As a member of the Notre Dame community, I acknowledge that it is my responsibility to learn and abide by principles of intellectual honesty and academic integrity, and therefore I will not participate in or tolerate academic dishonesty.

By writing my name below, I agree to uphold the honor code.

Name (print clearly): Siddharth Joshi

CSE 20221 Logic Design
Midterm Exam, October 07, 2021

75 minutes
Closed book, closed notes, no calculator

Problem	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	10	
Total	100	

Problem 1 (10 points)

i. (3 points) Let **f = x & (y | (~y | x) & z) | (y&~z)**. Select all the terms for which the entries of the truth table of **f** will contain a 1.

a. x=0, y=0, z=0
b. x=0, y=0, z=1
c. x=0, y=1, z=0
d. x=0, y=1, z=1
e. x=1, y=0, z=0
f. x=1, y=0, z=1
g. x=1, y=1, z=0
h. x=1, y=1, z=1

ii. (2 points) Select which of the following always outputs 0

a. x^1
b. x^x
c. ~(x^x)
d. x^0

iii. (5 points) Select all the verilog statements that are equivalent. 1'b1 denotes TRUE and 1'b0 denotes FALSE

a. assign f = (x^1'b1)^(y);
b. assign f = (x&y)|~(x|y);
c. assign f = (x^1'b0)^(y);
d. assign f = ~(x&y)|(x|y);

Problem 2 (10 points)

What are the minimum required number of AND, OR gates required to implement F as a sum-of-products. Assume you have inputs and their complements available as inputs to your circuit.

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

a) One 2-input gate
b) One 3-input gate
c) Two 2-input gates
d) Two 3-input gates
e) One 2-input gate and one 3-input gate
f) Three 2-input gates
g) Three 3-input gates
h) Two 2-input gates and one 3-input gate
i) None of the above

Ans : **C**

Problem 3 (10 points)

Let's use the subscript to denote the base number system. That is: $(1011)_2$ implies binary representation, while $(B)_{16}$ implies the same number in hexadecimal (base-16).

i. (3 points) Select all the options which are strictly smaller than $(136)_{10}$

a. $(1000\ 0100)_2$
b. $(89)_{16}$
c. $(88)_{16}$
d. $(1000\ 1000)_2$
e. None of the above

ii. (5 Points) If all these values values are encoded as two's complement, select the true statement:

a. $C7 + BB = (7E)_{16}$
b. $C7 + BB = (126)_{10}$
c. $C7 + BB = (82)_{16}$
d. None of the above

iii. (2 points) If these hexadecimal numbers represent two's complement encoded values, circle the largest value

a. C4
b. C6
c. E8
d. 70

Problem 4 (10 points)

(6 points) Each of the numbers below is represented in two's complement, each variable, w, x,y,z can be any hexadecimal value independently of each other.

i.

$$\begin{array}{r} (B3X1)_{16} \\ +\ (47WZ)_{16} \\ \hline \end{array}$$

a) i. and ii. both can result in overflow
b) i. can result in overflow, but not ii.
c) i. cannot result in overflow but ii. can
d) neither i. nor ii. can possibly result in an overflow

ii.

$$\begin{array}{r} (7YCD)_{16} \\ +\ (0110)_{16} \\ \hline \end{array}$$

Ans: **C**

(4 points) Consider integers X[15:0] and Y[15:0] that result in SUM[15:0] when added. If we know the values of X[15:14] and Y[15:14] as well as SUM[15] we can always determine if there's an overflow:

a. For signed integers only, not for unsigned integers

b. For both signed and unsigned integers

c. For unsigned integers only not for signed integers

d. We cannot **always** determine if there's been an overflow in either case

Problem 5 (10 points)

1. (3 points) If you had to make a 2:1 multiplexor (2 inputs, 1 select bit) using only NAND gates. How many NAND gates would you need?

   a. 3
   b. 4
   c. 5
   d. 6
   e. 7
   f. 8
   g. None of the above

2. (3 points) If you had to make a 2:1 Decoder (2 outputs, 1 select bit) using only NOR gates. How many NOR gates would you need?

