** San Francisco Bay University**

**EE461L - Verilog HDL Lab**

**Week#3 Combinational Logic**

Name: Melvin Divine Pritchard Student ID: 19857

**Answers :**

**IV. Exercises**

**a.** Are there something wrong in the following codes? And fix them.

1. always@(A or B) begin

if(A) C = B ^ A;

else C = D & E;

F = C | A;

end

* Has a syntax error, see below for the Verilog syntax for always block.

**Fixed**

always@(A or B) begin

if(A) begin

C = B - A;

end

else begin

F = C | A;

end

2. always@(B)

C = |B;

always@(E)

C = ^E;

* Syntax error in the line C =|B;

**Fixed**

reg C;

always@(B)begin

C | = B;

end

always@(E)

C \*= 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 :

**Fixed**

The value of F is determined based on the case statement and the Q value. If Q has a value of zero, F is equal to E; otherwise, F is equal to 1. The waveform value below is the result of a change in Q or E, which causes a change in the F value.

reg Q;

reg F;

always@(posedge clock) begin

if(A) Q <= D;

end

always@(Q or E)begin

case (Q)

0: F = E;

default: F = 1’b1;

endcase

end

initial begin

clock = ‘b00001111;

D = ‘bxxxxxxxx;

E = ‘b11110000;

Q = ‘bxxxx0000;

F = ‘b11110000;

end

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

**fixed**

wire A, C, B;

module top;

bar u1(A,B);

bar u2(C,B);

endmodule

module bar (input wire 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

**Fixed**

module foo(input wire A, input wire B; output reg E);

wire C,D;

always@(posedge clock)

begin

E = B & D;

end

initial

begin

assign C = A^D;

assign D = C|B;

endmodule

b. According to the modules IfMux8 and CaseMux8, draw the corresponding circuits and compare them.

* **IfMux8**

module IfMux8 (y, i, sel);

output y;

input [7:0] i;

input [2:0] sel;

reg y;

wire [7:0] i;

wire [2:0] sel;

always @(i or sel)begin

if (sel == 3'd0) y = i[0];

else if (sel == 3'd1) y = i[1];

else if (sel == 3'd2) y = i[2];

else if (sel == 3'd3) y = i[3];

else if (sel == 3'd4) y = i[4];

else if (sel == 3'd5) y = i[5];

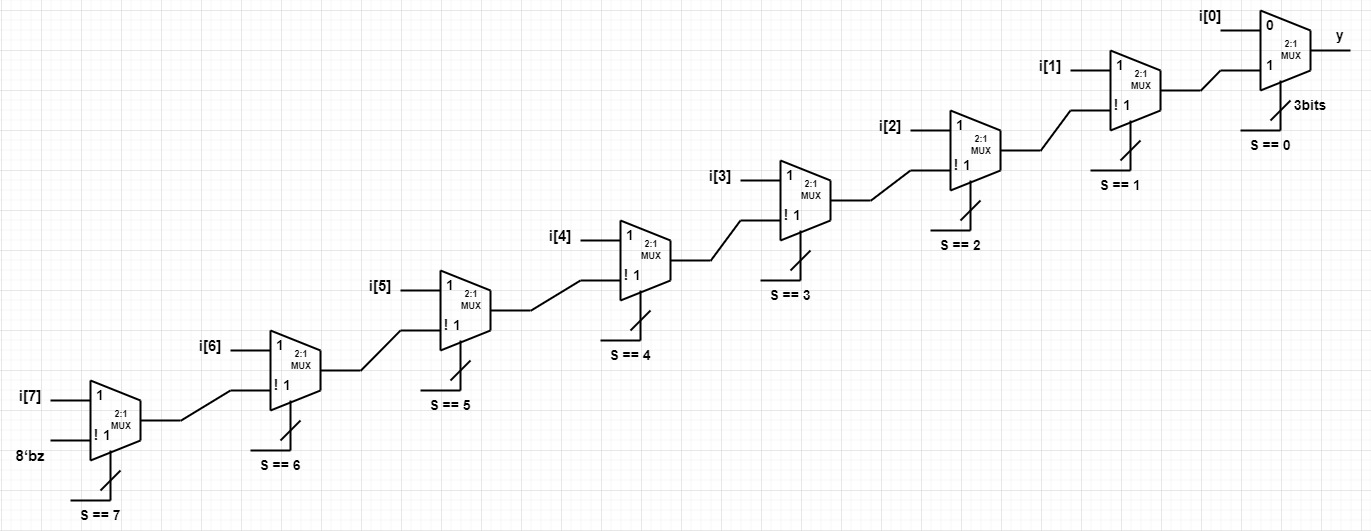
else if (sel == 3'd6) y = i[6];

else if (sel == 3'd7) y = i[7];

else y == 8’bz

end

endmodule



* **CaseMux8**

module CaseMux8 (y, i, sel);

output y;

input [7:0] i;

input [2:0] sel;

reg y;

wire [7:0] i;

wire [2:0] sel;

always @(i or sel)begin

case(sel)

3'd0: y = i[0];

3'd1: y = i[1];

3'd2: y = i[2];

3'd3: y = i[3];

3'd4: y = i[4];

3'd5: y = i[5];

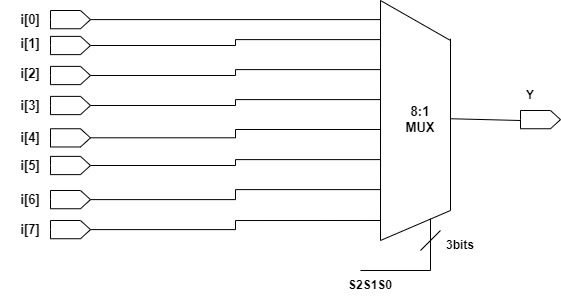
3'd6: y = i[6];

3'd7: y = i[7];

endcase

end

endmodule



Conclusion: IfMux8 and CaseMux8 are logically the same. However, their timing paths (delays) are not the same. The delay from i[1] to the output Y in the case mux, as shown in the above image, is the same as the delay from i[7] to Y. Whereas in the case of IfMux8, the timing path or delay is a "critical path" with different lengths from i[0] through i[7], which results in more layers of Mux and a longer delay time. Since timing is a key factor to consider when creating a branch structure, CaseMux8 would be the best method to use.