
signal d0x,d1x,d2x,d3x,d4x,d5x,d6x,d7x,d8x,d9x,d10x,d11x,d12x,d13x,d14x,d15x,d16x : std_logic_vector(1 to 28);
begin
  upc1: pc1 port map ( key=>key, c0x=>c0x, d0x=>d0x );
---- modifed to make it synthesizable with UT's synthesis toolset
c1x <= c0x(2 to 28) & c0x(1);          d1x <= d0x(2 to 28) & d0x(1);
    c2x <= c1x(2 to 28) & c1x(1);          d2x <= d1x(2 to 28) & d1x(1);
    c3x <= c2x(3 to 28) & c2x(1 to 2);     d3x <= d2x(3 to 28) & d2x(1 to 2);
    c4x <= c3x(3 to 28) & c3x(1 to 2);     d4x <= d3x(3 to 28) & d3x(1 to 2);
    c5x <= c4x(3 to 28) & c4x(1 to 2);     d5x <= d4x(3 to 28) & d4x(1 to 2);
    c6x <= c5x(3 to 28) & c5x(1 to 2);     d6x <= d5x(3 to 28) & d5x(1 to 2);
    c7x <= c6x(3 to 28) & c6x(1 to 2);     d7x <= d6x(3 to 28) & d6x(1 to 2);
    c8x <= c7x(3 to 28) & c7x(1 to 2);     d8x <= d7x(3 to 28) & d7x(1 to 2);
    c9x <= c8x(2 to 28) & c8x(1);          d9x <= d8x(2 to 28) & d8x(1);
    c10x <= c9x(3 to 28) & c9x(1 to 2);     d10x <= d9x(3 to 28) & d9x(1 to 2);
    c11x <= c10x(3 to 28) & c10x(1 to 2);     d11x <= d10x(3 to 28) & d10x(1 to 2);
    c12x <= c11x(3 to 28) & c11x(1 to 2);     d12x <= d11x(3 to 28) & d11x(1 to 2);
    c13x <= c12x(3 to 28) & c12x(1 to 2);     d13x <= d12x(3 to 28) & d12x(1 to 2);
    c14x <= c13x(3 to 28) & c13x(1 to 2);     d14x <= d13x(3 to 28) & d13x(1 to 2);
    c15x <= c14x(3 to 28) & c14x(1 to 2);     d15x <= d14x(3 to 28) & d14x(1 to 2);
    c16x <= c15x(2 to 28) & c15x(1);       d16x <= d15x(2 to 28) & d15x(1);

  pc2x1: pc2 port map ( c=>c1x, d=>d1x, k=>k1x );
pc2x2: pc2 port map ( c=>c2x, d=>d2x, k=>k2x );
pc2x3: pc2 port map ( c=>c3x, d=>d3x, k=>k3x );
pc2x4: pc2 port map ( c=>c4x, d=>d4x, k=>k4x );
pc2x5: pc2 port map ( c=>c5x, d=>d5x, k=>k5x );
pc2x6: pc2 port map ( c=>c6x, d=>d6x, k=>k6x );
pc2x7: pc2 port map ( c=>c7x, d=>d7x, k=>k7x );
pc2x8: pc2 port map ( c=>c8x, d=>d8x, k=>k8x );
pc2x9: pc2 port map ( c=>c9x, d=>d9x, k=>k9x );
pc2x10: pc2 port map ( c=>c10x, d=>d10x, k=>k10x );
pc2x11: pc2 port map ( c=>c11x, d=>d11x, k=>k11x );
pc2x12: pc2 port map ( c=>c12x, d=>d12x, k=>k12x );
pc2x13: pc2 port map ( c=>c13x, d=>d13x, k=>k13x );
pc2x14: pc2 port map ( c=>c14x, d=>d14x, k=>k14x );
pc2x15: pc2 port map ( c=>c15x, d=>d15x, k=>k15x );
pc2x16: pc2 port map ( c=>c16x, d=>d16x, k=>k16x );
end;

library ieee;

use ieee.std_logic_1164.all;

entity ip is port
(
pt : in std_logic_vector(1 TO 64);
l0x : out std_logic_vector(1 TO 32);
r0x : out std_logic_vector(1 TO 32)
);
end ip;

architecture behavior of ip is
begin
l0x(1)<=pt(58); l0x(2)<=pt(50); l0x(3)<=pt(42); l0x(4)<=pt(34);
l0x(5)<=pt(26); l0x(6)<=pt(18); l0x(7)<=pt(10); l0x(8)<=pt(2);
```

```
l0x(9)<=pt(60); l0x(10)<=pt(52); l0x(11)<=pt(44); l0x(12)<=pt(36);
l0x(13)<=pt(28); l0x(14)<=pt(20); l0x(15)<=pt(12); l0x(16)<=pt(4);
l0x(17)<=pt(62); l0x(18)<=pt(54); l0x(19)<=pt(46); l0x(20)<=pt(38);
l0x(21)<=pt(30); l0x(22)<=pt(22); l0x(23)<=pt(14); l0x(24)<=pt(6);
l0x(25)<=pt(64); l0x(26)<=pt(56); l0x(27)<=pt(48); l0x(28)<=pt(40);
l0x(29)<=pt(32); l0x(30)<=pt(24); l0x(31)<=pt(16); l0x(32)<=pt(8);

r0x(1)<=pt(57); r0x(2)<=pt(49); r0x(3)<=pt(41); r0x(4)<=pt(33);
r0x(5)<=pt(25); r0x(6)<=pt(17); r0x(7)<=pt(9); r0x(8)<=pt(1);
r0x(9)<=pt(59); r0x(10)<=pt(51); r0x(11)<=pt(43); r0x(12)<=pt(35);
r0x(13)<=pt(27); r0x(14)<=pt(19); r0x(15)<=pt(11); r0x(16)<=pt(3);
r0x(17)<=pt(61); r0x(18)<=pt(53); r0x(19)<=pt(45); r0x(20)<=pt(37);
r0x(21)<=pt(29); r0x(22)<=pt(21); r0x(23)<=pt(13); r0x(24)<=pt(5);
r0x(25)<=pt(63); r0x(26)<=pt(55); r0x(27)<=pt(47); r0x(28)<=pt(39);
r0x(29)<=pt(31); r0x(30)<=pt(23); r0x(31)<=pt(15); r0x(32)<=pt(7);
end behavior;

library ieee;

use ieee.std_logic_1164.all;
 entity xp is port
(
ri : in std_logic_vector(1 TO 32);
e : out std_logic_vector(1 TO 48));
end xp;
 architecture behavior of xp is
begin
e(1)<=ri(32); e(2)<=ri(1); e(3)<=ri(2); e(4)<=ri(3); e(5)<=ri(4); e(6)<=ri(5); e(7)<=ri(4); e(8)<=ri(5);
e(9)<=ri(6); e(10)<=ri(7); e(11)<=ri(8); e(12)<=ri(9); e(13)<=ri(8); e(14)<=ri(9); e(15)<=ri(10); e(16)<=ri(11);
e(17)<=ri(12); e(18)<=ri(13); e(19)<=ri(12); e(20)<=ri(13); e(21)<=ri(14); e(22)<=ri(15); e(23)<=ri(16); e(24)<=ri(17);
e(25)<=ri(16); e(26)<=ri(17); e(27)<=ri(18); e(28)<=ri(19); e(29)<=ri(20); e(30)<=ri(21); e(31)<=ri(20); e(32)<=ri(21);
e(33)<=ri(22); e(34)<=ri(23); e(35)<=ri(24); e(36)<=ri(25); e(37)<=ri(24); e(38)<=ri(25); e(39)<=ri(26); e(40)<=ri(27);
e(41)<=ri(28); e(42)<=ri(29); e(43)<=ri(28); e(44)<=ri(29); e(45)<=ri(30); e(46)<=ri(31); e(47)<=ri(32); e(48)<=ri(1);
end behavior;
 library ieee;
use ieee.std_logic_1164.all;
 entity desxor1 is port
(
e : in std_logic_vector(1 TO 48);
b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
: out std_logic_vector (1 TO 6);
k : in std_logic_vector (1 TO 48)
); end desxor1;
 architecture behavior of desxor1 is
signal XX : std_logic_vector( 1 to 48);
begin
XX<=k xor e;
b1x<=XX(1 to 6);
b2x<=XX(7 to 12);
b3x<=XX(13 to 18);
b4x<=XX(19 to 24);
b5x<=XX(25 to 30);  b6x<=XX(31 to 36);
b7x<=XX(37 to 42);
b8x<=XX(43 to 48);
end behavior;

library ieee;
 use ieee.std_logic_1164.all;
 entity s1 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s1;
 architecture behaviour of s1 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"e";
when b"000010"=> so<=x"4";
when b"000100"=> so<=x"d";
when b"000110"=> so<=x"1";
when b"001000"=> so<=x"2";
when b"001010"=> so<=x"f";
when b"001100"=> so<=x"b";
when b"001110"=> so<=x"8";
when b"010000"=> so<=x"3";
when b"010010"=> so<=x"a";
```

```
when b"010100"=> so<=x"6";
when b"010110"=> so<=x"c";
when b"011000"=> so<=x"5";
when b"011010"=> so<=x"9";
when b"011100"=> so<=x"0";
when b"011110"=> so<=x"7";
when b"000001"=> so<=x"0";
when b"000011"=> so<=x"f";
when b"000101"=> so<=x"7";
when b"000111"=> so<=x"4";
when b"001001"=> so<=x"e";
when b"001011"=> so<=x"2";
when b"001101"=> so<=x"d";
when b"001111"=> so<=x"1";
when b"010001"=> so<=x"a";
when b"010011"=> so<=x"6";
when b"010101"=> so<=x"c";
when b"010111"=> so<=x"b";
when b"011001"=> so<=x"9";
when b"011011"=> so<=x"5";
when b"011101"=> so<=x"3";
when b"011111"=> so<=x"8";
when b"100000"=> so<=x"4";
when b"100010"=> so<=x"1";
when b"100100"=> so<=x"e";
when b"100110"=> so<=x"8";
when b"101000"=> so<=x"d";
when b"101010"=> so<=x"6";
when b"101100"=> so<=x"2";
when b"101110"=> so<=x"b";
when b"110000"=> so<=x"f";
when b"110010"=> so<=x"c";
when b"110100"=> so<=x"9";
when b"110110"=> so<=x"7";
when b"111000"=> so<=x"3";
when b"111010"=> so<=x"a";
when b"111100"=> so<=x"5";
when b"111110"=> so<=x"0";
when b"100001"=> so<=x"f";
when b"100011"=> so<=x"c";
when b"100101"=> so<=x"8";
when b"100111"=> so<=x"2";
when b"101001"=> so<=x"4";
when b"101011"=> so<=x"9";
when b"101101"=> so<=x"1";
when b"101111"=> so<=x"7";
when b"110001"=> so<=x"5";
when b"110011"=> so<=x"b";
when b"110101"=> so<=x"3";
when b"110111"=> so<=x"e";
when b"111001"=> so<=x"a";
when b"111011"=> so<=x"0";
when b"111101"=> so<=x"6";
when others=> so<=x"d";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s2 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s2;
 architecture behaviour of s2 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"f";
when b"000010"=> so<=x"1";
when b"000100"=> so<=x"8";
when b"000110"=> so<=x"e";
when b"001000"=> so<=x"6";
when b"001010"=> so<=x"b";
when b"001100"=> so<=x"3";
when b"001110"=> so<=x"4";
```