   a. 3
   b. 4
   c. 5
   d. 6
   e. 7
   f. 8
   g. None of the above

3. (4 points) A Majority gate, with output f and inputs a, b, and c is given by **f = a&b | a&c | b&c**. If you could only use 2- and 3-input NAND gates, how many such gates would you need to implement this function?

   a. 3
   b. 4
   c. 5
   d. 6
   e. 7
   f. 8
   g. None of the above

Problem 6 (10 points)

i. (2 points) Consider a two-input XOR gate with two inputs and one output. Given the ability to set the inputs to constants (e.g., 1/0). Is it possible to construct a NAND gate using an infinite supply of such two-input XOR gates?

a. Yes

b. No

ii. (4 points) Given only NAND gates, what is the minimum number of NAND gates needed to implement a two-input XOR.

a. 3

b. 4

c. 5

d. 6

e. 7

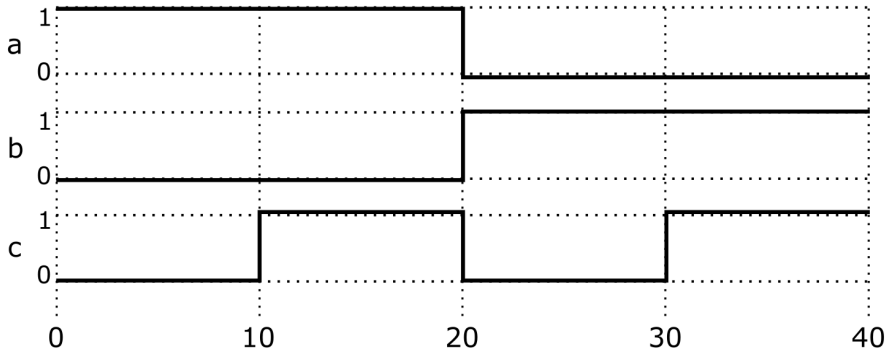
f. 8

iii. (4 points) Below is the Verilog code for two modules, thing and unknown:

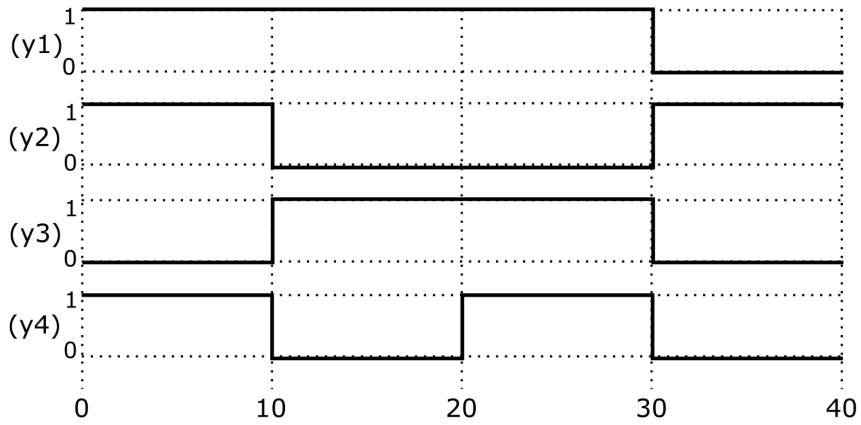
```
module thing (
   input    a,
   input    b,
   output   y
   );

   assign y = a & ~b | b & ~a;

endmodule

module unknown (
   input    a,
   input    b,
   input    c,
```

```
    output   y
    );

    wire w;

    thing thing1 (
       .a (a),
       .b (b),
       .y (w)
    );

    thing thing2 (
       .a (w),
       .b (c),
       .y (y)
    );

endmodule
```

The following waveforms are applied to the inputs of unknown:



Which of the following is the correct waveform for output y of module unknown?



