CENG 232: Logic Design
2019/2020 Spring
Lab Final Exam Solutions

Duration: 50 minutes (includes uploading of your document)
Number of Questions: 6
Total Points: 100

" I have read and understood the implications of the METU Honor Code. To be precise, I understand that this is an online exam, and I am forbidden to communicate by any means to anyone in or out of my class. I am forbidden to share any of the questions or answers with anyone during the exam. I understand and accept to obey all the rules announced by the course staff, and that failure to obey these will result in disciplinary action."

1.) (15 points)

Given the following variable definitions:
reg [4:1] regA; //4 bits
reg [4:1] regB; //4 bits

with the following initial values

regA=4'b1010;

regB=4'b1100;

What is the evaluation results of the following Verilog expressions?(3 points for each)

a)regA << 2 :_____4'b1000_____

b){regA,regB} :_____8'b10101100_____

c){4{1'b1}} :_____4'b1111_____

d){regA, {4 {1'b1}}}:_____8'b10101111_____

e) regA | regB :_____4'b1110_____

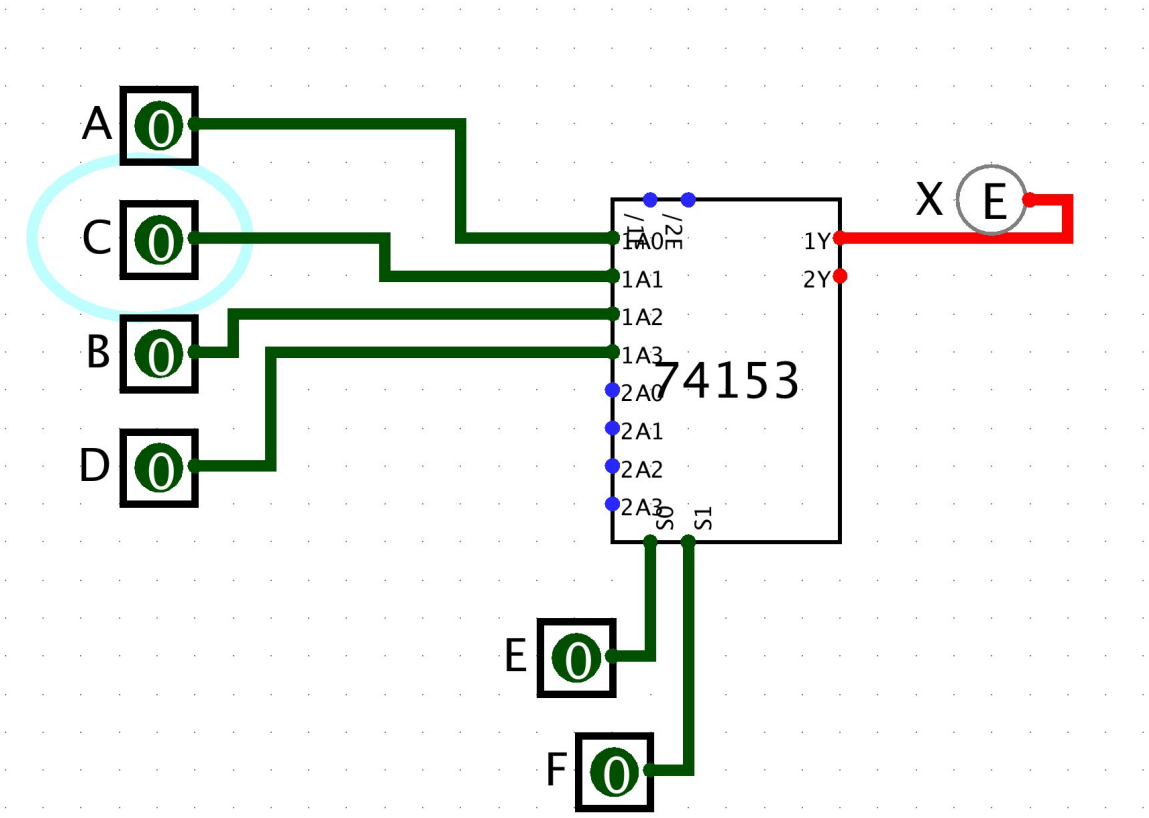
2.) (15 points)

X = 1 if (
(A is 1 AND E is NOT 1 AND F is NOT 1) OR
(B is 1 AND E is 1 AND F is NOT 1) OR
(C is 1 AND E is NOT 1 AND F is 1) OR

(D is 1 AND E is 1 AND F is 1)

)

Draw the circuit that takes inputs A, B, C, D, E, F and gives the correct output X.
(Hint: You can use 4x1 Multiplexer)



3.) (23 points)

Write a module with given specifications:

- The module basically reads the operationCode input, conducts the corresponding operation on A and B and stores the result to output register C.
- Any change in A, B or operationCode produces the output C.
- For each operationCode, the corresponding operations are given below:

operationCode	Operation
00	A and B
01	NOT(B)
10	A xor B
11	rotl(A)

PS: **rotl** shifts all the bits to the left by one and sets the right-most bit to the previous value of the left-most bit.