```
when b"010000"=> so<=x"9";
when b"010010"=> so<=x"7";
when b"010100"=> so<=x"2";
when b"010110"=> so<=x"d";
when b"011000"=> so<=x"c";
when b"011010"=> so<=x"0";
when b"011100"=> so<=x"5";
when b"011110"=> so<=x"a";
when b"000001"=> so<=x"3";
when b"000011"=> so<=x"d";
when b"000101"=> so<=x"4";
when b"000111"=> so<=x"7";
when b"001001"=> so<=x"f";
when b"001011"=> so<=x"2";
when b"001101"=> so<=x"8";
when b"001111"=> so<=x"e";
when b"010001"=> so<=x"c";
when b"010011"=> so<=x"0";
when b"010101"=> so<=x"1";
when b"010111"=> so<=x"a";
when b"011001"=> so<=x"6";
when b"011011"=> so<=x"9";
when b"011101"=> so<=x"b";
when b"011111"=> so<=x"5";
when b"100000"=> so<=x"0";
when b"100010"=> so<=x"e";
when b"100100"=> so<=x"7";
when b"100110"=> so<=x"b";
when b"101000"=> so<=x"a";
when b"101010"=> so<=x"4";
when b"101100"=> so<=x"d";
when b"101110"=> so<=x"1";
when b"110000"=> so<=x"5";
when b"110010"=> so<=x"8";
when b"110100"=> so<=x"c";
when b"110110"=> so<=x"6";
when b"111000"=> so<=x"9";
when b"111010"=> so<=x"3";
when b"111100"=> so<=x"2";
when b"111110"=> so<=x"f";
when b"100001"=> so<=x"d";
when b"100011"=> so<=x"8";
when b"100101"=> so<=x"a";
when b"100111"=> so<=x"1";
when b"101001"=> so<=x"3";
when b"101011"=> so<=x"f";
when b"101101"=> so<=x"4";
when b"101111"=> so<=x"2";
when b"110001"=> so<=x"b";
when b"110011"=> so<=x"6";
when b"110101"=> so<=x"7";
when b"110111"=> so<=x"c";
when b"111001"=> so<=x"0";
when b"111011"=> so<=x"5";
when b"111101"=> so<=x"e";
when others=> so<=x"9";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s3 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s3;
 architecture behaviour of s3 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"a";
when b"000010"=> so<=x"0";
when b"000100"=> so<=x"9";
when b"000110"=> so<=x"e";
when b"001000"=> so<=x"6";
when b"001010"=> so<=x"3";
```

```
when b"001100"=> so<=x"f";
when b"001110"=> so<=x"5";
when b"010000"=> so<=x"1";
when b"010010"=> so<=x"d";
when b"010100"=> so<=x"c";
when b"010110"=> so<=x"7";
when b"011000"=> so<=x"b";
when b"011010"=> so<=x"4";
when b"011100"=> so<=x"2";
when b"011110"=> so<=x"8";
when b"000001"=> so<=x"d";
when b"000011"=> so<=x"7";
when b"000101"=> so<=x"0";
when b"000111"=> so<=x"9";
when b"001001"=> so<=x"3";
when b"001011"=> so<=x"4";
when b"001101"=> so<=x"6";
when b"001111"=> so<=x"a";
when b"010001"=> so<=x"2";
when b"010011"=> so<=x"8";
when b"010101"=> so<=x"5";
when b"010111"=> so<=x"e";
when b"011001"=> so<=x"c";
when b"011011"=> so<=x"b";
when b"011101"=> so<=x"f";
when b"011111"=> so<=x"1";
when b"100000"=> so<=x"d";
when b"100010"=> so<=x"6";
when b"100100"=> so<=x"4";
when b"100110"=> so<=x"9";
when b"101000"=> so<=x"8";
when b"101010"=> so<=x"f";
when b"101100"=> so<=x"3";
when b"101110"=> so<=x"0";
when b"110000"=> so<=x"b";
when b"110010"=> so<=x"1";
when b"110100"=> so<=x"2";
when b"110110"=> so<=x"c";
when b"111000"=> so<=x"5";
when b"111010"=> so<=x"a";
when b"111100"=> so<=x"e";
when b"111110"=> so<=x"7";
when b"100001"=> so<=x"1";
when b"100011"=> so<=x"a";
when b"100101"=> so<=x"d";
when b"100111"=> so<=x"0";
when b"101001"=> so<=x"6";
when b"101011"=> so<=x"9";
when b"101101"=> so<=x"8";
when b"101111"=> so<=x"7";
when b"110001"=> so<=x"4";
when b"110011"=> so<=x"f";
when b"110101"=> so<=x"e";
when b"110111"=> so<=x"3";
when b"111001"=> so<=x"b";
when b"111011"=> so<=x"5";
when b"111101"=> so<=x"2";
when others=> so<=x"c";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s4 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s4;
 architecture behaviour of s4 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"7";
when b"000010"=> so<=x"d";
when b"000100"=> so<=x"e";
when b"000110"=> so<=x"3";
```

```
when b"001000"=> so<=x"0";
when b"001010"=> so<=x"6";
when b"001100"=> so<=x"9";
when b"001110"=> so<=x"a";
when b"010000"=> so<=x"1";
when b"010010"=> so<=x"2";
when b"010100"=> so<=x"8";
when b"010110"=> so<=x"5";
when b"011000"=> so<=x"b";
when b"011010"=> so<=x"c";
when b"011100"=> so<=x"4";
when b"011110"=> so<=x"f";
when b"000001"=> so<=x"d";
when b"000011"=> so<=x"8";
when b"000101"=> so<=x"b";
when b"000111"=> so<=x"5";
when b"001001"=> so<=x"6";
when b"001011"=> so<=x"f";
when b"001101"=> so<=x"0";
when b"001111"=> so<=x"3";
when b"010001"=> so<=x"4";
when b"010011"=> so<=x"7";
when b"010101"=> so<=x"2";
when b"010111"=> so<=x"c";
when b"011001"=> so<=x"1";
when b"011011"=> so<=x"a";
when b"011101"=> so<=x"e";
when b"011111"=> so<=x"9";
when b"100000"=> so<=x"a";
when b"100010"=> so<=x"6";
when b"100100"=> so<=x"9";
when b"100110"=> so<=x"0";
when b"101000"=> so<=x"c";
when b"101010"=> so<=x"b";
when b"101100"=> so<=x"7";
when b"101110"=> so<=x"d";
when b"110000"=> so<=x"f";
when b"110010"=> so<=x"1";
when b"110100"=> so<=x"3";
when b"110110"=> so<=x"e";
when b"111000"=> so<=x"5";
when b"111010"=> so<=x"2";
when b"111100"=> so<=x"8";
when b"111110"=> so<=x"4";
when b"100001"=> so<=x"3";
when b"100011"=> so<=x"f";
when b"100101"=> so<=x"0";
when b"100111"=> so<=x"6";
when b"101001"=> so<=x"a";
when b"101011"=> so<=x"1";
when b"101101"=> so<=x"d";
when b"101111"=> so<=x"8";
when b"110001"=> so<=x"9";
when b"110011"=> so<=x"4";
when b"110101"=> so<=x"5";
when b"110111"=> so<=x"b";
when b"111001"=> so<=x"c";
when b"111011"=> so<=x"7";
when b"111101"=> so<=x"2";
when others=> so<=x"e";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s5 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s5;
 architecture behaviour of s5 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"2";
when b"000010"=> so<=x"c";
```

```
when b"000100"=> so<=x"4";
when b"000110"=> so<=x"1";
when b"001000"=> so<=x"7";
when b"001010"=> so<=x"a";
when b"001100"=> so<=x"b";
when b"001110"=> so<=x"6";
when b"010000"=> so<=x"8";
when b"010010"=> so<=x"5";
when b"010100"=> so<=x"3";
when b"010110"=> so<=x"f";
when b"011000"=> so<=x"d";
when b"011010"=> so<=x"0";
when b"011100"=> so<=x"e";
when b"011110"=> so<=x"9";
when b"000001"=> so<=x"e";
when b"000011"=> so<=x"b";
when b"000101"=> so<=x"2";
when b"000111"=> so<=x"c";
when b"001001"=> so<=x"4";
when b"001011"=> so<=x"7";
when b"001101"=> so<=x"d";
when b"001111"=> so<=x"1";
when b"010001"=> so<=x"5";
when b"010011"=> so<=x"0";
when b"010101"=> so<=x"f";
when b"010111"=> so<=x"a";
when b"011001"=> so<=x"3";
when b"011011"=> so<=x"9";
when b"011101"=> so<=x"8";
when b"011111"=> so<=x"6";
when b"100000"=> so<=x"4";
when b"100010"=> so<=x"2";
when b"100100"=> so<=x"1";
when b"100110"=> so<=x"b";
when b"101000"=> so<=x"a";
when b"101010"=> so<=x"d";
when b"101100"=> so<=x"7";
when b"101110"=> so<=x"8";
when b"110000"=> so<=x"f";
when b"110010"=> so<=x"9";
when b"110100"=> so<=x"c";
when b"110110"=> so<=x"5";
when b"111000"=> so<=x"6";
when b"111010"=> so<=x"3";
when b"111100"=> so<=x"0";
when b"111110"=> so<=x"e";
when b"100001"=> so<=x"b";
when b"100011"=> so<=x"8";
when b"100101"=> so<=x"c";
when b"100111"=> so<=x"7";
when b"101001"=> so<=x"1";
when b"101011"=> so<=x"e";
when b"101101"=> so<=x"2";
when b"101111"=> so<=x"d";
when b"110001"=> so<=x"6";
when b"110011"=> so<=x"f";
when b"110101"=> so<=x"0";
when b"110111"=> so<=x"9";
when b"111001"=> so<=x"a";
when b"111011"=> so<=x"4";
when b"111101"=> so<=x"5";
when others=> so<=x"3";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s6 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s6;
 architecture behaviour of s6 is
begin
process(clk)
begin
if(clk'event and clk='1') then
```

```
case b is
when b"000000"=> so<=x"c";
when b"000010"=> so<=x"1";
when b"000100"=> so<=x"a";
when b"000110"=> so<=x"f";
when b"001000"=> so<=x"9";
when b"001010"=> so<=x"2";
when b"001100"=> so<=x"6";
when b"001110"=> so<=x"8";
when b"010000"=> so<=x"0";
when b"010010"=> so<=x"d";
when b"010100"=> so<=x"3";
when b"010110"=> so<=x"4";
when b"011000"=> so<=x"e";
when b"011010"=> so<=x"7";
when b"011100"=> so<=x"5";
when b"011110"=> so<=x"b";
when b"000001"=> so<=x"a";
when b"000011"=> so<=x"f";
when b"000101"=> so<=x"4";
when b"000111"=> so<=x"2";
when b"001001"=> so<=x"7";
when b"001011"=> so<=x"c";
when b"001101"=> so<=x"9";
when b"001111"=> so<=x"5";
when b"010001"=> so<=x"6";
when b"010011"=> so<=x"1";
when b"010101"=> so<=x"d";
when b"010111"=> so<=x"e";
when b"011001"=> so<=x"0";
when b"011011"=> so<=x"b";
when b"011101"=> so<=x"3";
when b"011111"=> so<=x"8";
when b"100000"=> so<=x"9";
when b"100010"=> so<=x"e";
when b"100100"=> so<=x"f";
when b"100110"=> so<=x"5";
when b"101000"=> so<=x"2";
when b"101010"=> so<=x"8";
when b"101100"=> so<=x"c";
when b"101110"=> so<=x"3";
when b"110000"=> so<=x"7";
when b"110010"=> so<=x"0";
when b"110100"=> so<=x"4";
when b"110110"=> so<=x"a";
when b"111000"=> so<=x"1";
when b"111010"=> so<=x"d";
when b"111100"=> so<=x"b";
when b"111110"=> so<=x"6";
when b"100001"=> so<=x"4";
when b"100011"=> so<=x"3";
when b"100101"=> so<=x"2";
when b"100111"=> so<=x"c";
when b"101001"=> so<=x"9";
when b"101011"=> so<=x"5";
when b"101101"=> so<=x"f";
when b"101111"=> so<=x"a";
when b"110001"=> so<=x"b";
when b"110011"=> so<=x"e";
when b"110101"=> so<=x"1";
when b"110111"=> so<=x"7";
when b"111001"=> so<=x"6";
when b"111011"=> so<=x"0";
when b"111101"=> so<=x"8";
when others=> so<=x"d";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s7 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s7;
 architecture behaviour of s7 is
begin
process(clk)
```