a. y1
b. y2
c. y3
d. y4

Problem 7 (10 points)

i. (3 points) Given a Karnaugh map for a 4-input function, the maximum number of product terms that can ever remain after minimization are:

a. 6

b. 7

c. 8

d. 9

e. Impossible to determine without knowing the entries

ii. (3 points) Select all the correctly minimized expressions for the K-Map below

ab \ cd	00	01	11	10
00		1	1	1
01				
11				
10	1			1

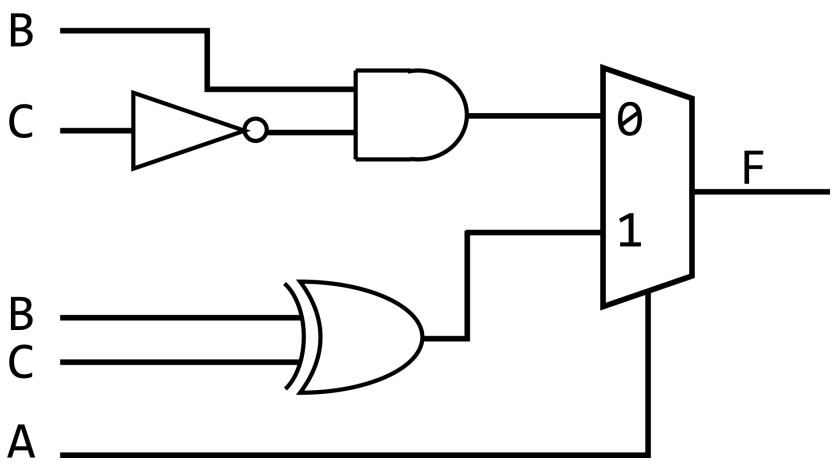
a. (~a&~b&d) | (a&~b&~d)|(~a&~b&c)

b. (~a&~b&d) | (a&~b&~d)|(~b&c&~d)

c. (~a&~b&d) | (a&~b&~d)|(~b&c&~d)|(~a&~b&c)

d. none of the above

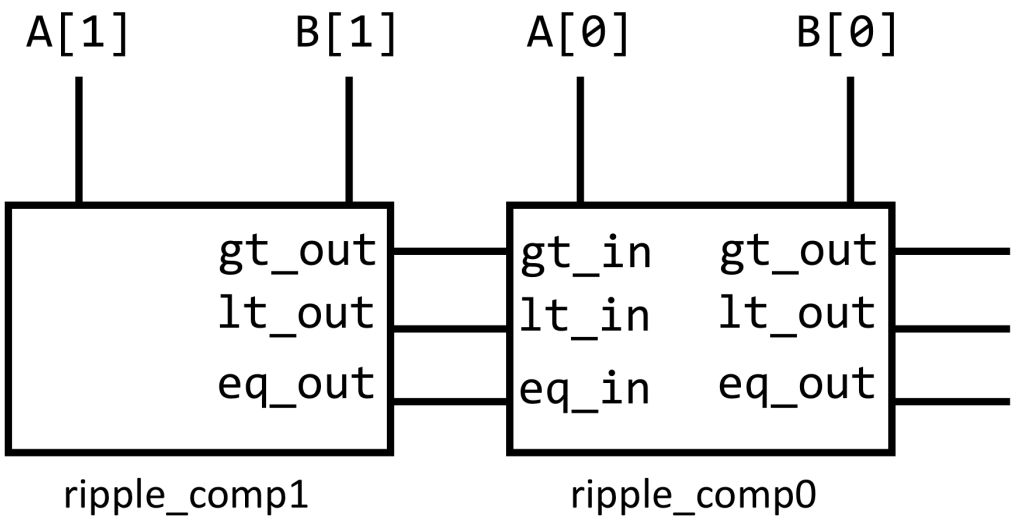
iii. (4 points) Correctly fill out the truth table for the given circuit.



