Computer-Aided VLSI System Design
Midterm Examination
Apr. 20, 2004

Your Name _____

Student ID Number _____

Instructions

Exams: Consultation during the exam is not permitted. This is an open book exam. The exam is to be completed in one and half hours. Don't use scratch paper; show all of your work on these pages. Before you start writing, please check if you have all pages of the exam.

Regrading Policy: Exams will be accepted for regrading up to two weeks after you get the graded exam. No regrades after two weeks. When submitted, the ENTIRE exam or problem set is up for regrading (and down grading!)

Please sign the following statement upon completing this exam:

I certify that I will follow the above instructions. I have neither received nor given unpermitted aid on this examination.

Your signature _____

********** Score Board (to be filled by graders) **********

	total points	your points
problem 1	15	
problem 2	20	
problem 3	10	
problem 4	20	
problem 5	10	
problem 6	20	
total	100	

Problem 1 (15 points) <Procedural Block>

Given below is a testbench with sequential and parallel blocks as well as blocking and non-blocking assignments. Please show the outputs if this testbench is simulated via Verilog-XL.

```
`timescale 1ns/1ns
module test;
reg x,y,a,b;

initial begin
  y=1'b1; x=1'b0;
  #5 y=x; x=1'b1;
  fork
    #15 b=x;
    begin
      #10 y <= x;
          x <= y;
    end
    #5  a=!y;
  join
  #20 b<=a; x<=b;
  #5 $finish;
end

initial begin
  $monitor("%t %b %b %b %b",$time,x,y,a,b);
end
endmodule
```

內同時一起作

	x	y	a	b
0	0	1	x	x
5	1	0	x	x
10	1	0	1	x
15	0	1	1	x
20	0	1	1	0
40	0	1	1	1

ANS

	x	y	a	b
0	**0**	**1**	**x**	**x**
5	**1**	**0**	**x**	**x**
10	**1**	**0**	**1**	**x**
15	**0**	**1**	**1**	**x**
20	**0**	**1**	**1**	**0**
40	**0**	**1**	**1**	**1**

(2 points each)

Problem 2 (20 points) <Verilog Debug >

Which part(s) of the following codes is/are wrong or not appropriate? Why? Please write your answer in the box.

ANS

```
module Code(out,in,op) ; <= the semicolon is missed

output [3:0] out;

input [3:0] in;   <= this input should be claimed
input [2:0] op;

reg [3:0] temp_out;

wire [3:0] out; <= the output port should be claimed as wire type
 (If you skip this error, that's ok, too. You can get the points.)

assign out=op[2]?temp_out:in;

always@(op or in) <= the sensitive list should include "in"

begin

  if(op[0])

    temp_out =in;

  else if(op[1])

    temp_out =~in;
  else                <= without this case, latch will be introduced
    temp_out =in;
end
endmodule   <= this is missed
```

(total 7 bugs, 3 points each)

Problem 3 (10 points) <Synthesis>

A. (5 points) What's the difference between the synthesized results of using "set structure true" and "set flatten true"?

structure: 有合併的term

flatten: 展開. 每了term都展開, 不合併

ANS

Structuring:

Creates intermediate structures and can help both area and speed of a design.

Flattening:

Reduces design to SOP form and can be very area-intensive.

B. (5 points) What's the difference between "top-down hierarchical compile" and "bottom-up hierarchical compile"?

ANS

Top-Down

Advantage: 1. Inter-module dependencies are taken care of automatically; 2. Less time spent to drive the tool.

Disadvantage: 1. Long compile run time with large designs or complex constraints; 2. Even for designs without aggressive timing constraints, can be very memory and CPU intensive. 缺: 時間久, memory 需求大.

優: block 與 block 間有考慮到

Bottom-Up

Advantages: 1. Not limited by available memory; 2. Allow for time budgeting.

Disadvantage: 1. Require iterations until the interfaces of blocks are stable; 2. Require careful revision control.

Problem 4 (20 points) <Synthesis Optimization>

A. (10 points) What is the difference of the path delay between the implied implementation of the following two coding styles?