```
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"4";
when b"000010"=> so<=x"b";
when b"000100"=> so<=x"2";
when b"000110"=> so<=x"e";
when b"001000"=> so<=x"f";
when b"001010"=> so<=x"0";
when b"001100"=> so<=x"8";
when b"001110"=> so<=x"d";
when b"010000"=> so<=x"3";
when b"010010"=> so<=x"c";
when b"010100"=> so<=x"9";
when b"010110"=> so<=x"7";
when b"011000"=> so<=x"5";
when b"011010"=> so<=x"a";
when b"011100"=> so<=x"6";
when b"011110"=> so<=x"1";
when b"000001"=> so<=x"d";
when b"000011"=> so<=x"0";
when b"000101"=> so<=x"b";
when b"000111"=> so<=x"7";
when b"001001"=> so<=x"4";
when b"001011"=> so<=x"9";
when b"001101"=> so<=x"1";
when b"001111"=> so<=x"a";
when b"010001"=> so<=x"e";
when b"010011"=> so<=x"3";
when b"010101"=> so<=x"5";
when b"010111"=> so<=x"c";
when b"011001"=> so<=x"2";
when b"011011"=> so<=x"f";
when b"011101"=> so<=x"8";
when b"011111"=> so<=x"6";
when b"100000"=> so<=x"1";
when b"100010"=> so<=x"4";
when b"100100"=> so<=x"b";
when b"100110"=> so<=x"d";
when b"101000"=> so<=x"c";
when b"101010"=> so<=x"3";
when b"101100"=> so<=x"7";
when b"101110"=> so<=x"e";
when b"110000"=> so<=x"a";
when b"110010"=> so<=x"f";
when b"110100"=> so<=x"6";
when b"110110"=> so<=x"8";
when b"111000"=> so<=x"0";
when b"111010"=> so<=x"5";
when b"111100"=> so<=x"9";
when b"111110"=> so<=x"2";
when b"100001"=> so<=x"6";
when b"100011"=> so<=x"b";
when b"100101"=> so<=x"d";
when b"100111"=> so<=x"8";
when b"101001"=> so<=x"1";
when b"101011"=> so<=x"4";
when b"101101"=> so<=x"a";
when b"101111"=> so<=x"7";
when b"110001"=> so<=x"9";
when b"110011"=> so<=x"5";
when b"110101"=> so<=x"0";
when b"110111"=> so<=x"f";
when b"111001"=> so<=x"e";
when b"111011"=> so<=x"2";
when b"111101"=> so<=x"3";
when others=> so<=x"c";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s8 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s8;
```

```
 architecture behaviour of s8 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"d";
when b"000010"=> so<=x"2";
when b"000100"=> so<=x"8";
when b"000110"=> so<=x"4";
when b"001000"=> so<=x"6";
when b"001010"=> so<=x"f";
when b"001100"=> so<=x"b";
when b"001110"=> so<=x"1";
when b"010000"=> so<=x"a";
when b"010010"=> so<=x"9";
when b"010100"=> so<=x"3";
when b"010110"=> so<=x"e";
when b"011000"=> so<=x"5";
when b"011010"=> so<=x"0";
when b"011100"=> so<=x"c";
when b"011110"=> so<=x"7";
when b"000001"=> so<=x"1";
when b"000011"=> so<=x"f";
when b"000101"=> so<=x"d";
when b"000111"=> so<=x"8";
when b"001001"=> so<=x"a";
when b"001011"=> so<=x"3";
when b"001101"=> so<=x"7";
when b"001111"=> so<=x"4";
when b"010001"=> so<=x"c";
when b"010011"=> so<=x"5";
when b"010101"=> so<=x"6";
when b"010111"=> so<=x"b";
when b"011001"=> so<=x"0";
when b"011011"=> so<=x"e";
when b"011101"=> so<=x"9";
when b"011111"=> so<=x"2";
when b"100000"=> so<=x"7";
when b"100010"=> so<=x"b";
when b"100100"=> so<=x"4";
when b"100110"=> so<=x"1";
when b"101000"=> so<=x"9";
when b"101010"=> so<=x"c";
when b"101100"=> so<=x"e";
when b"101110"=> so<=x"2";
when b"110000"=> so<=x"0";
when b"110010"=> so<=x"6";
when b"110100"=> so<=x"a";
when b"110110"=> so<=x"d";
when b"111000"=> so<=x"f";
when b"111010"=> so<=x"3";
when b"111100"=> so<=x"5";
when b"111110"=> so<=x"8";
when b"100001"=> so<=x"2";
when b"100011"=> so<=x"1";
when b"100101"=> so<=x"e";
when b"100111"=> so<=x"7";
when b"101001"=> so<=x"4";
when b"101011"=> so<=x"a";
when b"101101"=> so<=x"8";
when b"101111"=> so<=x"d";
when b"110001"=> so<=x"f";
when b"110011"=> so<=x"c";
when b"110101"=> so<=x"9";
when b"110111"=> so<=x"0";
when b"111001"=> so<=x"3";
when b"111011"=> so<=x"5";
when b"111101"=> so<=x"6";
when others=> so<=x"b";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity pp is port
(
so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
```

```
: in std_logic_vector(1 to 4);
ppo : out std_logic_vector(1 to 32)
);
end pp;

architecture behaviour of pp is
signal XX : std_logic_vector(1 to 32);
begin
XX(1 to 4)<=so1x; XX(5 to 8)<=so2x; XX(9 to 12)<=so3x; XX(13 to 16)<=so4x;
XX(17 to 20)<=so5x; XX(21 to 24)<=so6x; XX(25 to 28)<=so7x; XX(29 to 32)<=so8x;

ppo(1)<=XX(16); ppo(2)<=XX(7); ppo(3)<=XX(20); ppo(4)<=XX(21);
ppo(5)<=XX(29); ppo(6)<=XX(12); ppo(7)<=XX(28); ppo(8)<=XX(17);
ppo(9)<=XX(1); ppo(10)<=XX(15); ppo(11)<=XX(23); ppo(12)<=XX(26);
ppo(13)<=XX(5); ppo(14)<=XX(18); ppo(15)<=XX(31); ppo(16)<=XX(10);
ppo(17)<=XX(2); ppo(18)<=XX(8); ppo(19)<=XX(24); ppo(20)<=XX(14);
ppo(21)<=XX(32); ppo(22)<=XX(27); ppo(23)<=XX(3); ppo(24)<=XX(9);
ppo(25)<=XX(19); ppo(26)<=XX(13); ppo(27)<=XX(30); ppo(28)<=XX(6);
ppo(29)<=XX(22); ppo(30)<=XX(11); ppo(31)<=XX(4); ppo(32)<=XX(25);
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity desxor2 is port
(
d,l : in std_logic_vector(1 to 32);
q : out std_logic_vector(1 to 32)
);
end desxor2;

architecture behaviour of desxor2 is
begin
q<=d xor l;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity roundfunc is port
(
clk : in std_logic;
li,ri : in std_logic_vector(1 to 32);
k : in std_logic_vector(1 to 48);
lo,ro : out std_logic_vector(1 to 32)
);
end roundfunc;

architecture behaviour of roundfunc is

component s1 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s2 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s3 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s4 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;
```

```
component s5 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s6 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s7 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s8 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component pp port
(
so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
: in std_logic_vector(1 to 4);
ppo : out std_logic_vector(1 to 32)
);
end component;

component desxor2 port
(
d,l : in std_logic_vector(1 to 32);
q : out std_logic_vector(1 to 32)
);
end component;

component desxor1 port
(
e : in std_logic_vector(1 TO 48);
b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
: out std_logic_vector (1 TO 6);
k : in std_logic_vector (1 TO 48)
);
end component;

component xp port
(
ri : in std_logic_vector(1 TO 32);
e : out std_logic_vector(1 TO 48));
end component;

signal e : std_logic_vector(1 to 48);
signal b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
: std_logic_vector(1 to 6);
signal so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
: std_logic_vector(1 to 4);
signal ppo : std_logic_vector(1 to 32);
begin

uxp: xp port map ( ri=>ri, e=>e );

udesxor1: desxor1 port map ( e=>e, k=>k, b1x=>b1x, b2x=>b2x, b3x=>b3x, b4x=>b4x, b5x=>b5x,
b6x=>b6x, b7x=>b7x, b8x=>b8x );

us1: s1 port map ( clk=>clk, b=>b1x, so=>so1x );
us2: s2 port map ( clk=>clk, b=>b2x, so=>so2x );
us3: s3 port map ( clk=>clk, b=>b3x, so=>so3x );
us4: s4 port map ( clk=>clk, b=>b4x, so=>so4x );
us5: s5 port map ( clk=>clk, b=>b5x, so=>so5x );
```

```
us6: s6 port map ( clk=>clk, b=>b6x, so=>so6x );
us7: s7 port map ( clk=>clk, b=>b7x, so=>so7x );
us8: s8 port map ( clk=>clk, b=>b8x, so=>so8x );

upp: pp port map ( so1x=>so1x, so2x=>so2x, so3x=>so3x, so4x=>so4x, so5x=>so5x, so6x=>so6x,
so7x=>so7x, so8x=>so8x, ppo=>ppo );

udesxor2: desxor2 port map ( d=>ppo, l=>li, q=>ro );

lo<=ri;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity fp is port
(
l,r : in std_logic_vector(1 to 32);
ct : out std_logic_vector(1 to 64)
);
end fp;

architecture behaviour of fp is
begin
ct(1)<=r(8); ct(2)<=l(8); ct(3)<=r(16); ct(4)<=l(16); ct(5)<=r(24); ct(6)<=l(24); ct(7)<=r(32); ct(8)<=l(32);
ct(9)<=r(7); ct(10)<=l(7); ct(11)<=r(15); ct(12)<=l(15); ct(13)<=r(23); ct(14)<=l(23); ct(15)<=r(31); ct(16)<=l(31);
ct(17)<=r(6); ct(18)<=l(6); ct(19)<=r(14); ct(20)<=l(14); ct(21)<=r(22); ct(22)<=l(22); ct(23)<=r(30); ct(24)<=l(30);
ct(25)<=r(5); ct(26)<=l(5); ct(27)<=r(13); ct(28)<=l(13); ct(29)<=r(21); ct(30)<=l(21); ct(31)<=r(29); ct(32)<=l(29);
ct(33)<=r(4); ct(34)<=l(4); ct(35)<=r(12); ct(36)<=l(12); ct(37)<=r(20); ct(38)<=l(20); ct(39)<=r(28); ct(40)<=l(28);
ct(41)<=r(3); ct(42)<=l(3); ct(43)<=r(11); ct(44)<=l(11); ct(45)<=r(19); ct(46)<=l(19); ct(47)<=r(27); ct(48)<=l(27);
ct(49)<=r(2); ct(50)<=l(2); ct(51)<=r(10); ct(52)<=l(10); ct(53)<=r(18); ct(54)<=l(18); ct(55)<=r(26); ct(56)<=l(26);
ct(57)<=r(1); ct(58)<=l(1); ct(59)<=r(9); ct(60)<=l(9); ct(61)<=r(17); ct(62)<=l(17); ct(63)<=r(25); ct(64)<=l(25);
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity desenc is port
(
pt : in std_logic_vector(1 TO 64);
key : in std_logic_vector(1 TO 64);
ct : out std_logic_vector(1 TO 64);
clk : in std_logic
);
end desenc;

architecture behavior of desenc is

component keysched port
(
key : in std_logic_vector(1 to 64);
k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
: out std_logic_vector(1 to 48)
);
end component;

component roundfunc port
(
clk : in std_logic;
li,ri : in std_logic_vector(1 to 32);
k : in std_logic_vector(1 to 48);
lo,ro : out std_logic_vector(1 to 32)
);
end component;

component ip port
(
pt : in std_logic_vector(1 TO 64);
l0x : out std_logic_vector(1 TO 32);
r0x : out std_logic_vector(1 TO 32)
);
end component;

component fp port
(
l,r : in std_logic_vector(1 to 32);
ct : out std_logic_vector(1 to 64)
);
end component;
```

