**Course Code: EE461**

**Assignment**

**PREPARED BY**

**Khandoker Samiul Hoque**

**Student Id: 19837**

**1 No Answer:**

Design:  
module bit\_detect\_rtl (

input [7:0] data\_input,

output zeroflag\_rtl,

output oneflag\_rtl

);

assign zeroflag\_rtl = ~|data\_input;

assign oneflag\_rtl = |data\_input;

endmodule  
  
test bench :

module tb;

reg [7:0] data\_input;

wire zeroflag\_rtl, oneflag\_rtl, zeroflag\_gate, oneflag\_gate;

bit\_detect\_rtl foo\_rtl (.data\_input(data\_input),.zeroflag\_rtl(zeroflag\_rtl),.oneflag\_rtl(oneflag\_rtl));

reg clk = 0;

always

#5

clk = ~clk;

initial begin

data\_input = 8'b00100100;

#5;

data\_input = 8'b10000001;

#5;

data\_input = 8'b00001001;

#5;

$finish;

end

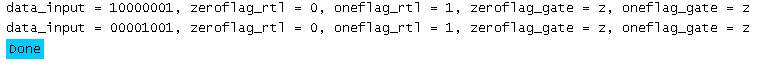
always @(posedge clk) begin

$display("data\_input = %b, zeroflag\_rtl = %b, oneflag\_rtl = %b, zeroflag\_gate = %b, oneflag\_gate = %b",data\_input, zeroflag\_rtl, oneflag\_rtl, zeroflag\_gate, oneflag\_gate);

end

endmodule

output:



design:

module detect\_gate (input [7:0] data\_in,output zeroflag\_gate,output oneflag\_gate);

wire [7:0] not\_data\_in;

wire [7:0] or\_data\_in;

assign not\_data\_in = ~data\_in;

assign or\_data\_in = |data\_in;

and gate1 (zeroflag\_gate, not\_data\_in);

and gate2 (oneflag\_gate, or\_data\_in);

endmodule

testbench:

module tb;

reg [7:0] data\_in;

wire zeroflag\_rtl, oneflag\_rtl, zeroflag\_gate, oneflag\_gate;

detect\_gate foo\_gate (

.data\_in(data\_in),

.zeroflag\_gate(zeroflag\_gate),

.oneflag\_gate(oneflag\_gate)

);

reg clk = 0;

always

#10

clk = ~clk;

initial begin

data\_in = 8'b00100100;

#20;

data\_in = 8'b10000001;

#20;

data\_in = 8'b00001001;

#20;

$finish;

end

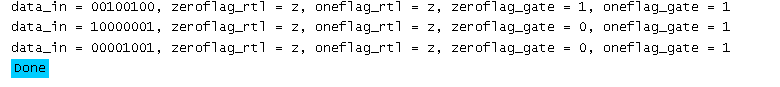
always @(posedge clk) begin

$display("data\_in = %b, zeroflag\_rtl = %b, oneflag\_rtl = %b, zeroflag\_gate = %b, oneflag\_gate = %b", data\_in, zeroflag\_rtl, oneflag\_rtl, zeroflag\_gate, oneflag\_gate);

end

endmodule

output:



When tested with the same input patterns, the bit detector module's RTL and gate level implementations provide different outputs for the zeroflag and oneflag signals.The RTL implementation's zeroflag rtl and oneflag rtl signals have a value of 0 for every one of the three input patterns (00100100, 10000001, and 00001001).For each of the three input patterns in the gate level integration, the zeroflag gate and oneflag gate signals are set to 1.

**2 No Answer:**

**Design:**module number\_converter (input [3:0] x\_input,output [3:0] x\_output);

wire sign\_bit = x\_input[3];

wire [2:0]

magnitude\_bits = sign\_bit ? ~x\_input[2:0] + 1 : x\_input[2:0];

assign x\_output = {sign\_bit, magnitude\_bits};

endmodule  
  
**TestBench:**module tb;

reg [3:0] x\_input;

wire [3:0] x\_output;

number\_converter prob2(.x\_input(x\_input),.x\_output(x\_output));

initial begin

$monitor("x\_input = %b, x\_output = %b", x\_input, x\_output);

x\_input = 4'b0101;

#5 // Test unsigned positive input

x\_input = 4'b1011;

#5 // Test unsigned negative input

x\_input = 4'b0101;

#5

x\_input = 4'b1101;