A	B	C	F
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

Problem 8 (10 points)

A ripple comparator, like a ripple carry adder, chains instances of a single comparator module to output whether one unsigned integer is larger than, equal to, or lesser than the other. In this problem we will design such a comparator for two bit values. Note, that unlike in a ripple carry adder, for the comparator the comparison should start at the MSB and end at the LSB.



i. (3 points) Complete the verilog module for ripple_comp1. **To get you started, if A[1] = 1 and B[1] = 0; gt_out = 1;**

```
module ripple_comp1(
input a,
input b,
output eq_out,
output gt_out,
output lt
);

assign gt_out = _a&~b_____________________;

assign lt_out = _~a&b_____________________;

assign eq_out = __~(a^b)__________________;


endmodule
```

ii. (3 points) Complete the truth table for ripple_comp0. **X** indicates that the value doesn't matter. e.g. if gt_in = 1, then the value of the LSBs don't matter.

A[0]	B[0]	gt_in	lt_in	eq_in	gt_out	lt_out	eq_out
x	x	1	x	x	1	0	0
x	x	x	1	x	0	1	0
0	0	x	x	1	**0**	**0**	**1**
0	1	x	x	1	**0**	**1**	**0**
1	0	x	x	1	**1**	**0**	**0**
1	1	x	x	1	**0**	**0**	**1**

iii. (4 points) complete the verilog code for ripple_comp0

```
module ripple_comp0(
input a,
input b,
input gt_in,
input eq_in,
input lt_in,
output eq_out,
output gt_out,
output lt
);

assign gt_out = gt_in | (a&~b) ; //students could also & the second term with eq_in
assign lt_out = lt_in | (~a&b) ;
assign eq_out = eq_in | ~(a^b) ;

endmodule
```

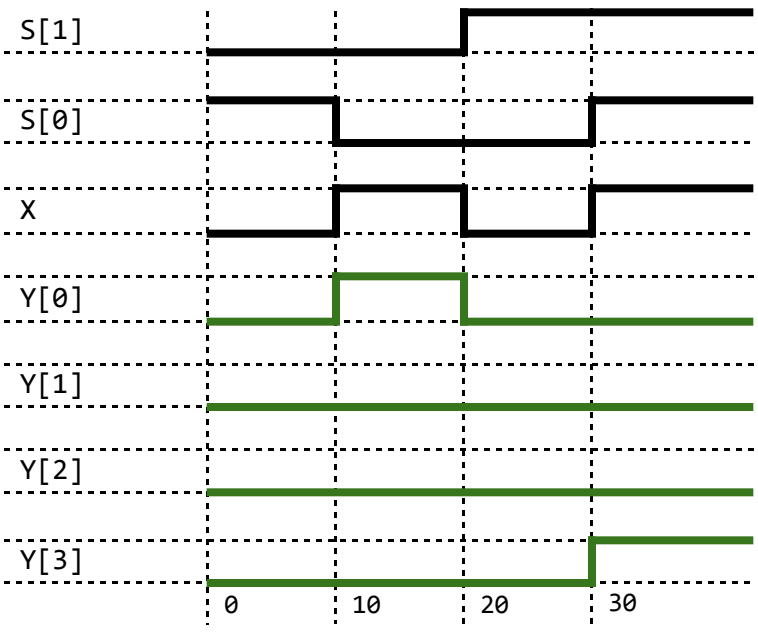
Problem 9 (10 points)

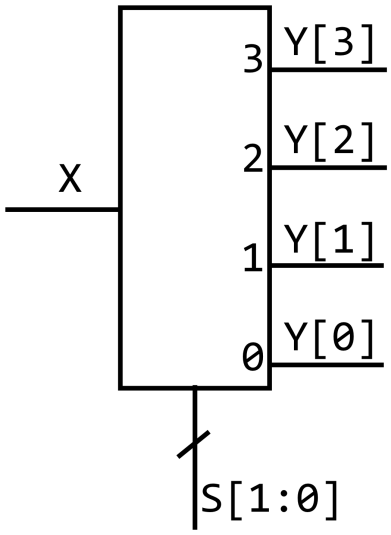
A De-Multiplexor or a DEMUX takes an input (X) and places it on one of 2^n outputs (Y[2^n-1:0]) based on an n-bit selection input (S[n-1:0]).

i. (4 points) Complete the verilog code below to correctly describe a DEMUX.

```
module demux(
input X,
input S,
output [1:0] Y
);

assign Y[0] = __X&~S_____________________________________;
assign Y[1] = __X&S______________________________________;

endmodule
```

ii. (2 points) Fill in the timing diagram, with the correct waveforms for Y[3:0]