```
signal k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x : std_logic_vector(1 to 48);
signal l0x,l1x,l2x,l3x,l4x,l5x,l6x,l7x,l8x,l9x,l10x,l11x,l12x,l13x,l14x,l15x,l16x : std_logic_vector(1 to 32);
signal r0x,r1x,r2x,r3x,r4x,r5x,r6x,r7x,r8x,r9x,r10x,r11x,r12x,r13x,r14x,r15x,r16x : std_logic_vector(1 to 32);
begin
ukeysched: keysched port map ( key=>key, k1x=>k1x, k2x=>k2x, k3x=>k3x, k4x=>k4x, k5x=>k5x, k6x=>k6x,
k7x=>k7x, k8x=>k8x, k9x=>k9x, k10x=>k10x, k11x=>k11x, k12x=>k12x, k13x=>k13x,
k14x=>k14x, k15x=>k15x, k16x=>k16x );

uip: ip  port map ( pt=>pt, l0x=>l0x, r0x=>r0x );

round1: roundfunc port map ( clk=>clk, li=>l0x, ri=>r0x,   lo=>l1x, ro=>r1x, k=>k1x );
round2: roundfunc port map ( clk=>clk, li=>l1x, ri=>r1x,   lo=>l2x, ro=>r2x, k=>k2x );
round3: roundfunc port map ( clk=>clk, li=>l2x, ri=>r2x,   lo=>l3x, ro=>r3x, k=>k3x );
round4: roundfunc port map ( clk=>clk, li=>l3x, ri=>r3x,   lo=>l4x, ro=>r4x, k=>k4x );
round5: roundfunc port map ( clk=>clk, li=>l4x, ri=>r4x,   lo=>l5x, ro=>r5x, k=>k5x );
round6: roundfunc port map ( clk=>clk, li=>l5x, ri=>r5x,   lo=>l6x, ro=>r6x, k=>k6x );
round7: roundfunc port map ( clk=>clk, li=>l6x, ri=>r6x,   lo=>l7x, ro=>r7x, k=>k7x );
round8: roundfunc port map ( clk=>clk, li=>l7x, ri=>r7x,   lo=>l8x, ro=>r8x, k=>k8x );
round9: roundfunc port map ( clk=>clk, li=>l8x, ri=>r8x,   lo=>l9x, ro=>r9x, k=>k9x );
round10: roundfunc port map ( clk=>clk, li=>l9x, ri=>r9x,   lo=>l10x, ro=>r10x, k=>k10x );
round11: roundfunc port map ( clk=>clk, li=>l10x, ri=>r10x,   lo=>l11x, ro=>r11x, k=>k11x );
round12: roundfunc port map ( clk=>clk, li=>l11x, ri=>r11x,   lo=>l12x, ro=>r12x, k=>k12x );
round13: roundfunc port map ( clk=>clk, li=>l12x, ri=>r12x,   lo=>l13x, ro=>r13x, k=>k13x );
round14: roundfunc port map ( clk=>clk, li=>l13x, ri=>r13x,   lo=>l14x, ro=>r14x, k=>k14x );
round15: roundfunc port map ( clk=>clk, li=>l14x, ri=>r14x,   lo=>l15x, ro=>r15x, k=>k15x );
round16: roundfunc port map ( clk=>clk, li=>l15x, ri=>r15x,   lo=>l16x, ro=>r16x, k=>k16x );

ufp: fp port map ( l=>r16x, r=>l16x, ct=>ct );

end behavior;
```

A.2 test_desenc.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test_desenc is
-- Test bench has no external interface

end test_desenc;

architecture testbench of test_desenc is

component desenc port
(
pt : in std_logic_vector(1 to 64);
key : in std_logic_vector(1 to 64);
ct : out std_logic_vector(1 to 64);
clk : in std_logic
);
end component;

type test_vector is record
key : std_logic_vector(1 to 64);
pt : std_logic_vector(1 to 64);
ct : std_logic_vector(1 to 64);
end record;

type test_vector_array is array(natural range <>) of test_vector;

constant test_vectors: test_vector_array :=(
( key=>x"0000000000000000", pt=>x"0000000000000000", ct=>x"8ca64de9c1b123a7" ),
   ( key=>x"ffffffffffffffff", pt=>x"ffffffffffffffff", ct=>x"7359b2163e4edc58" ),
   ( key=>x"3000000000000000", pt=>x"1000000000000001", ct=>x"958e6e627a05557b" ),
   ( key=>x"1111111111111111", pt=>x"1111111111111111", ct=>x"f40379ab9e0ec533" ),
   ( key=>x"0123456789abcdef", pt=>x"1111111111111111", ct=>x"17668dfc7292532d" ),
   ( key=>x"1111111111111111", pt=>x"0123456789abcdef", ct=>x"8a5ae1f81ab8f2dd" ),
( key=>x"0000000000000000", pt=>x"0000000000000000", ct=>x"8ca64de9c1b123a7" ),
   ( key=>x"fedcba9876543210", pt=>x"0123456789abcdef", ct=>x"ed39d950fa74bcc4" ),
   ( key=>x"7ca110454a1a6e57", pt=>x"01a1d6d039776742", ct=>x"690f5b0d9a26939b" ),
   ( key=>x"0131d9619dc1376e", pt=>x"5cd54ca83def57da", ct=>x"7a389d10354bd271" ),
   ( key=>x"07a1133e4a0b2686", pt=>x"0248d43806f67172", ct=>x"868ebb51cab4599a" ),
   ( key=>x"3849674c2602319e", pt=>x"51454b582ddf440a", ct=>x"7178876e01f19b2a" ),
   ( key=>x"04b915ba43feb5b6", pt=>x"42fd443059577fa2", ct=>x"af37fb421f8c4095" ),
   ( key=>x"0113b970fd34f2ce", pt=>x"059b5e0851cf143a", ct=>x"86a560f10ec6d85b" ),
   ( key=>x"0170f175468fb5e6", pt=>x"0756d8e0774761d2", ct=>x"0cd3da020021dc09" ),
```

```
   ( key=>x"43297fad38e373fe", pt=>x"762514b829bf486a", ct=>x"ea676b2cb7db2b7a" ),
   ( key=>x"07a7137045da2a16", pt=>x"3bdd119049372802", ct=>x"dfd64a815caf1a0f" ),
   ( key=>x"04689104c2fd3b2f", pt=>x"26955f6835af609a", ct=>x"5c513c9c4886c088" ),
   ( key=>x"37d06bb516cb7546", pt=>x"164d5e404f275232", ct=>x"0a2aeeae3ff4ab77" ),
   ( key=>x"1f08260d1ac2465e", pt=>x"6b056e18759f5cca", ct=>x"ef1bf03e5dfa575a" ),
   ( key=>x"584023641aba6176", pt=>x"004bd6ef09176062", ct=>x"88bf0db6d70dee56" ),
   ( key=>x"025816164629b007", pt=>x"480d39006ee762f2", ct=>x"a1f9915541020b56" ),
   ( key=>x"49793ebc79b3258f", pt=>x"437540c8698f3cfa", ct=>x"6fbf1cafcffd0556" ),
   ( key=>x"4fb05e1515ab73a7", pt=>x"072d43a077075292", ct=>x"2f22e49bab7ca1ac" ),
   ( key=>x"49e95d6d4ca229bf", pt=>x"02fe55778117f12a", ct=>x"5a6b612cc26cce4a" ),
   ( key=>x"018310dc409b26d6", pt=>x"1d9d5c5018f728c2", ct=>x"5f4c038ed12b2e41" ),
   ( key=>x"1c587f1c13924fef", pt=>x"305532286d6f295a", ct=>x"63fac0d034d9f793" ),
   ( key=>x"0101010101010101", pt=>x"0123456789abcdef", ct=>x"617b3a0ce8f07100" ),
   ( key=>x"1f1f1f1f0e0e0e0e", pt=>x"0123456789abcdef", ct=>x"db958605f8c8c606" ),
   ( key=>x"e0fee0fef1fef1fe", pt=>x"0123456789abcdef", ct=>x"edbfd1c66c29ccc7" ),
   ( key=>x"0000000000000000", pt=>x"ffffffffffffffff", ct=>x"355550b2150e2451" ),
   ( key=>x"ffffffffffffffff", pt=>x"0000000000000000", ct=>x"caaaaf4deaf1dbae" ),
( key=>x"0123456789abcdef", pt=>x"0000000000000000", ct=>x"d5d44ff720683d0d" ),
( key=>x"fedcba9876543210", pt=>x"ffffffffffffffff", ct=>x"2a2bb008df97c2f2" )
);

signal   key : std_logic_vector(1 to 64);
signal   pt : std_logic_vector(1 to 64);
signal   ct : std_logic_vector(1 to 64);
signal   clk : std_logic;

begin

  DUT: desenc
   port map (
   pt=>pt,
   key=>key,
   ct=>ct,
   clk=>clk
   );

process
variable vector : test_vector;
variable errors : boolean:=false;
begin
for i in test_vectors'range loop
vector:=test_vectors(i);
key<=vector.key; pt<=vector.pt;

for j in 0 to 15 loop clk<='0'; wait for 250 ns; clk<='1'; wait for 250 ns; end loop;

if(ct/=vector.ct) then
assert false
report "Implementation Failure"
severity note;
errors:=true;
end if;
end loop;

    assert not errors
report "Test vectors failed"
severity note;
assert errors
report "Test vectors passed"
severity note;
wait;

end process;

end testbench;
```