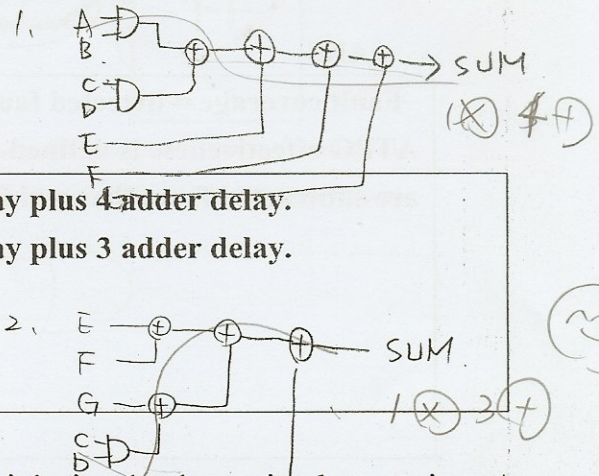
1. SUM <= A * B + C * D + E + F + G
2. SUM <= E + F + (G + C * D) + A * B

ANS

1. The maximum delay would be 1 multiplier delay plus 4 adder delay.

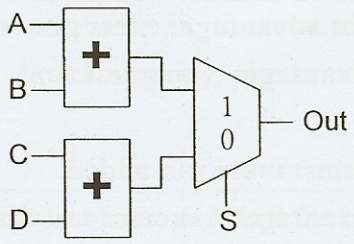
2. The maximum delay would be 1 multiplier delay plus 3 adder delay.

The second code has less path delay.



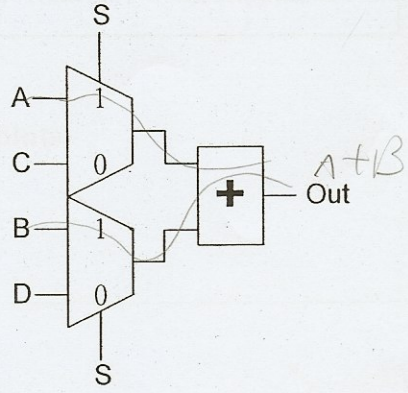
B. (10 points) The if-else construct implies multiplexing hardware implementation. A simple verilog example and the corresponding implementation are shown below. Although the Synopsys Design_Compiler could perform logic simplification, it's good for us to know how to reduce the hardware when doing RTL coding. Please write another verilog code which could perform the same function but have smaller area. Please draw the corresponding implementation.

```
If ( S )
   Out <= A + B;
else
   Out <= C + D;
end
```



ANS

```
if ( S )
   begin
   Op1 <= A;
   Op2 <= B;
   end
else
   begin
   Op1 <= C;
   Op2 <= D;
   end
Out <= Op1 + Op2;
```



FC = detect faults / total fault

Problem 5 (10 points) <DfT ATPG>

A. (5 points) What is the difference between fault coverage and ATPG effectiveness?

ANS

Fault coverage = detected faults / total faults

ATPG effectiveness is defined by the ATPG vendor. Usually the untestable faults are subtracted from the total faults.

B. (5 points) If your design has 10,000 scan cells in it, you can stitch them all into one chain of length 10,000. Or, you can spilt them into 10 chains of 1,000 scan cells. Please compare the advantage(s) and disadvantage(s) of these two schemes.

ANS

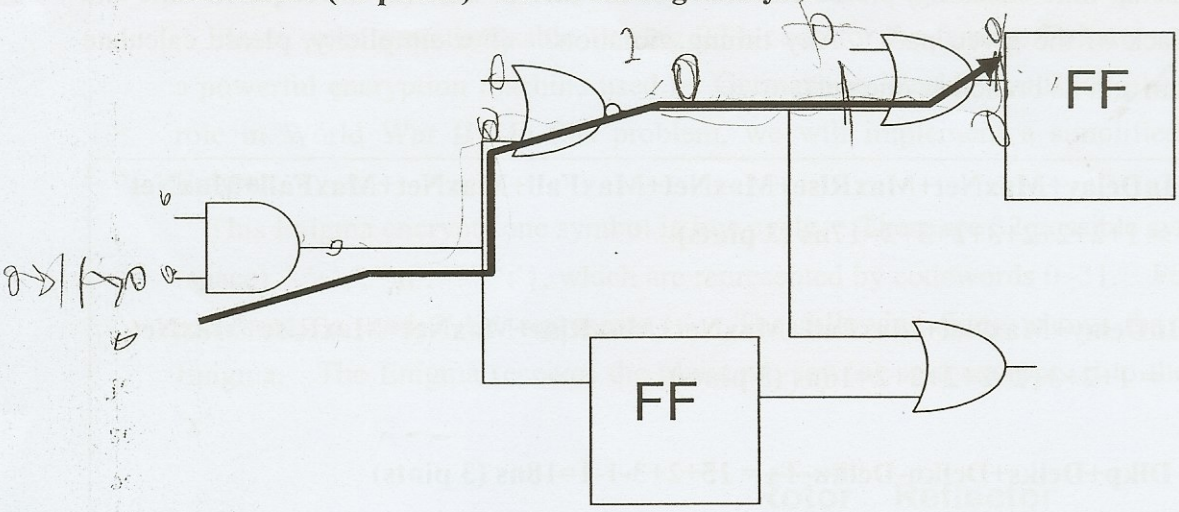
1 chain advantage: less pins added.