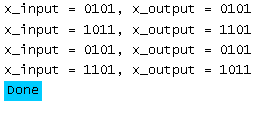
#5

$finish;

end

endmodule

**Output:**



**3 No Answer:**

**Design:**

module mux4\_1\_cont\_assign (input [3:0] data\_input,input [1:0] sel, output wire out);

assign out = (sel == 2'b00) ? data\_input[0] :

(sel == 2'b01) ? data\_input[1]:

(sel == 2'b10) ? data\_input[2] :

data\_input[3];

endmodule

**TestBench:**module tb;

reg [3:0] data\_input;

reg [1:0] sel;

wire out;

mux4\_1\_cont\_assign prob3(.data\_input(data\_input), .sel(sel), .out(out));

initial begin

$dumpfile("tb.vcd");

$dumpvars(0,tb);

// Test case 1

data\_input = 4'b0001;

sel = 2'b00;

$display("Test case 1: data\_input=%b, sel=%b, out=%b", data\_input, sel, out);

#10

// Test case 2

data\_input = 4'b0010;

sel = 2'b01;

$display("Test case 2: data\_input=%b, sel=%b, out=%b", data\_input, sel, out);

#10

// Test case 3

data\_input = 4'b0100;

sel = 2'b10;

$display("Test case 3: data\_input=%b, sel=%b, out=%b", data\_input, sel, out);

#10

// Test case 4

data\_input = 4'b1000;

sel = 2'b11;

$display("Test case 4: data\_input=%b, sel=%b, out=%b", data\_input, sel, out);

#10

//Test case5

sel = 2'b01;

data\_input = 4'bz;

$display("Test case 5: data\_input=%b, sel=%b, out=%b", data\_input, sel, out);

#10

//Test case 6

sel = 2'b01;

data\_input = 4'bx;

$display("Test case 6: data\_input=%b, sel=%b, out=%b", data\_input, sel, out);

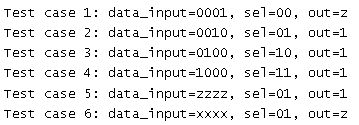
#10

$finish();

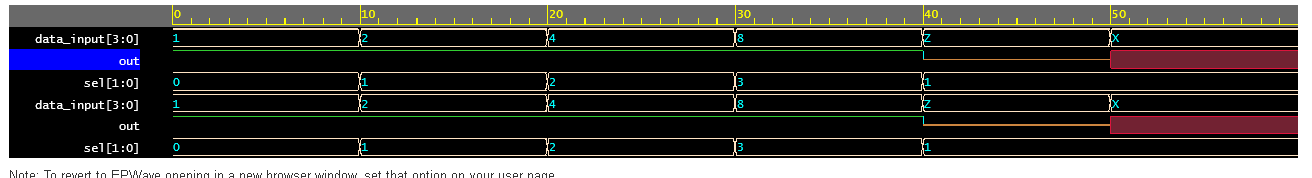
end

endmodule

**Output:**

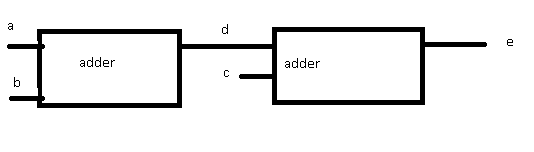


**Graph Output:**

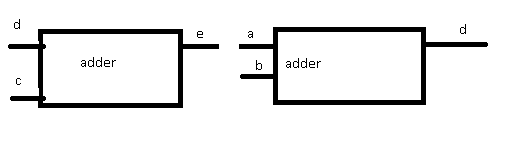


**4 No Answer:**

a.



b.



Even if the identical inputs 'a', 'b', and 'c' are used, the two code snippets will output different values for 'd' and 'e' because of the variations in the sequence of assignments and the adders order.

**5 No Answer:**

module mux(a, b, c, d, e, sel, out);

input a, b, c, d, e;

input [2:0] sel;

output reg out;

always @(\*) begin

case(sel)

3'b000: out = a;

3'b001: out = b;

3'b010: out = c;

3'b011: out = d;

3'b100: out = e;

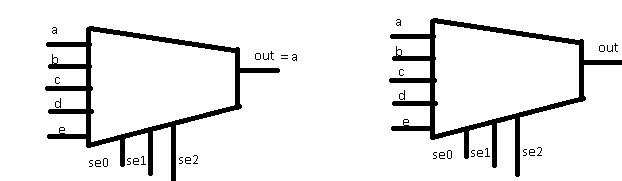
default: out = a; // assign a default value to avoid inferred latches

endcase

end

endmodule

After assigning a value to the out by default, which will make this to avoid, inferred latches.



New old