Cairo University

Faculty of Computers and Artificial Intelligence

CS322

Computer Architecture and Organization

Assignment 2

Instructor: Dr. Mohammad El-Ramly

Team members:

Abdelrahman Nasr Abdelsalam Ali **20170343**

Bahaa El Deen Osama Sayed **20170078**

Task 2

Problem 3.31 = 4.42 (Bahaa El-Deen Osama):

State Transition Table

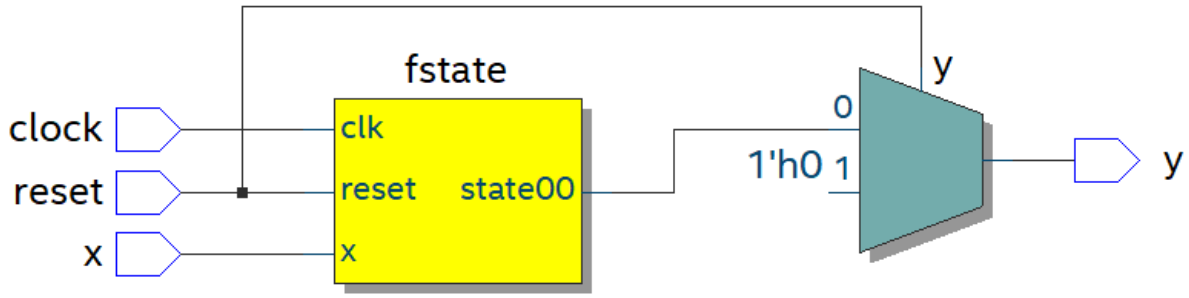
Current State	Input	Next State
S00	0	S01
S00	1	S11
S01	0	S00
S01	1	S10
S10	0	S10
S10	1	S01
S11	0	S11
S11	1	S01