A.3 test_desenc_testable.vhd

```
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test_desenc_testable is
-- Test bench has no external interface
```

```
end test_desenc_testable;

architecture testbench of test_desenc_testable is

component desenc_testable port
(
pt : in std_logic_vector(1 to 64);
key : in std_logic_vector(1 to 64);
ct : out std_logic_vector(1 to 64);
clk : in std_logic;
test_se : in STD_LOGIC;
        test_si1 : in STD_LOGIC;
        test_so1 : out STD_LOGIC;
        test_si2 : in STD_LOGIC;
        test_so2 : out STD_LOGIC

);
end component;

type test_vector is record
key : std_logic_vector(1 to 64);
pt : std_logic_vector(1 to 64);
ct : std_logic_vector(1 to 64);
end record;

type test_vector_array is array(natural range <>) of test_vector;

constant test_vectors: test_vector_array :=(
( key=>x"0000000000000000", pt=>x"0000000000000000", ct=>x"8ca64de9c1b123a7" ),
   ( key=>x"ffffffffffffffff", pt=>x"ffffffffffffffff", ct=>x"7359b2163e4edc58" ),
   ( key=>x"3000000000000000", pt=>x"1000000000000001", ct=>x"958e6e627a05557b" ),
   ( key=>x"1111111111111111", pt=>x"1111111111111111", ct=>x"f40379ab9e0ec533" ),
   ( key=>x"0123456789abcdef", pt=>x"1111111111111111", ct=>x"17668dfc7292532d" ),
   ( key=>x"1111111111111111", pt=>x"0123456789abcdef", ct=>x"8a5ae1f81ab8f2dd" ),
( key=>x"0000000000000000", pt=>x"0000000000000000", ct=>x"8ca64de9c1b123a7" ),
   ( key=>x"fedcba9876543210", pt=>x"0123456789abcdef", ct=>x"ed39d950fa74bcc4" ),
   ( key=>x"7ca110454a1a6e57", pt=>x"01a1d6d039776742", ct=>x"690f5b0d9a26939b" ),
   ( key=>x"0131d9619dc1376e", pt=>x"5cd54ca83def57da", ct=>x"7a389d10354bd271" ),
   ( key=>x"07a1133e4a0b2686", pt=>x"0248d43806f67172", ct=>x"868ebb51cab4599a" ),
   ( key=>x"3849674c2602319e", pt=>x"51454b582ddf440a", ct=>x"7178876e01f19b2a" ),
   ( key=>x"04b915ba43feb5b6", pt=>x"42fd443059577fa2", ct=>x"af37fb421f8c4095" ),
   ( key=>x"0113b970fd34f2ce", pt=>x"059b5e0851cf143a", ct=>x"86a560f10ec6d85b" ),
   ( key=>x"0170f175468fb5e6", pt=>x"0756d8e0774761d2", ct=>x"0cd3da020021dc09" ),
   ( key=>x"43297fad38e373fe", pt=>x"762514b829bf486a", ct=>x"ea676b2cb7db2b7a" ),
   ( key=>x"07a7137045da2a16", pt=>x"3bdd119049372802", ct=>x"dfd64a815caf1a0f" ),
   ( key=>x"04689104c2fd3b2f", pt=>x"26955f6835af609a", ct=>x"5c513c9c4886c088" ),
   ( key=>x"37d06bb516cb7546", pt=>x"164d5e404f275232", ct=>x"0a2aeeae3ff4ab77" ),
   ( key=>x"1f08260d1ac2465e", pt=>x"6b056e18759f5cca", ct=>x"ef1bf03e5dfa575a" ),
   ( key=>x"584023641aba6176", pt=>x"004bd6ef09176062", ct=>x"88bf0db6d70dee56" ),
   ( key=>x"025816164629b007", pt=>x"480d39006ee762f2", ct=>x"a1f9915541020b56" ),
   ( key=>x"49793ebc79b3258f", pt=>x"437540c8698f3cfa", ct=>x"6fbf1cafcffd0556" ),
   ( key=>x"4fb05e1515ab73a7", pt=>x"072d43a077075292", ct=>x"2f22e49bab7ca1ac" ),
   ( key=>x"49e95d6d4ca229bf", pt=>x"02fe55778117f12a", ct=>x"5a6b612cc26cce4a" ),
   ( key=>x"018310dc409b26d6", pt=>x"1d9d5c5018f728c2", ct=>x"5f4c038ed12b2e41" ),
   ( key=>x"1c587f1c13924fef", pt=>x"305532286d6f295a", ct=>x"63fac0d034d9f793" ),
   ( key=>x"0101010101010101", pt=>x"0123456789abcdef", ct=>x"617b3a0ce8f07100" ),
   ( key=>x"1f1f1f1f0e0e0e0e", pt=>x"0123456789abcdef", ct=>x"db958605f8c8c606" ),
   ( key=>x"e0fee0fef1fef1fe", pt=>x"0123456789abcdef", ct=>x"edbfd1c66c29ccc7" ),
   ( key=>x"0000000000000000", pt=>x"ffffffffffffffff", ct=>x"355550b2150e2451" ),
   ( key=>x"ffffffffffffffff", pt=>x"0000000000000000", ct=>x"caaaaf4deaf1dbae" ),
( key=>x"0123456789abcdef", pt=>x"0000000000000000", ct=>x"d5d44ff720683d0d" ),
( key=>x"fedcba9876543210", pt=>x"ffffffffffffffff", ct=>x"2a2bb008df97c2f2" )
);

signal   key : std_logic_vector(1 to 64);
signal   pt : std_logic_vector(1 to 64);
signal   ct : std_logic_vector(1 to 64);
signal   clk : std_logic;
signal   test_se :  STD_LOGIC;     -- added by Fu 6/23/02
        signal   test_si1 :  STD_LOGIC;     -- added by Fu 6/23/02
        signal   test_so1 :  STD_LOGIC;     -- added by Fu 6/23/02
        signal   test_si2 :  STD_LOGIC;     -- added by Fu 7/22/02
        signal   test_so2 :  STD_LOGIC;

begin

  DUT: desenc_testable
   port map (
   pt=>pt,
   key=>key,
```

```
   ct=>ct,
   clk=>clk,
   test_se=>test_se,    -- added by Fu 6/23/02
   test_si1=>test_si1,    -- added by Fu 7/22/02
   test_so1=>test_so1,     -- added by Fu 7/22/02
               test_si2=>test_si2,    -- added by Fu 7/22/02
   test_so2=>test_so2     -- added by Fu 7/22/02
   );

process
variable vector : test_vector;
variable errors : boolean:=false;
begin

        test_se <= '0';   -- added by Fu 6/23/02
                test_si1 <= '1';   -- added by Fu
                test_si2 <= '1';   -- added by Fu


for i in test_vectors'range loop
vector:=test_vectors(i);
key<=vector.key; pt<=vector.pt;

for j in 0 to 15 loop clk<='0'; wait for 250 ns; clk<='1'; wait for 250 ns; end loop;

if(ct/=vector.ct) then
assert false
report "Implementation Failure"
severity note;
errors:=true;
end if;
end loop;

    assert not errors
report "Test vectors failed"
severity note;
assert errors
report "Test vectors passed"
severity note;
wait;

end process;

end testbench;
```