PS: and, not and xor are all bitwise operations.

```
module ALU(
input [3:0] A,
input [3:0] B,
input [1:0] operationCode,
output reg [3:0] C
);
```

```
//write your code below
always@ (A or B or operationCode)  //3pts.
begin
      if (operationCode ==2'b00) C= A & B; //5 pts.
      else if (operationCode ==2'b01) C= ~B; //5 pts.
      else if (operationCode ==2'b10)  C = A ^ B //5 pts.
      else C= {A[2:0],A[3]} //5 pts.
end
```

```
endmodule
```

4.) (15 points)

```
module rgModule(
  input [a] dataIn,
  input [b] mask,
  output reg [c] out1,
  output reg [d] out2
);
always @(dataIn)
begin
   out1={e} ;
   out2= {dataIn[7:5] , dataIn[1] ,
dataIn[2:0] | mask[2:0] , dataIn[4] } ;
end
endmodule
```

a.) Consider a data input of 8 bits and a mask of 4 bits. The out1 is supposed to be an 8-bit output generated by the following:

- Leftmost four bits of out1 = Leftmost four bits of dataIn AND mask.

- Rightmost four bits of out1 = Right most four bits of dataIn XOR mask.

Fill the variables (**a,b,c,d** and **e**) by using the above. (You can use out2 as an example to fill the **e** value).

a)(0.5 pts.):____[7:0]____
b) (0.5 pts.):____[3:0]____
c) (0.5 pts.):____[7:0]____
d) (0.5 pts.):____[7:0]____
e) (5 pts.):____dataIn[7:4] & mask[3:0], dataIn[3:0] ^ mask[3:0]____

b.) Now consider the following inputs:
dataIn = 10100111
mask = 1001

What is the value of out1 and out2?

out1:(4 pts.):___10001110_____
out2: (4 pts.):___10111110_____

5.) (20 points)

Write a Counter module with given specifications:

- Initially Count is 0.
- Count is set to zero if Reset is set to 1 (Reset operation does not get affected by Enable input).
- Counter should be incremented by 1 at each operation if and only if enable is 1.
- Counting operation is both synchronous with positive edge of the clock, and it is asynchronous with Reset (posedge) input.
- The maximum value the counter can hold is 255, if count becomes higher than that it should return to 0.

```
module Counter(
output reg [7:0] Count,
input Enable,
input Reset,
input Clk
);
```

```
//write your code below
initial
begin
Count<= 8'b000000000;
end

always@(posedge Clk or
posedge Reset)
begin

if (Reset) Count<=0;
else if (Enable)
begin
if(Count<255)
Count<=Count+1;
else
Count<=0;
end

end
```

```
endmodule
```

6.) (12 points)

a) Explain the behaviors of ”assign” and ”$monitor” statements briefly.

(3 Points) assign: statement is used for continuous value transfer for net type variables. As soon as a value change occurs on the right side of the assigment, the left side of the equation is updated after evaluating the right side. Left side only can be net type (net vector, net concatanation etc.), register type variable usage is not allowed on the left side. Right side can consist of both register and net types.
$monitor : statement continuously inspects the signals (reg, net) provided to it and when a value change occurs, it outputs all signal values in the specified format.

b)How do these two declarations differ?

```
reg   [7:0]    x;
reg   [0:7]    x;
```

(**3 Points**) Both statements declare a register vector with 8 bits. The left number on brackets specifies the most significant bit (MSB) of the vector. In the first line, an x register vector is created with indexing 7 to 0 where bit 7 is MSB and bit 0 is LSB whereas in second line indexing is from 0 to 7 where bit 0 is the MSB and bit 7 is LSB. Declarations impose different access regimes during read and write to register. With first declaration x[7:5] accesses the three most significant bits (bit7bit6bit5) whereas with second declaration x[5:7] accesses the 3 least significant bits (bit5bit6bit7). The order of the bits selected are determined via declaration of the variable.

c)What is the difference between these two declarations? Explain briefly.

```
reg   [7:0]     x  [15:0][20:0];
reg   x  [15:0][20:0][7:0];
```

(**3 Points**) First line declares a 2D array of register vector of length 8 bits and the 2D array has 16 rows with 21 columns whereas second line declares a 3D array of 1 bit register and the 3D array has 16 rows, 21 columns and 8 layers.

d)What is the final value of register y?

```
module   Test;
reg   [7:0]    x,   y;
reg   [0:7]    z
initial        begin
z  =  15;
x  =  25;
y  =  x+z;
end
```

(**3 Points**) 40