Disadvantage: long test time

10 chains: more pin added.

Disadvantage: shorter test time.

Problem 6 (20 points) <Static Timing Analysis>



Assume all gate have 2ns/3ns max rise/fall delay and 2ns/2ns min rise/fall delay. And, all nets have 2ns max delay and 1ns delay. The flip-flop have 2ns CLK-Q delay, 1ns setup time (Ts), and 1ns hold time (Th). The clock period is 15ns (Dclkp). The clock source latency is 2ns (Dclks). The clock network latency is 3ns (Dclkn). The clock uncertainty is 1ns. Moreover, all inputs delay is 1ns. All output delay is 3ns.

A. For hold time checking, please calculate (1) the arrival time (2) the required time and (3) the slack of the given path. Any timing violation? (For simplicity, please calculate fanout stem and its branch as one net).

ANS

AT rise: InDelay+MinNet+MinRise+MinNet+MinFall+MinNet+MinFall+MinNet
= 1+1+2+1+2+1+2+1=11ns (3 pints)

AT fall: InDelay+MinNet+MinFall+MinNet+MinRise+MinNet+MinRise+MinNet
= 1+1+2+1+2+1+2+1=11ns (3 pints)

RT R/F: Dclks+Dclkn+Dclku+Th = 2+3+1+1=7ns (3 pints)

Slack: AT-RT = 4/4ns > 0, No violation. (1 pints)

B. For setup time checking, please calculate (1) the arrival time (2) the required time and (3) the slack of the given path. Any timing violation? (For simplicity, please calculate fanout stem and its branch as one net).

ANS

AT rise: InDelay+MaxNet+MaxRise+MaxNet+MaxFall+MaxNet+MaxFall+MaxNet
= 1+2+2+2+3+2+3+2=17ns (3 pints)

AT fall: InDelay+MaxNet+MaxFall+MaxNet+MaxRise+MaxNet+MaxRise+MaxNet
= 1+2+3+2+2+2+2+2=16ns (3 pints)

RT R/F: Dlkp+Dclks+Dclkn-Dclku-Ts = 15+2+3-1-1=18ns (3 pints)

Slack: RT-AT = 1/2ns > 0, No violation. (1 pints)

RTL on-line Coding (100 points)

Please write a synthesizable Verilog RTL code for the famous Enigma machine, which is a powerful encryption machine used by Germans in World War II. (It played an important role in World War II.) In this problem, we will implement a simplified version of the Enigma.

This Enigma encrypts one symbol in one cycle. There are 32 possible symbols: {a~z, □ (space), '.', ',', '\n', '?', ':'}, which are represented by codewords 0~31. For example, 5'd0 represents 'a' and 5'd31 represents ':'. The following figure shows the structure of this Enigma. The Enigma receives the plaintext, say 'a', and encrypts it to the ciphertext, say 'x'.

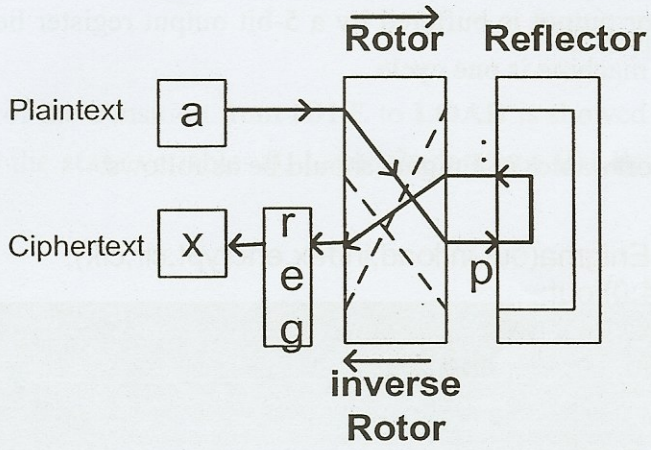


Fig. 1 Structure of Enigma

The Enigma machine mainly consists of these parts: a **rotor**, a **reflector**, an **inverse rotor**, and an output register. The Rotor is equivalent to a one-to-one substitution map (e.g. a→p, b→j, c→w, d→f, x→:). The following table shows an example rotor map. The **Rotor map** is not fixed. It is loaded by users during operation from the testbench.