Answer some questions about a testbench for the DEMUX.

iii. (2 points) Select all the true statements about the testbench for this

a. **S should be registers**
b. S should be wires
c. **X should be a register**
d. X should be a wire
e. Y should be registers
f. **Y should be wires**

iv. (2 points) Consider a testbench that replicates the waveform above, after the "initial begin" --- you type

a. S[1]=0; S[0]=0; X=0;
b. #10 S[1]=0; S[0]=1; X=0;
c. **S[1]=0; S[0]=1; X=0;**
d. #10 S[1]=0; S[0]=0; X=0;
e. S[1]=0; S[0]=0; Y=0;
f. #10 S[1]=0; S[0]=0; Y=0;
g. #10 S[1]=0; S[0]=1; Y=0;
h. S[1]=0; S[0]=1; Y=0;

Problem 10 (10 points)

The following represents machine code to be implemented on Albacore. This code is meant to loop a few times and then terminate.

Addr	Label	Pseudocode	Assembly	Hex
0				7000
1				7101
2				7222
3				3312
4				B033
5				0001
6				6331
7				AFD0
8				F000

i. (3 Points) As currently written, this code results in an infinite loop. If this error can be fixed by changing just a single line, circle the line number corresponding to the line number which has the error.

0 1 2 3 4 5 6 7 8

ii. (4 Points) Write down the **assembly** code for the corrected command below

BZ, R3, 4

iii. (3 points) What is the value of R0 after the corrected program is executed.

a. 0x0005
b. 0x0006
c. 0x0007
d. 0x0008

albaCore Instruction Set and Encoding

Assembly Language Syntax				**Behavior**
add	rw	ra	rb	R[rw] ← R[ra] + R[rb]
sub	rw	ra	rb	R[rw] ← R[ra] - R[rb]
and	rw	ra	rb	R[rw] ← R[ra] & R[rb]
or	rw	ra	rb	R[rw] ← R[ra] | R[rb]
not	rw	ra		R[rw] ← ~R[ra]
shl	rw	ra	shamt4	R[rw] ← R[ra] << shamt4
shr	rw	ra	shamt4	R[rw] ← R[ra] >> shamt4
ldi	rw	imm8		R[rw] ← {4'h0, imm8}
ld	rw	rb	offset4	R[rw] ← M[R[rb]+offset4]
st	ra	rb	offset4	M[R[rb]+offset4] ← R[ra]
br	disp8			pc ← pc + signExtend(disp8)
bz	rb	disp		if (R[rb] == 0) pc ← pc + signExtend(disp8) else pc ← pc + 1
bn	rb	disp		if (R[rb] < 0) pc ← pc + signExtend(disp8) else pc ← pc + 1
jal	target12			pc ← {pc[15:12, target12} R[15] ← pc + 1
jr	ra			pc ← R[ra]
quit				pc freezes

Instruction				Encoding													
mnemonic	arg1	arg2	arg3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
add	rw	ra	rb	opcode = 0				rw				ra				rb			
sub	rw	ra	rb	opcode = 1				rw				ra				rb			
and	rw	ra	rb	opcode = 2				rw				ra				rb			
or	rw	ra	rb	opcode = 3				rw				ra				rb			
not	rw	ra		opcode = 4				rw				ra							
shl	rw	ra	shamt	opcode = 5				rw				ra				shamt			
shr	rw	ra	shamt	opcode = 6				rw				ra				shamt			
ldi	rw	imm		opcode = 7				rw				imm							
ld	rw	rb	offset_ld	opcode = 8				rw				offset_ld				rb			
st	ra	rb	offset_st	opcode = 9				offset_st				ra				rb			
br	disp			opcode = 10				displacement											
bz	rb	disp		opcode = 11				displacement								rb			
bn	rb	disp		opcode = 12				displacement								rb			
jal	target			opcode = 13				target											
jr	ra			opcode = 14								ra							
quit				opcode = 15								imm							