A.4 roundfunc.vhd

```
LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity pp is port
(
so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
: in std_logic_vector(1 to 4);
ppo : out std_logic_vector(1 to 32)
);
end pp;

architecture behaviour of pp is
signal XX : std_logic_vector(1 to 32);
begin
XX(1 to 4)<=so1x; XX(5 to 8)<=so2x; XX(9 to 12)<=so3x; XX(13 to 16)<=so4x;
XX(17 to 20)<=so5x; XX(21 to 24)<=so6x; XX(25 to 28)<=so7x; XX(29 to 32)<=so8x;

ppo(1)<=XX(16); ppo(2)<=XX(7); ppo(3)<=XX(20); ppo(4)<=XX(21);
ppo(5)<=XX(29); ppo(6)<=XX(12); ppo(7)<=XX(28); ppo(8)<=XX(17);
ppo(9)<=XX(1); ppo(10)<=XX(15); ppo(11)<=XX(23); ppo(12)<=XX(26);
ppo(13)<=XX(5); ppo(14)<=XX(18); ppo(15)<=XX(31); ppo(16)<=XX(10);
ppo(17)<=XX(2); ppo(18)<=XX(8); ppo(19)<=XX(24); ppo(20)<=XX(14);
ppo(21)<=XX(32); ppo(22)<=XX(27); ppo(23)<=XX(3); ppo(24)<=XX(9);
ppo(25)<=XX(19); ppo(26)<=XX(13); ppo(27)<=XX(30); ppo(28)<=XX(6);
ppo(29)<=XX(22); ppo(30)<=XX(11); ppo(31)<=XX(4); ppo(32)<=XX(25);
end;

library ieee;
use ieee.std_logic_1164.all;
 entity desxor1 is port
```

```
(
e : in std_logic_vector(1 TO 48);
b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
: out std_logic_vector (1 TO 6);
k : in std_logic_vector (1 TO 48)
);
end desxor1;
 architecture behavior of desxor1 is
signal XX : std_logic_vector( 1 to 48);
begin
XX<=k xor e;
b1x<=XX(1 to 6);
b2x<=XX(7 to 12);
b3x<=XX(13 to 18);
b4x<=XX(19 to 24);
b5x<=XX(25 to 30);
b6x<=XX(31 to 36);
b7x<=XX(37 to 42);
b8x<=XX(43 to 48);
end behavior;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity desxor2 is port
(
d,l : in std_logic_vector(1 to 32);
q : out std_logic_vector(1 to 32)
);
end desxor2;

architecture behaviour of desxor2 is
begin
q<=d xor l;
end;

library ieee;

use ieee.std_logic_1164.all;
 entity xp is port
(
ri : in std_logic_vector(1 TO 32);
e : out std_logic_vector(1 TO 48));
end xp;
 architecture behavior of xp is
begin
e(1)<=ri(32); e(2)<=ri(1); e(3)<=ri(2); e(4)<=ri(3); e(5)<=ri(4); e(6)<=ri(5); e(7)<=ri(4); e(8)<=ri(5);
e(9)<=ri(6); e(10)<=ri(7); e(11)<=ri(8); e(12)<=ri(9); e(13)<=ri(8); e(14)<=ri(9); e(15)<=ri(10); e(16)<=ri(11);
e(17)<=ri(12); e(18)<=ri(13); e(19)<=ri(12); e(20)<=ri(13); e(21)<=ri(14); e(22)<=ri(15); e(23)<=ri(16); e(24)<=ri(17);
e(25)<=ri(16); e(26)<=ri(17); e(27)<=ri(18); e(28)<=ri(19); e(29)<=ri(20); e(30)<=ri(21); e(31)<=ri(20); e(32)<=ri(21);
e(33)<=ri(22); e(34)<=ri(23); e(35)<=ri(24); e(36)<=ri(25); e(37)<=ri(24); e(38)<=ri(25); e(39)<=ri(26); e(40)<=ri(27);
e(41)<=ri(28); e(42)<=ri(29); e(43)<=ri(28); e(44)<=ri(29); e(45)<=ri(30); e(46)<=ri(31); e(47)<=ri(32); e(48)<=ri(1);
end behavior;

library ieee;
 use ieee.std_logic_1164.all;
 entity s1 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s1;
 architecture behaviour of s1 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"e";
when b"000010"=> so<=x"4";
when b"000100"=> so<=x"d";
when b"000110"=> so<=x"1";
when b"001000"=> so<=x"2";
when b"001010"=> so<=x"f";
when b"001100"=> so<=x"b";
when b"001110"=> so<=x"8";
when b"010000"=> so<=x"3";
when b"010010"=> so<=x"a";
when b"010100"=> so<=x"6";
when b"010110"=> so<=x"c";
```

```
when b"011000"=> so<=x"5";
when b"011010"=> so<=x"9";
when b"011100"=> so<=x"0";
when b"011110"=> so<=x"7";
when b"000001"=> so<=x"0";
when b"000011"=> so<=x"f";
when b"000101"=> so<=x"7";
when b"000111"=> so<=x"4";
when b"001001"=> so<=x"e";
when b"001011"=> so<=x"2";
when b"001101"=> so<=x"d";
when b"001111"=> so<=x"1";
when b"010001"=> so<=x"a";
when b"010011"=> so<=x"6";
when b"010101"=> so<=x"c";
when b"010111"=> so<=x"b";
when b"011001"=> so<=x"9";
when b"011011"=> so<=x"5";
when b"011101"=> so<=x"3";
when b"011111"=> so<=x"8";
when b"100000"=> so<=x"4";
when b"100010"=> so<=x"1";
when b"100100"=> so<=x"e";
when b"100110"=> so<=x"8";
when b"101000"=> so<=x"d";
when b"101010"=> so<=x"6";
when b"101100"=> so<=x"2";
when b"101110"=> so<=x"b";
when b"110000"=> so<=x"f";
when b"110010"=> so<=x"c";
when b"110100"=> so<=x"9";
when b"110110"=> so<=x"7";
when b"111000"=> so<=x"3";
when b"111010"=> so<=x"a";
when b"111100"=> so<=x"5";
when b"111110"=> so<=x"0";
when b"100001"=> so<=x"f";
when b"100011"=> so<=x"c";
when b"100101"=> so<=x"8";
when b"100111"=> so<=x"2";
when b"101001"=> so<=x"4";
when b"101011"=> so<=x"9";
when b"101101"=> so<=x"1";
when b"101111"=> so<=x"7";
when b"110001"=> so<=x"5";
when b"110011"=> so<=x"b";
when b"110101"=> so<=x"3";
when b"110111"=> so<=x"e";
when b"111001"=> so<=x"a";
when b"111011"=> so<=x"0";
when b"111101"=> so<=x"6";
when others=> so<=x"d";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s2 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s2;
 architecture behaviour of s2 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"f";
when b"000010"=> so<=x"1";
when b"000100"=> so<=x"8";
when b"000110"=> so<=x"e";
when b"001000"=> so<=x"6";
when b"001010"=> so<=x"b";
when b"001100"=> so<=x"3";
when b"001110"=> so<=x"4";
when b"010000"=> so<=x"9";
when b"010010"=> so<=x"7";
```

```
when b"010100"=> so<=x"2";
when b"010110"=> so<=x"d";
when b"011000"=> so<=x"c";
when b"011010"=> so<=x"0";
when b"011100"=> so<=x"5";
when b"011110"=> so<=x"a";
when b"000001"=> so<=x"3";
when b"000011"=> so<=x"d";
when b"000101"=> so<=x"4";
when b"000111"=> so<=x"7";
when b"001001"=> so<=x"f";
when b"001011"=> so<=x"2";
when b"001101"=> so<=x"8";
when b"001111"=> so<=x"e";
when b"010001"=> so<=x"c";
when b"010011"=> so<=x"0";
when b"010101"=> so<=x"1";
when b"010111"=> so<=x"a";
when b"011001"=> so<=x"6";
when b"011011"=> so<=x"9";
when b"011101"=> so<=x"b";
when b"011111"=> so<=x"5";
when b"100000"=> so<=x"0";
when b"100010"=> so<=x"e";
when b"100100"=> so<=x"7";
when b"100110"=> so<=x"b";
when b"101000"=> so<=x"a";
when b"101010"=> so<=x"4";
when b"101100"=> so<=x"d";
when b"101110"=> so<=x"1";
when b"110000"=> so<=x"5";
when b"110010"=> so<=x"8";
when b"110100"=> so<=x"c";
when b"110110"=> so<=x"6";
when b"111000"=> so<=x"9";
when b"111010"=> so<=x"3";
when b"111100"=> so<=x"2";
when b"111110"=> so<=x"f";
when b"100001"=> so<=x"d";
when b"100011"=> so<=x"8";
when b"100101"=> so<=x"a";
when b"100111"=> so<=x"1";
when b"101001"=> so<=x"3";
when b"101011"=> so<=x"f";
when b"101101"=> so<=x"4";
when b"101111"=> so<=x"2";
when b"110001"=> so<=x"b";
when b"110011"=> so<=x"6";
when b"110101"=> so<=x"7";
when b"110111"=> so<=x"c";
when b"111001"=> so<=x"0";
when b"111011"=> so<=x"5";
when b"111101"=> so<=x"e";
when others=> so<=x"9";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s3 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s3;
 architecture behaviour of s3 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"a";
when b"000010"=> so<=x"0";
when b"000100"=> so<=x"9";
when b"000110"=> so<=x"e";
when b"001000"=> so<=x"6";
when b"001010"=> so<=x"3";
when b"001100"=> so<=x"f";
```

```
when b"001110"=> so<=x"5";
when b"010000"=> so<=x"1";
when b"010010"=> so<=x"d";
when b"010100"=> so<=x"c";
when b"010110"=> so<=x"7";
when b"011000"=> so<=x"b";
when b"011010"=> so<=x"4";
when b"011100"=> so<=x"2";
when b"011110"=> so<=x"8";
when b"000001"=> so<=x"d";
when b"000011"=> so<=x"7";
when b"000101"=> so<=x"0";
when b"000111"=> so<=x"9";
when b"001001"=> so<=x"3";
when b"001011"=> so<=x"4";
when b"001101"=> so<=x"6";
when b"001111"=> so<=x"a";
when b"010001"=> so<=x"2";
when b"010011"=> so<=x"8";
when b"010101"=> so<=x"5";
when b"010111"=> so<=x"e";
when b"011001"=> so<=x"c";
when b"011011"=> so<=x"b";
when b"011101"=> so<=x"f";
when b"011111"=> so<=x"1";
when b"100000"=> so<=x"d";
when b"100010"=> so<=x"6";
when b"100100"=> so<=x"4";
when b"100110"=> so<=x"9";
when b"101000"=> so<=x"8";
when b"101010"=> so<=x"f";
when b"101100"=> so<=x"3";
when b"101110"=> so<=x"0";
when b"110000"=> so<=x"b";
when b"110010"=> so<=x"1";
when b"110100"=> so<=x"2";
when b"110110"=> so<=x"c";
when b"111000"=> so<=x"5";
when b"111010"=> so<=x"a";
when b"111100"=> so<=x"e";
when b"111110"=> so<=x"7";
when b"100001"=> so<=x"1";
when b"100011"=> so<=x"a";
when b"100101"=> so<=x"d";
when b"100111"=> so<=x"0";
when b"101001"=> so<=x"6";
when b"101011"=> so<=x"9";
when b"101101"=> so<=x"8";
when b"101111"=> so<=x"7";
when b"110001"=> so<=x"4";
when b"110011"=> so<=x"f";
when b"110101"=> so<=x"e";
when b"110111"=> so<=x"3";
when b"111001"=> so<=x"b";
when b"111011"=> so<=x"5";
when b"111101"=> so<=x"2";
when others=> so<=x"c";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s4 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s4;
 architecture behaviour of s4 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"7";
when b"000010"=> so<=x"d";
when b"000100"=> so<=x"e";
when b"000110"=> so<=x"3";
```

```
when b"001000"=> so<=x"0";
when b"001010"=> so<=x"6";
when b"001100"=> so<=x"9";
when b"001110"=> so<=x"a";
when b"010000"=> so<=x"1";
when b"010010"=> so<=x"2";
when b"010100"=> so<=x"8";
when b"010110"=> so<=x"5";
when b"011000"=> so<=x"b";
when b"011010"=> so<=x"c";
when b"011100"=> so<=x"4";
when b"011110"=> so<=x"f";
when b"000001"=> so<=x"d";
when b"000011"=> so<=x"8";
when b"000101"=> so<=x"b";
when b"000111"=> so<=x"5";
when b"001001"=> so<=x"6";
when b"001011"=> so<=x"f";
when b"001101"=> so<=x"0";
when b"001111"=> so<=x"3";
when b"010001"=> so<=x"4";
when b"010011"=> so<=x"7";
when b"010101"=> so<=x"2";
when b"010111"=> so<=x"c";
when b"011001"=> so<=x"1";
when b"011011"=> so<=x"a";
when b"011101"=> so<=x"e";
when b"011111"=> so<=x"9";
when b"100000"=> so<=x"a";
when b"100010"=> so<=x"6";
when b"100100"=> so<=x"9";
when b"100110"=> so<=x"0";
when b"101000"=> so<=x"c";
when b"101010"=> so<=x"b";
when b"101100"=> so<=x"7";
when b"101110"=> so<=x"d";
when b"110000"=> so<=x"f";
when b"110010"=> so<=x"1";
when b"110100"=> so<=x"3";
when b"110110"=> so<=x"e";
when b"111000"=> so<=x"5";
when b"111010"=> so<=x"2";
when b"111100"=> so<=x"8";
when b"111110"=> so<=x"4";
when b"100001"=> so<=x"3";
when b"100011"=> so<=x"f";
when b"100101"=> so<=x"0";
when b"100111"=> so<=x"6";
when b"101001"=> so<=x"a";
when b"101011"=> so<=x"1";
when b"101101"=> so<=x"d";
when b"101111"=> so<=x"8";
when b"110001"=> so<=x"9";
when b"110011"=> so<=x"4";
when b"110101"=> so<=x"5";
when b"110111"=> so<=x"b";
when b"111001"=> so<=x"c";
when b"111011"=> so<=x"7";
when b"111101"=> so<=x"2";
when others=> so<=x"e";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s5 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s5;
 architecture behaviour of s5 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"2";
```

```
when b"000010"=> so<=x"c";
when b"000100"=> so<=x"4";
when b"000110"=> so<=x"1";
when b"001000"=> so<=x"7";
when b"001010"=> so<=x"a";
when b"001100"=> so<=x"b";
when b"001110"=> so<=x"6";
when b"010000"=> so<=x"8";
when b"010010"=> so<=x"5";
when b"010100"=> so<=x"3";
when b"010110"=> so<=x"f";
when b"011000"=> so<=x"d";
when b"011010"=> so<=x"0";
when b"011100"=> so<=x"e";
when b"011110"=> so<=x"9";
when b"000001"=> so<=x"e";
when b"000011"=> so<=x"b";
when b"000101"=> so<=x"2";
when b"000111"=> so<=x"c";
when b"001001"=> so<=x"4";
when b"001011"=> so<=x"7";
when b"001101"=> so<=x"d";
when b"001111"=> so<=x"1";
when b"010001"=> so<=x"5";
when b"010011"=> so<=x"0";
when b"010101"=> so<=x"f";
when b"010111"=> so<=x"a";
when b"011001"=> so<=x"3";
when b"011011"=> so<=x"9";
when b"011101"=> so<=x"8";
when b"011111"=> so<=x"6";
when b"100000"=> so<=x"4";
when b"100010"=> so<=x"2";
when b"100100"=> so<=x"1";
when b"100110"=> so<=x"b";
when b"101000"=> so<=x"a";
when b"101010"=> so<=x"d";
when b"101100"=> so<=x"7";
when b"101110"=> so<=x"8";
when b"110000"=> so<=x"f";
when b"110010"=> so<=x"9";
when b"110100"=> so<=x"c";
when b"110110"=> so<=x"5";
when b"111000"=> so<=x"6";
when b"111010"=> so<=x"3";
when b"111100"=> so<=x"0";
when b"111110"=> so<=x"e";
when b"100001"=> so<=x"b";
when b"100011"=> so<=x"8";
when b"100101"=> so<=x"c";
when b"100111"=> so<=x"7";
when b"101001"=> so<=x"1";
when b"101011"=> so<=x"e";
when b"101101"=> so<=x"2";
when b"101111"=> so<=x"d";
when b"110001"=> so<=x"6";
when b"110011"=> so<=x"f";
when b"110101"=> so<=x"0";
when b"110111"=> so<=x"9";
when b"111001"=> so<=x"a";
when b"111011"=> so<=x"4";
when b"111101"=> so<=x"5";
when others=> so<=x"3";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s6 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s6;
 architecture behaviour of s6 is
begin
process(clk)
begin
```

```
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"c";
when b"000010"=> so<=x"1";
when b"000100"=> so<=x"a";
when b"000110"=> so<=x"f";
when b"001000"=> so<=x"9";
when b"001010"=> so<=x"2";
when b"001100"=> so<=x"6";
when b"001110"=> so<=x"8";
when b"010000"=> so<=x"0";
when b"010010"=> so<=x"d";
when b"010100"=> so<=x"3";
when b"010110"=> so<=x"4";
when b"011000"=> so<=x"e";
when b"011010"=> so<=x"7";
when b"011100"=> so<=x"5";
when b"011110"=> so<=x"b";
when b"000001"=> so<=x"a";
when b"000011"=> so<=x"f";
when b"000101"=> so<=x"4";
when b"000111"=> so<=x"2";
when b"001001"=> so<=x"7";
when b"001011"=> so<=x"c";
when b"001101"=> so<=x"9";
when b"001111"=> so<=x"5";
when b"010001"=> so<=x"6";
when b"010011"=> so<=x"1";
when b"010101"=> so<=x"d";
when b"010111"=> so<=x"e";
when b"011001"=> so<=x"0";
when b"011011"=> so<=x"b";
when b"011101"=> so<=x"3";
when b"011111"=> so<=x"8";
when b"100000"=> so<=x"9";
when b"100010"=> so<=x"e";
when b"100100"=> so<=x"f";
when b"100110"=> so<=x"5";
when b"101000"=> so<=x"2";
when b"101010"=> so<=x"8";
when b"101100"=> so<=x"c";
when b"101110"=> so<=x"3";
when b"110000"=> so<=x"7";
when b"110010"=> so<=x"0";
when b"110100"=> so<=x"4";
when b"110110"=> so<=x"a";
when b"111000"=> so<=x"1";
when b"111010"=> so<=x"d";
when b"111100"=> so<=x"b";
when b"111110"=> so<=x"6";
when b"100001"=> so<=x"4";
when b"100011"=> so<=x"3";
when b"100101"=> so<=x"2";
when b"100111"=> so<=x"c";
when b"101001"=> so<=x"9";
when b"101011"=> so<=x"5";
when b"101101"=> so<=x"f";
when b"101111"=> so<=x"a";
when b"110001"=> so<=x"b";
when b"110011"=> so<=x"e";
when b"110101"=> so<=x"1";
when b"110111"=> so<=x"7";
when b"111001"=> so<=x"6";
when b"111011"=> so<=x"0";
when b"111101"=> so<=x"8";
when others=> so<=x"d";
end case;
end if;
end process;
end;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s7 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s7;
 architecture behaviour of s7 is
begin
```

```
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"4";
when b"000010"=> so<=x"b";  when b"000100"=> so<=x"2";
when b"000110"=> so<=x"e";
when b"001000"=> so<=x"f";
when b"001010"=> so<=x"0";
when b"001100"=> so<=x"8";
when b"001110"=> so<=x"d";
when b"010000"=> so<=x"3";
when b"010010"=> so<=x"c";  when b"010100"=> so<=x"9";
when b"010110"=> so<=x"7";
when b"011000"=> so<=x"5";
when b"011010"=> so<=x"a";
when b"011100"=> so<=x"6";
when b"011110"=> so<=x"1";
when b"000001"=> so<=x"d";
when b"000011"=> so<=x"0";  when b"000101"=> so<=x"b";
when b"000111"=> so<=x"7";
when b"001001"=> so<=x"4";
when b"001011"=> so<=x"9";
when b"001101"=> so<=x"1";
when b"001111"=> so<=x"a";
when b"010001"=> so<=x"e";
when b"010011"=> so<=x"3";  when b"010101"=> so<=x"5";
when b"010111"=> so<=x"c";
when b"011001"=> so<=x"2";
when b"011011"=> so<=x"f";
when b"011101"=> so<=x"8";
when b"011111"=> so<=x"6";
when b"100000"=> so<=x"1";
when b"100010"=> so<=x"4";  when b"100100"=> so<=x"b";
when b"100110"=> so<=x"d";
when b"101000"=> so<=x"c";
when b"101010"=> so<=x"3";
when b"101100"=> so<=x"7";
when b"101110"=> so<=x"e";
when b"110000"=> so<=x"a";
when b"110010"=> so<=x"f";  when b"110100"=> so<=x"6";
when b"110110"=> so<=x"8";
when b"111000"=> so<=x"0";
when b"111010"=> so<=x"5";
when b"111100"=> so<=x"9";
when b"111110"=> so<=x"2";
when b"100001"=> so<=x"6";
when b"100011"=> so<=x"b";  when b"100101"=> so<=x"d";
when b"100111"=> so<=x"8";
when b"101001"=> so<=x"1";
when b"101011"=> so<=x"4";
when b"101101"=> so<=x"a";
when b"101111"=> so<=x"7";
when b"110001"=> so<=x"9";
when b"110011"=> so<=x"5";  when b"110101"=> so<=x"0";
when b"110111"=> so<=x"f";
when b"111001"=> so<=x"e";
when b"111011"=> so<=x"2";
when b"111101"=> so<=x"3";
when others=> so<=x"c";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity s8 is port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s8;
 architecture behaviour of s8 is
begin
process(clk)
begin
if(clk'event and clk='1') then
case b is
when b"000000"=> so<=x"d";
```

```
when b"000010"=> so<=x"2";
when b"000100"=> so<=x"8";
when b"000110"=> so<=x"4";
when b"001000"=> so<=x"6";
when b"001010"=> so<=x"f";
when b"001100"=> so<=x"b";
when b"001110"=> so<=x"1";
when b"010000"=> so<=x"a";
when b"010010"=> so<=x"9";
when b"010100"=> so<=x"3";
when b"010110"=> so<=x"e";
when b"011000"=> so<=x"5";
when b"011010"=> so<=x"0";
when b"011100"=> so<=x"c";
when b"011110"=> so<=x"7";
when b"000001"=> so<=x"1";
when b"000011"=> so<=x"f";
when b"000101"=> so<=x"d";
when b"000111"=> so<=x"8";
when b"001001"=> so<=x"a";
when b"001011"=> so<=x"3";
when b"001101"=> so<=x"7";
when b"001111"=> so<=x"4";
when b"010001"=> so<=x"c";
when b"010011"=> so<=x"5";
when b"010101"=> so<=x"6";
when b"010111"=> so<=x"b";
when b"011001"=> so<=x"0";
when b"011011"=> so<=x"e";
when b"011101"=> so<=x"9";
when b"011111"=> so<=x"2";
when b"100000"=> so<=x"7";
when b"100010"=> so<=x"b";
when b"100100"=> so<=x"4";
when b"100110"=> so<=x"1";
when b"101000"=> so<=x"9";
when b"101010"=> so<=x"c";
when b"101100"=> so<=x"e";
when b"101110"=> so<=x"2";
when b"110000"=> so<=x"0";
when b"110010"=> so<=x"6";
when b"110100"=> so<=x"a";
when b"110110"=> so<=x"d";
when b"111000"=> so<=x"f";
when b"111010"=> so<=x"3";
when b"111100"=> so<=x"5";
when b"111110"=> so<=x"8";
when b"100001"=> so<=x"2";
when b"100011"=> so<=x"1";
when b"100101"=> so<=x"e";
when b"100111"=> so<=x"7";
when b"101001"=> so<=x"4";
when b"101011"=> so<=x"a";
when b"101101"=> so<=x"8";
when b"101111"=> so<=x"d";
when b"110001"=> so<=x"f";
when b"110011"=> so<=x"c";
when b"110101"=> so<=x"9";
when b"110111"=> so<=x"0";
when b"111001"=> so<=x"3";
when b"111011"=> so<=x"5";
when b"111101"=> so<=x"6";
when others=> so<=x"b";
end case;
end if;
end process;
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity roundfunc is port
(
clk : in std_logic;
li,ri : in std_logic_vector(1 to 32);
k : in std_logic_vector(1 to 48);
lo,ro : out std_logic_vector(1 to 32)
);
end roundfunc;

architecture behaviour of roundfunc is
```

```
component s1 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s2 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s3 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s4 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s5 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s6 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s7 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component s8 port
(
clk : in std_logic;
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end component;

component pp port
(
so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
: in std_logic_vector(1 to 4);
ppo : out std_logic_vector(1 to 32)
);
end component;

component desxor2 port
(
d,l : in std_logic_vector(1 to 32);
q : out std_logic_vector(1 to 32)
);
end component;

component desxor1 port
```

```
(
e : in std_logic_vector(1 TO 48);
b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
: out std_logic_vector (1 TO 6);
k : in std_logic_vector (1 TO 48)
);
end component;

component xp port
(
ri : in std_logic_vector(1 TO 32);
e : out std_logic_vector(1 TO 48));
end component;

signal e : std_logic_vector(1 to 48);
signal b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
: std_logic_vector(1 to 6);
signal so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
: std_logic_vector(1 to 4);
signal ppo : std_logic_vector(1 to 32);
begin

uxp: xp port map ( ri=>ri, e=>e );

udesxor1: desxor1 port map ( e=>e, k=>k, b1x=>b1x, b2x=>b2x, b3x=>b3x, b4x=>b4x, b5x=>b5x,
b6x=>b6x, b7x=>b7x, b8x=>b8x );

us1: s1 port map ( clk=>clk, b=>b1x, so=>so1x );
us2: s2 port map ( clk=>clk, b=>b2x, so=>so2x );
us3: s3 port map ( clk=>clk, b=>b3x, so=>so3x );
us4: s4 port map ( clk=>clk, b=>b4x, so=>so4x );
us5: s5 port map ( clk=>clk, b=>b5x, so=>so5x );
us6: s6 port map ( clk=>clk, b=>b6x, so=>so6x );
us7: s7 port map ( clk=>clk, b=>b7x, so=>so7x );
us8: s8 port map ( clk=>clk, b=>b8x, so=>so8x );

upp: pp port map ( so1x=>so1x, so2x=>so2x, so3x=>so3x, so4x=>so4x, so5x=>so5x, so6x=>so6x,
so7x=>so7x, so8x=>so8x, ppo=>ppo );

udesxor2: desxor2 port map ( d=>ppo, l=>li, q=>ro );

lo<=ri;
end behaviour;
```