Rotor Input	a	b	c	d	.....	x	.....	,	\n	?	:
Rotor output	P	j	w	f	.....	:	.....	?	k	z	□

The **inverse rotor map** is the inverse of the rotor map (e.g. p→a, j→b, w→c, f→d, :→x). The inverse rotor map is also a one-to-one mapping. The dashed lines in the Figure 1 mean that the Rotor map (and the inverse rotor map) is rotated in every cycle while encrypting. The details will be mentioned later.

The Reflector is a one-to-one bi-directional map (e.g. a→o if o→a). In this problem, the **Reflector map** is fixed as follows:

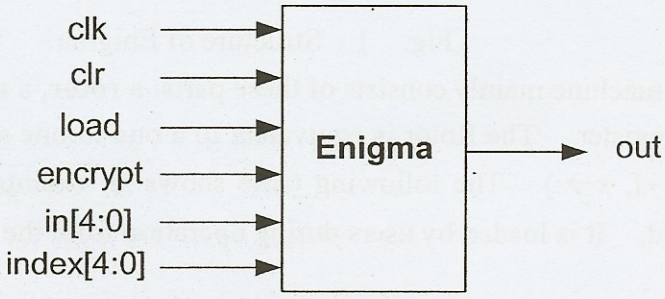
Reflector Input	a	b	c	d	e	F	G	H	i	J	k	l	m	n	o	P
Reflector Output	o	m	?	z	x	s	,	K	w	Y	h	□	b	v	a	:
Reflector Input	q	r	S	t	u	v	W	X	y	Z	□	.	,	\n	?	:
Reflector Output	.	t	F	r	\n	n	I	E	j	D	l	q	g	u	c	P

For a given plaintext, the Enigma maps it using the Rotor, the Reflector, and the inverse Rotor. For example, a→p→:→x in Figure 1. The rotor maps a to p. The reflector maps p to :. Then the inverse rotor maps : to x. It is easy to see that the operation of decryption is exactly the same as encryption. For example, if you feed 'x' into the Enigma machine, you will get 'a'.

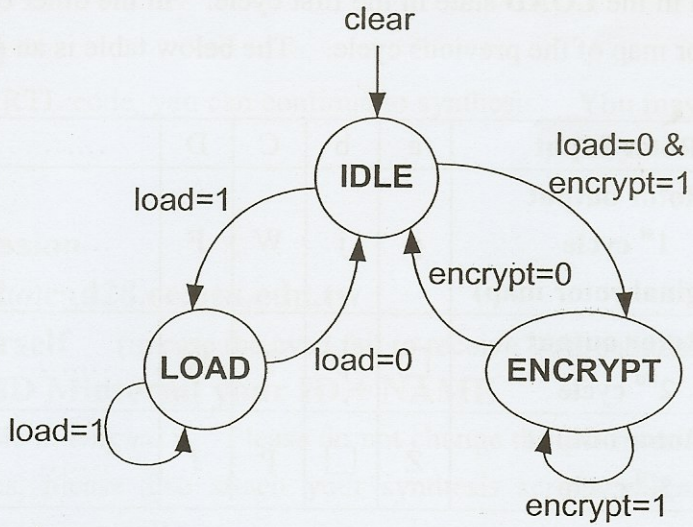
The inverse rotor output is buffered by a 5-bit output register before going out. That's, the latency of this machine is one cycle.

The module and port names of Enigma should be as follows:

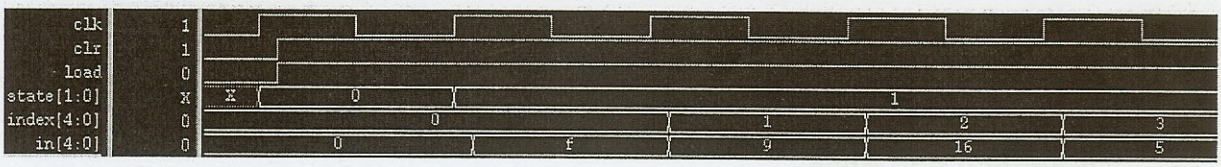
```
module Enigma(out,in,load,index,encrypt,clr,clk);
output [4:0] out;
input [4:0] in, index;
input load, clr, clk, encrypt;
```



All the registers are positive edge-triggered by the **clk** signal. The **clr** is a synchronous reset signal (active low). The signal **out** comes from the output register. This Enigma machine should be controlled by two control signals: **load** and **encrypt**. The state transition diagram is as follows, which includes three states.

