ESS-201 Assignment on Verilog

[Total Marks = 60]

A]. The 1-bit full adder can be expressed in a sum of products form.

sum = a.b.c_in + a'.b.c_in' + a'.b'.c_in + a.b'.c_in'
c_out = a.b + b.c_in + a.c_in
Assuming **a, b, c**_in are the inputs and sum and **c_out** are the outputs, design a logic circuit to implement
the 1-bit full adder, using only AND, NOT, and OR gates. Write the Verilog description for the circuit.
You may use up to 4-input Verilog primitive and AND, OR gates. Write the stimulus for the full adder and
check the functionality for all input combinations.
[10]

B]. A magnitude comparator checks if one number is greater than or equal to or less than another number.
A 4-bit magnitude comparator takes two 4-bit numbers, **A** and **B**, as input. We write the bits in **A** and **B** as follows.
The leftmost bit is the most significant bit.
A = A(3) A(2) A(1) A(0)
B = B(3) B(2) B(1) B(0)
The magnitude can be compared by comparing the numbers bit by bit, starting with the most significant bit.
If any bit mismatches, the number with bit **0** is the lower number.
To realize this functionality in logic equations, let us define an intermediate variable.
Notice that the function below is an XNOR function.
x(i) = A(i).B(i) + A(i)'.B(i)'
The three outputs of the magnitude comparator are
A_gt_B,
A_lt_B,
A_eq_B.
They are defined with the following logic equations:
A_gt_B = A(3).B(3)' + x(3).A(2).B(2)' + x(3).x(2).A(1).B(1)' + x(3).x(2).x(1).A(0).B(0)'
A_lt_B = A(3)'.B(3) + x(3).A(2)'.B(2) + x(3).x(2).A(1)'.B(1) + x(3).x(2).x(1).A(0)'.B(0)
A_eq_B = x(3).x(2).x(1).x(0)
Write the Structural Verilog description of the module magnitude_comparator using in built logic gates in Verilog.

Instantiate the magnitude comparator inside the stimulus module and try out a few combinations of **A** and **B**. **[10]**

C]. Show how the function, **f = w1w3 + w1w3 + w2w3 + w1w2**, can be realized using one or more instances of the circuit
in **Figure P4.1**. Note that there are no NOT gates in the circuit; hence complements of signals have to be generated
using the multiplexers in the logic block. Write the Structural Verilog description of your implementation using
Behavioral Verilog description of the circuit shown in **Figure P4.1**. Instantiate your implementation inside the stimulus
module and try out all combinations of the inputs **w1, w2** and **w3**, respectively.
[10]

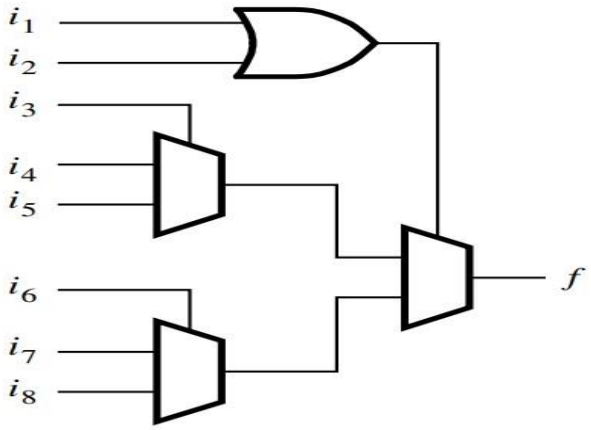

Figure P4.1 A multiplexer-based circuit.

D]. The following code checks for adjacent ones in an **8-bit binary vector**.

```
always @(A)
begin
    f = A[1] & A[0];
    for (k = 2; k < n; k = k+1)
        f = f | (A[k] & A[k−1]);
end
```

With blocking assignments this code produces the desired logic function, which is **f = A1A0 + A2A1 + ··· + A7A6.**
What logic function is produced if we change the code to use non-blocking assignments? Complete the Verilog
behavioral description of the above code with blocking assignment which is given in terms of an n-bit binary
vector by using a module with interface description. Simulate your module using a test-setup module invoked

in a test bench and try out a few 8 bit binary vectors to validate your completed Verilog code. **[10]**

E]. An FSM is defined by the state-assigned table in **Figure P6.1**.

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
0 0	1 0	1 1	0
0 1	0 1	0 0	0
1 0	1 1	0 0	0
1 1	1 0	0 1	1

Figure P6.1 State-assigned table

Re-write the state table with symbolic states **A, B, C** and **D** replacing the assigned states of **00, 01, 10** and **11,**
respectively in the state assigned table in **Figure P6.1**, i.e, assume **State A** is **00**, **State B** is **01**, **State C** is **10** and
State D is **11**. Do a new state assignment (or encoding) which is different from the one given in **Figure P6.1**.
Derive FSM circuits that respectively realizes both the state assigned table using D flip-flops. Write the Verilog
behavioral description of the two FSMs and check the correctness of the state-tables using appropriate test-setup
modules and your Verilog implementation modules within a test bench. Comment on the different Boolean
expressions obtained for the next state bits in the Next State Function and the output bits in the Output Function
for the two state assigned tables.
[20]