A.5 desenc.vhd – for hierarchical placement and routing

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- to use some unnecessary cells in order for Silicon Ensemble to work because it doesn't work if no core components

entity my_or2 is
        port (I1: in STD_LOGIC;
              I2: in STD_LOGIC;
              O: out STD_LOGIC
             );
end my_or2;

architecture BEHAVIORAL of my_or2 is

begin

   o <= i1 or i2;

end BEHAVIORAL;


library ieee;

use ieee.std_logic_1164.all;

entity pc1 is port
(
key : in std_logic_vector(1 TO 64);
c0x,d0x : out std_logic_vector(1 TO 28)
);
```

```
end pc1;

architecture behavior of pc1 is

component my_or2 port
(
    I1: in STD_LOGIC;
    I2: in STD_LOGIC;
    O: out STD_LOGIC
);
end component;

        signal stemp    :       std_logic;
signal XX : std_logic_vector(1 to 56);
begin

   umy_or2 : my_or2  port map ( I1 => key(57), I2 => key(57), O => stemp );

XX(1)<=stemp; XX(2)<=key(49); XX(3)<=key(41); XX(4)<=key(33); XX(5)<=key(25); XX(6)<=key(17); XX(7)<=key(9);
XX(8)<=key(1); XX(9)<=key(58); XX(10)<=key(50); XX(11)<=key(42); XX(12)<=key(34); XX(13)<=key(26); XX(14)<=key(18);
XX(15)<=key(10); XX(16)<=key(2); XX(17)<=key(59); XX(18)<=key(51); XX(19)<=key(43); XX(20)<=key(35); XX(21)<=key(27);
XX(22)<=key(19); XX(23)<=key(11); XX(24)<=key(3); XX(25)<=key(60); XX(26)<=key(52); XX(27)<=key(44); XX(28)<=key(36);
XX(29)<=key(63); XX(30)<=key(55); XX(31)<=key(47); XX(32)<=key(39); XX(33)<=key(31); XX(34)<=key(23); XX(35)<=key(15);
XX(36)<=key(7); XX(37)<=key(62); XX(38)<=key(54); XX(39)<=key(46); XX(40)<=key(38); XX(41)<=key(30); XX(42)<=key(22);
XX(43)<=key(14); XX(44)<=key(6); XX(45)<=key(61); XX(46)<=key(53); XX(47)<=key(45); XX(48)<=key(37); XX(49)<=key(29);
XX(50)<=key(21); XX(51)<=key(13); XX(52)<=key(5); XX(53)<=key(28); XX(54)<=key(20); XX(55)<=key(12); XX(56)<=key(4);

c0x<=XX(1 to 28); d0x<=XX(29 to 56);
end behavior;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity pc2 is port
(
c,d : in std_logic_vector(1 TO 28);
k : out std_logic_vector(1 TO 48)
);
end pc2;
 architecture behavior of pc2 is
signal YY : std_logic_vector(1 to 56);
begin
YY(1 to 28)<=c; YY(29 to 56)<=d;

k(1)<=YY(14); k(2)<=YY(17); k(3)<=YY(11); k(4)<=YY(24); k(5)<=YY(1); k(6)<=YY(5);
k(7)<=YY(3); k(8)<=YY(28); k(9)<=YY(15); k(10)<=YY(6); k(11)<=YY(21); k(12)<=YY(10);
k(13)<=YY(23); k(14)<=YY(19); k(15)<=YY(12); k(16)<=YY(4); k(17)<=YY(26); k(18)<=YY(8);
k(19)<=YY(16); k(20)<=YY(7); k(21)<=YY(27); k(22)<=YY(20); k(23)<=YY(13); k(24)<=YY(2);
k(25)<=YY(41); k(26)<=YY(52); k(27)<=YY(31); k(28)<=YY(37); k(29)<=YY(47); k(30)<=YY(55);
k(31)<=YY(30); k(32)<=YY(40); k(33)<=YY(51); k(34)<=YY(45); k(35)<=YY(33); k(36)<=YY(48);
k(37)<=YY(44); k(38)<=YY(49); k(39)<=YY(39); k(40)<=YY(56); k(41)<=YY(34); k(42)<=YY(53);
k(43)<=YY(46); k(44)<=YY(42); k(45)<=YY(50); k(46)<=YY(36); k(47)<=YY(29); k(48)<=YY(32);
end behavior;
 LIBRARY ieee ;
use ieee.std_logic_1164.all;
 entity keysched is port
(
key : in std_logic_vector(1 to 64);
k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
: out std_logic_vector(1 to 48)
);
end keysched;
 architecture behaviour of keysched is

COMPONENT pc1 port
(
key : in std_logic_vector(1 TO 64);
c0x,d0x : out std_logic_vector(1 TO 28)
);
end COMPONENT;

COMPONENT pc2 port
(
c,d : in std_logic_vector(1 TO 28);
k : out std_logic_vector(1 TO 48)
);
end COMPONENT;

signal c0x,c1x,c2x,c3x,c4x,c5x,c6x,c7x,c8x,c9x,c10x,c11x,c12x,c13x,c14x,c15x,c16x : std_logic_vector(1 to 28);
signal d0x,d1x,d2x,d3x,d4x,d5x,d6x,d7x,d8x,d9x,d10x,d11x,d12x,d13x,d14x,d15x,d16x : std_logic_vector(1 to 28);
begin
```