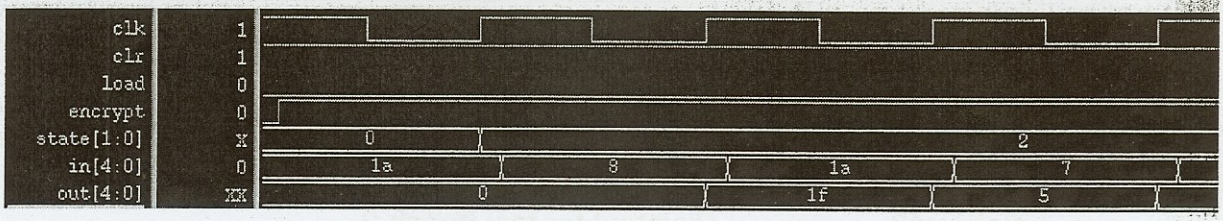


The waveform of the transition from **IDLE** to **LOAD** is showed in the following Debussy waveform, where the **state** variables 0, 1, and 2 represent **IDLE**, **LOAD**, and **ENCRYPT**, respectively.



In the state **LOAD**, the **original Rotor map** is loaded using signals: **index** and **in**. The signal **index** corresponds to the original Rotor map input, and **in** corresponds to the original rotor map output. For example, in the first cycle of the **LOAD** state in the above waveform, **index**=5'h0 and **in**=5'hf. This means the symbol 'a' (5'h0) should be mapped to 'p' (5'hf). Thus, it needs 32 cycles to load a complete Rotor map.

The waveform of the transition from **IDLE** to **ENCRYPT** is showed in the following Debussy waveform. The plaintext is applied and the ciphertext comes out at the following clock rising edge. For example, in this waveform, the symbol 'i' (5'h08) is encrypted to ':' (5'h1f) in the first cycle of **ENCRYPT** state.



The Rotor map should be rotated in every cycle, which is very important for the security of Enigma. When Enigma first enters the **ENCRYPT** state, it will use the **original Rotor**

map loaded in the **LOAD** state in the first cycle. In the other cycles, it will use the "right" rotated Rotor map of the previous cycle. The below table is an example:

Rotor Input	a	b	C	D	.....	,	\n	?	:
Rotor output 1st cycle (Original rotor map)	p	j	W	F	.....	?	k	z	□
Rotor output 2nd cycle	□	p	J	W	.....	r	?	k	z
Rotor output 3rd cycle	Z	□	P	J	.....	d	r	?	k

When the rotor map rotates, the inverse rotor map will follow. For example, in the second cycle, the rotor maps a→ □ and the inverse rotor maps □ → a.

Hint: You may use two register files to implement the rotor and the inverse rotor, respectively. However, you need to control them properly for the correct functions.

Testbench and Files

All the files you need are located in the ~cvsd/04S/Midterm directory. Please copy all the files into your own directory. Please use the *enigma.v* as a starting point. Please use the testbench files, *test_enigma_1.v* and *test_enigma_2.v*, to test your circuits (including encryption and decryption) until no encryption errors are reported and you can see the meaningful plaintext.

File List	
Filename	Description
ciphertext1	Ciphertext for test bench 1.
ciphertext2	Ciphertext for test bench 2.
enigma.v	Write your RTL code here. Please fill your name and id in the heading.
plaintext1	Plaintext for test bench 1.
plaintext2	Plaintext for test bench 2.
rotor1	Rotor for test bench 1.
rotor2	Rotor for test bench 2.
test_enigma_1.v	The given test bench 1.
test_enigma_2.v	The given test bench 2.

Bonus (20 points)

After finishing your RTL code, you can continue to synthesis. You may set the input delay to 1ns.

Electronic Submission

Please email to **cvsd@cad28.ee.ntu.edu.tw**

and cc to **yourself** (in case the cvsd fail to receive your submission).

Please entitle: **[CVSD Midterm] your ID + NAME.**

Attach your RTL code in *enigma.v*. Please do not change the I/O interface name and order. If you do the bonus, please also attach your synthesis scripts *enigma_synth.script* and *enigma_synth.v* in the same email.

Notice: Partial credits will be given. So please send your RTL code even if it does not pass all testbenches.