Assignment 3

Ashmitha Murali

Prof. Alex Yang

Verilog HDL Lab E461L

17th September 2025

a.

1.

**always@(A or B) begin  
if(A) C = B ^ A;  
else C = D & E;  
F = C | A;**  
**end**

The issue with the above code is that only ‘A’ and ‘B’ are included in the always block.

To fix this issue, we need to have all the variables from the RHS in the always block.

The fixed code is:

**always@(A or B or C or D or E) begin  
if(A) C = B ^ A;  
else C = D & E;  
F = C | A;**  
**end**

2.

always@(B)  
C = |B;  
always@(E)  
C = ^E;

The issue with the above code is that the value of C is being driven inside two different always blocks. The C is assigned |B and ^E at the same time, and this will create a race condition.

To avoid this issue, assign only one value to C at any given time. In my below example I am assigning C based on the s1 switch.

**always@(B or E) begin**

**C = s1 ? |B : ^E**

**end**

3.

always@(posedge clock)  
if(A) Q <= D;  
always@(Q or E)  
case (Q)  
0: F = E;  
default: F = 1;  
endcase

clock : 00001111  
D : xxxxxxxx  
E : 11110000  
Q :  
F :

In the above scenario, A value is not updated, and hence, the Q value never gets updated. The clock has only one positive edge at 5th bit, and during the same bit E also gets updated, but since Q is always X, the value of F will always be 1

**At time unit 1: A=x Q=x D=x E=1 F=1 clock=0  
At time unit 5: A=x Q=x D=x E=0 F=1 clock=1**

We can fix this by updating the value of A or make sure Q is updated.

I used below code to test this:

module test\_m();

reg A,Q,D,E,F,clock;

always@(posedge clock)

if(A) Q <= D;

always@(Q or E)

case(Q)

0: F = E;

default: F = 1;

endcase

initial begin

$monitor("%g A=%b Q=%b D=%b E=%b F=%b clock=%b", $time,A,Q,D,E,F,clock);

#1 D=1'bx;E=1;clock=0;

#1 D=1'bx;E=1;clock=0;

#1 D=1'bx;E=1;clock=0;

#1 D=1'bx;E=1;clock=0;

#1 D=1'bx;E=0;clock=1;

#1 D=1'bx;E=0;clock=1;

#1 D=1'bx;E=0;clock=1;

#1 D=1'bx;E=0;clock=1;

end

endmodule

4.

module top;  
 wire B;  
 bar u1 (A,B);  
 bar u2 (C,B);  
endmodule  
module bar (input D, output wire E);  
 assign E=~D;  
endmodule

In the above code, ‘A’ and ‘C’ ports are not declared, and hence they would be assigned as wires.

Also, there is a semicolon in “module bar (input D; output wire E);”. This should be a comma(,) as shown below:

**module bar (input D, output wire E);**

We have **two instances of bar driving the same wire B** as the output. This can create a race condition and should be avoided.

The fixed code is :

module top;  
 wire A,B,C,D;  
 bar u1 (A,B);  
 bar u2 (C,D);  
endmodule  
module bar (input D, output wire E);  
 assign E=~D;  
endmodule

5.

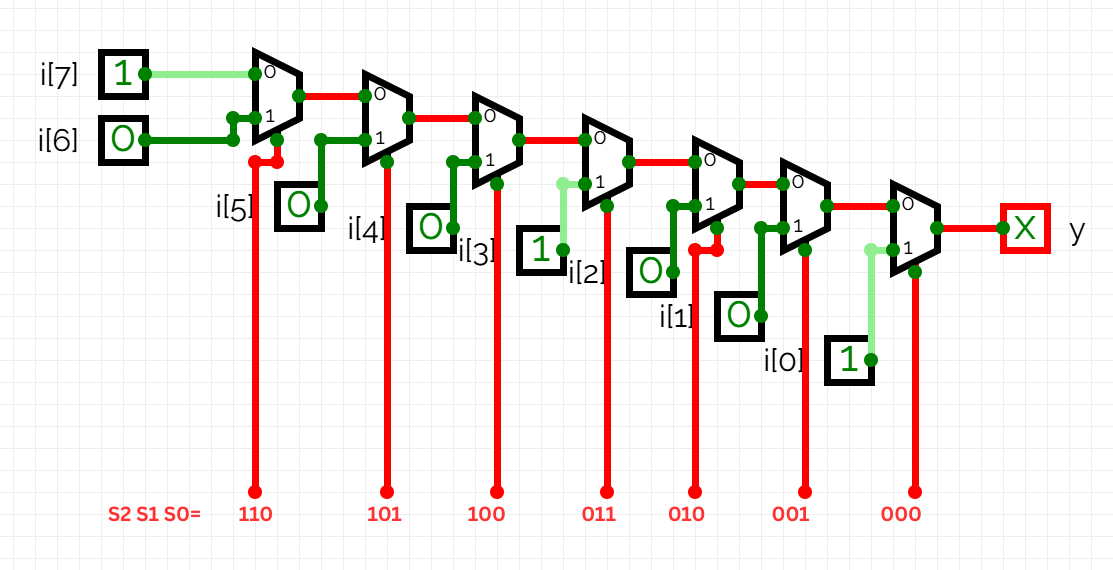
module foo(input A,B; output reg E);  
 wire C,D;  
 always@(posedge clock) E=B&D;  
 assign C=A^D;  
 assign D=C|B;  
endmodule

Everything looks good in the above module except for the clock signal. The clock variable needs to be explicitly declared as shown below. Also, the semicolon should be replaced by a comma in the module line:

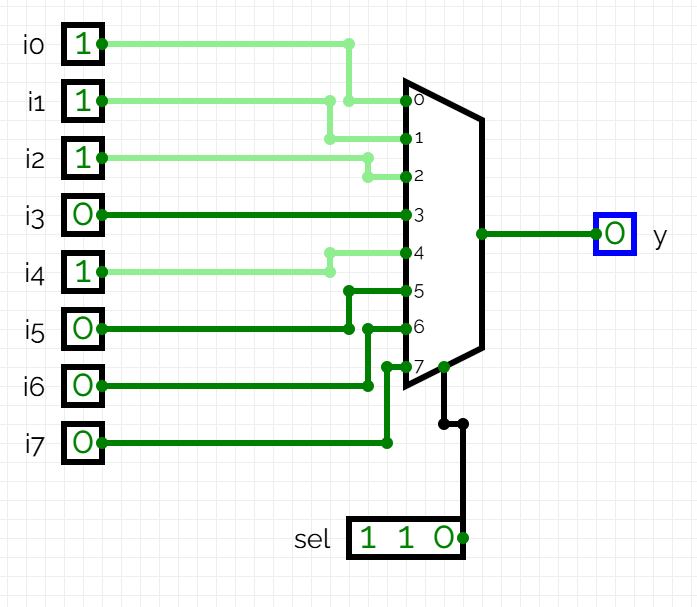
**module foo(input A,B, output reg E);  
 wire C,D;  
 wire clock;  
 always@(posedge clock) E=B&D;  
 assign C=A^D;  
 assign D=C|B;  
endmodule**

b.

With the IfMux8 module, a priority mux will be synthesized as shown below:



With the CaseMux8 module a non-priority mux will be synthesized as shown below.



With the IFMux8 module, the highest priority is given to the 0 input, and priority decreases as we go to higher bits, whereas in the CaseMux8 module, all the inputs have equal priority.