```
  upc1: pc1 port map ( key=>key, c0x=>c0x, d0x=>d0x );

c1x <= c0x(2 to 28) & c0x(1);          d1x <= d0x(2 to 28) & d0x(1);
    c2x <= c1x(2 to 28) & c1x(1);          d2x <= d1x(2 to 28) & d1x(1);
    c3x <= c2x(3 to 28) & c2x(1 to 2);     d3x <= d2x(3 to 28) & d2x(1 to 2);
    c4x <= c3x(3 to 28) & c3x(1 to 2);     d4x <= d3x(3 to 28) & d3x(1 to 2);
    c5x <= c4x(3 to 28) & c4x(1 to 2);     d5x <= d4x(3 to 28) & d4x(1 to 2);
    c6x <= c5x(3 to 28) & c5x(1 to 2);     d6x <= d5x(3 to 28) & d5x(1 to 2);
    c7x <= c6x(3 to 28) & c6x(1 to 2);     d7x <= d6x(3 to 28) & d6x(1 to 2);
    c8x <= c7x(3 to 28) & c7x(1 to 2);     d8x <= d7x(3 to 28) & d7x(1 to 2);
    c9x <= c8x(2 to 28) & c8x(1);          d9x <= d8x(2 to 28) & d8x(1);
    c10x <= c9x(3 to 28) & c9x(1 to 2);     d10x <= d9x(3 to 28) & d9x(1 to 2);
    c11x <= c10x(3 to 28) & c10x(1 to 2);     d11x <= d10x(3 to 28) & d10x(1 to 2);
    c12x <= c11x(3 to 28) & c11x(1 to 2);     d12x <= d11x(3 to 28) & d11x(1 to 2);
    c13x <= c12x(3 to 28) & c12x(1 to 2);     d13x <= d12x(3 to 28) & d12x(1 to 2);
    c14x <= c13x(3 to 28) & c13x(1 to 2);     d14x <= d13x(3 to 28) & d13x(1 to 2);
    c15x <= c14x(3 to 28) & c14x(1 to 2);     d15x <= d14x(3 to 28) & d14x(1 to 2);
    c16x <= c15x(2 to 28) & c15x(1);       d16x <= d15x(2 to 28) & d15x(1);

  pc2x1: pc2 port map ( c=>c1x, d=>d1x, k=>k1x );
pc2x2: pc2 port map ( c=>c2x, d=>d2x, k=>k2x );
pc2x3: pc2 port map ( c=>c3x, d=>d3x, k=>k3x );
pc2x4: pc2 port map ( c=>c4x, d=>d4x, k=>k4x );
pc2x5: pc2 port map ( c=>c5x, d=>d5x, k=>k5x );
pc2x6: pc2 port map ( c=>c6x, d=>d6x, k=>k6x );
pc2x7: pc2 port map ( c=>c7x, d=>d7x, k=>k7x );
pc2x8: pc2 port map ( c=>c8x, d=>d8x, k=>k8x );
pc2x9: pc2 port map ( c=>c9x, d=>d9x, k=>k9x );
pc2x10: pc2 port map ( c=>c10x, d=>d10x, k=>k10x );
pc2x11: pc2 port map ( c=>c11x, d=>d11x, k=>k11x );
pc2x12: pc2 port map ( c=>c12x, d=>d12x, k=>k12x );
pc2x13: pc2 port map ( c=>c13x, d=>d13x, k=>k13x );
pc2x14: pc2 port map ( c=>c14x, d=>d14x, k=>k14x );
pc2x15: pc2 port map ( c=>c15x, d=>d15x, k=>k15x );
pc2x16: pc2 port map ( c=>c16x, d=>d16x, k=>k16x );
end;

library ieee;

use ieee.std_logic_1164.all;

entity ip is port
(
pt : in std_logic_vector(1 TO 64);
l0x : out std_logic_vector(1 TO 32);
r0x : out std_logic_vector(1 TO 32)
);
end ip;

architecture behavior of ip is
begin
l0x(1)<=pt(58); l0x(2)<=pt(50); l0x(3)<=pt(42); l0x(4)<=pt(34);
l0x(5)<=pt(26); l0x(6)<=pt(18); l0x(7)<=pt(10); l0x(8)<=pt(2);
l0x(9)<=pt(60); l0x(10)<=pt(52); l0x(11)<=pt(44); l0x(12)<=pt(36);
l0x(13)<=pt(28); l0x(14)<=pt(20); l0x(15)<=pt(12); l0x(16)<=pt(4);
l0x(17)<=pt(62); l0x(18)<=pt(54); l0x(19)<=pt(46); l0x(20)<=pt(38);
l0x(21)<=pt(30); l0x(22)<=pt(22); l0x(23)<=pt(14); l0x(24)<=pt(6);
l0x(25)<=pt(64); l0x(26)<=pt(56); l0x(27)<=pt(48); l0x(28)<=pt(40);
l0x(29)<=pt(32); l0x(30)<=pt(24); l0x(31)<=pt(16); l0x(32)<=pt(8);

r0x(1)<=pt(57); r0x(2)<=pt(49); r0x(3)<=pt(41); r0x(4)<=pt(33);
r0x(5)<=pt(25); r0x(6)<=pt(17); r0x(7)<=pt(9); r0x(8)<=pt(1);
r0x(9)<=pt(59); r0x(10)<=pt(51); r0x(11)<=pt(43); r0x(12)<=pt(35);
r0x(13)<=pt(27); r0x(14)<=pt(19); r0x(15)<=pt(11); r0x(16)<=pt(3);
r0x(17)<=pt(61); r0x(18)<=pt(53); r0x(19)<=pt(45); r0x(20)<=pt(37);
r0x(21)<=pt(29); r0x(22)<=pt(21); r0x(23)<=pt(13); r0x(24)<=pt(5);
r0x(25)<=pt(63); r0x(26)<=pt(55); r0x(27)<=pt(47); r0x(28)<=pt(39);
r0x(29)<=pt(31); r0x(30)<=pt(23); r0x(31)<=pt(15); r0x(32)<=pt(7);
end behavior;


LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity fp is port
(
l,r : in std_logic_vector(1 to 32);
ct : out std_logic_vector(1 to 64)
);
end fp;
```

```
architecture behaviour of fp is
begin
ct(1)<=r(8); ct(2)<=l(8); ct(3)<=r(16); ct(4)<=l(16); ct(5)<=r(24); ct(6)<=l(24); ct(7)<=r(32); ct(8)<=l(32);
ct(9)<=r(7); ct(10)<=l(7); ct(11)<=r(15); ct(12)<=l(15); ct(13)<=r(23); ct(14)<=l(23); ct(15)<=r(31); ct(16)<=l(31);
ct(17)<=r(6); ct(18)<=l(6); ct(19)<=r(14); ct(20)<=l(14); ct(21)<=r(22); ct(22)<=l(22); ct(23)<=r(30); ct(24)<=l(30);
ct(25)<=r(5); ct(26)<=l(5); ct(27)<=r(13); ct(28)<=l(13); ct(29)<=r(21); ct(30)<=l(21); ct(31)<=r(29); ct(32)<=l(29);
ct(33)<=r(4); ct(34)<=l(4); ct(35)<=r(12); ct(36)<=l(12); ct(37)<=r(20); ct(38)<=l(20); ct(39)<=r(28); ct(40)<=l(28);
ct(41)<=r(3); ct(42)<=l(3); ct(43)<=r(11); ct(44)<=l(11); ct(45)<=r(19); ct(46)<=l(19); ct(47)<=r(27); ct(48)<=l(27);
ct(49)<=r(2); ct(50)<=l(2); ct(51)<=r(10); ct(52)<=l(10); ct(53)<=r(18); ct(54)<=l(18); ct(55)<=r(26); ct(56)<=l(26);
ct(57)<=r(1); ct(58)<=l(1); ct(59)<=r(9); ct(60)<=l(9); ct(61)<=r(17); ct(62)<=l(17); ct(63)<=r(25); ct(64)<=l(25);
end;

LIBRARY ieee ;
use ieee.std_logic_1164.all;

entity desenc is port
(
pt : in std_logic_vector(1 TO 64);
key : in std_logic_vector(1 TO 64);
ct : out std_logic_vector(1 TO 64);
clk : in std_logic
);
end desenc;

architecture behavior of desenc is

component keysched port
(
key : in std_logic_vector(1 to 64);
k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
: out std_logic_vector(1 to 48)
);
end component;

component roundfunc port
(
clk : in std_logic;
li,ri : in std_logic_vector(1 to 32);
k : in std_logic_vector(1 to 48);
lo,ro : out std_logic_vector(1 to 32)
);
end component;

component ip port
(
pt : in std_logic_vector(1 TO 64);
l0x : out std_logic_vector(1 TO 32);
r0x : out std_logic_vector(1 TO 32)
);
end component;

component fp port
(
l,r : in std_logic_vector(1 to 32);
ct : out std_logic_vector(1 to 64)
);
end component;


signal k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x : std_logic_vector(1 to 48);
signal l0x,l1x,l2x,l3x,l4x,l5x,l6x,l7x,l8x,l9x,l10x,l11x,l12x,l13x,l14x,l15x,l16x : std_logic_vector(1 to 32);
signal r0x,r1x,r2x,r3x,r4x,r5x,r6x,r7x,r8x,r9x,r10x,r11x,r12x,r13x,r14x,r15x,r16x : std_logic_vector(1 to 32);
begin
ukeysched: keysched port map ( key=>key, k1x=>k1x, k2x=>k2x, k3x=>k3x, k4x=>k4x, k5x=>k5x, k6x=>k6x,
k7x=>k7x, k8x=>k8x, k9x=>k9x, k10x=>k10x, k11x=>k11x, k12x=>k12x, k13x=>k13x,
k14x=>k14x, k15x=>k15x, k16x=>k16x );

uip: ip  port map ( pt=>pt, l0x=>l0x, r0x=>r0x );

round1: roundfunc port map ( clk=>clk, li=>l0x, ri=>r0x,   lo=>l1x, ro=>r1x, k=>k1x );
round2: roundfunc port map ( clk=>clk, li=>l1x, ri=>r1x,   lo=>l2x, ro=>r2x, k=>k2x );
round3: roundfunc port map ( clk=>clk, li=>l2x, ri=>r2x,   lo=>l3x, ro=>r3x, k=>k3x );
round4: roundfunc port map ( clk=>clk, li=>l3x, ri=>r3x,   lo=>l4x, ro=>r4x, k=>k4x );
round5: roundfunc port map ( clk=>clk, li=>l4x, ri=>r4x,   lo=>l5x, ro=>r5x, k=>k5x );
round6: roundfunc port map ( clk=>clk, li=>l5x, ri=>r5x,   lo=>l6x, ro=>r6x, k=>k6x );
round7: roundfunc port map ( clk=>clk, li=>l6x, ri=>r6x,   lo=>l7x, ro=>r7x, k=>k7x );
round8: roundfunc port map ( clk=>clk, li=>l7x, ri=>r7x,   lo=>l8x, ro=>r8x, k=>k8x );
round9: roundfunc port map ( clk=>clk, li=>l8x, ri=>r8x,   lo=>l9x, ro=>r9x, k=>k9x );
round10: roundfunc port map ( clk=>clk, li=>l9x, ri=>r9x,   lo=>l10x, ro=>r10x, k=>k10x );
round11: roundfunc port map ( clk=>clk, li=>l10x, ri=>r10x,   lo=>l11x, ro=>r11x, k=>k11x );
round12: roundfunc port map ( clk=>clk, li=>l11x, ri=>r11x,   lo=>l12x, ro=>r12x, k=>k12x );
```

```
round13: roundfunc port map ( clk=>clk, li=>l12x, ri=>r12x,   lo=>l13x, ro=>r13x, k=>k13x );
round14: roundfunc port map ( clk=>clk, li=>l13x, ri=>r13x,   lo=>l14x, ro=>r14x, k=>k14x );
round15: roundfunc port map ( clk=>clk, li=>l14x, ri=>r14x,   lo=>l15x, ro=>r15x, k=>k15x );
round16: roundfunc port map ( clk=>clk, li=>l15x, ri=>r15x,   lo=>l16x, ro=>r16x, k=>k16x );

ufp: fp port map ( l=>r16x, r=>l16x, ct=>ct );

end behavior;
```

Vita

Xiaoquan Fu was born on March 9, 1972 in Changsha, Hunan, China. He was the only child of Songru Fu and Jianying Wen. He graduated with a Bachelor's Degree in Electrical Engineering from Hunan University in July, 1993. In August 1997 he came to the United States to begin his study and work in the Mechanical Engineering Department at the University of Tennessee. In 1999 he graduated with M.S degree in Mechanical Engineering and continued his study in the Department of Electrical and Computer Engineering. He enjoys everything full of competition and challenge.