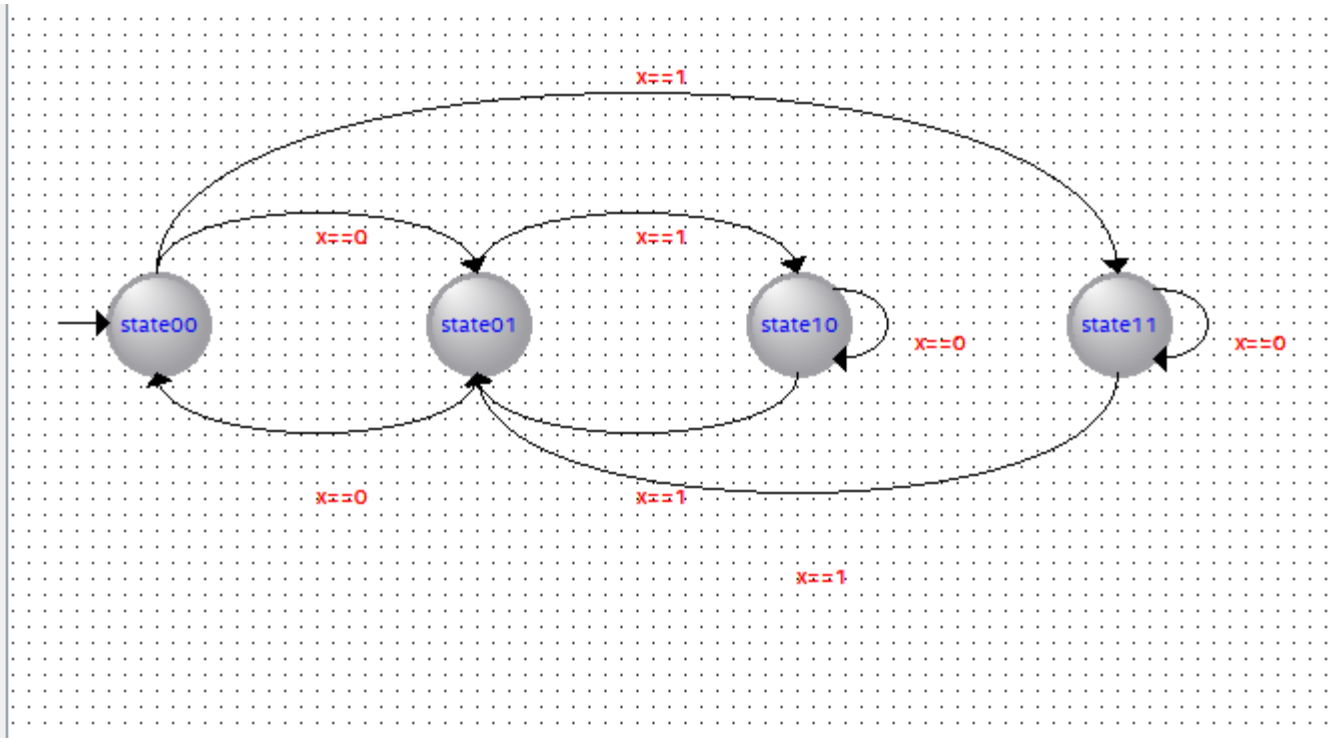
Output Table

State	Output
S00	0
S01	1
S1X	1

```
module FSM4_42(input logic clk, input logic  reset, input logic  x, output logic y);
        typedef enum logic [1:0] {S00, S01, S10, S11} statetype;
        statetype [1:0] state, nextstate;
        //State register
        always_ff@(posedge clk ,posedge reset)
                        if(reset) state <= S00;
                        else              state <= nextstate;
        // next state logic
        always_comb
                case(state)
                        S00: if(x) nextstate = S11;
                                else   nextstate=S01;
                        S01: if(x) nextstate=S10;
                                else   nextstate=S00;
                        S10: if(x)  nextstate=S01;
                                else   nextstate=S10;
                        S11: if(x)  nextstate=S01;
                                else nextstate=S11;
                        default: nextstate=S00;
                endcase
 //output logic
        assign y = (state==S11 )  | (state==S01) | (state==S10);
endmodule
```

The schematic diagram *(resulting from synthesizing the code on Quartus)*:





Note: **The output is TRUE only and only if the nextstate S01 or S1X (X is don't care)**

Problem 3.32 (Abdelrahman Nasr):

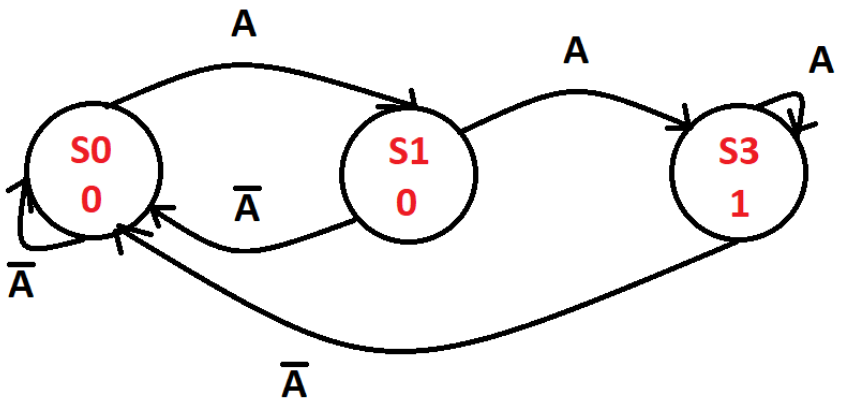
State Transition Table

Current State	Input	Next State
S0	0	S0
S0	1	S1
S1	0	S0
S1	1	S2
S2	0	S0
S2	1	S2

Output Table

State	Input
S0	0
S1	0
S2	1

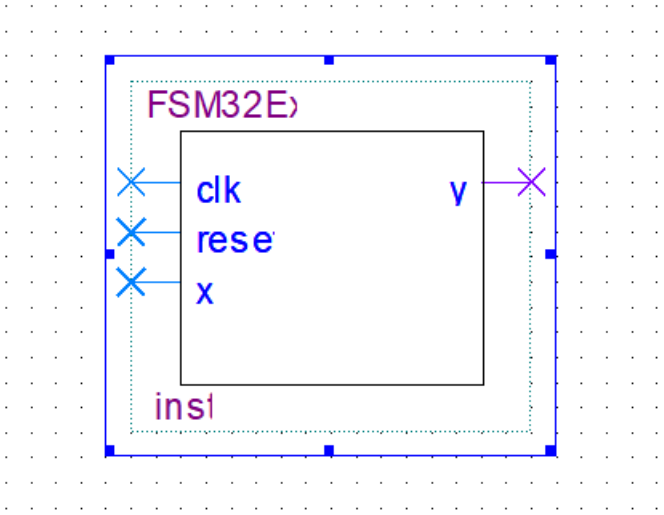
State Transition Diagram



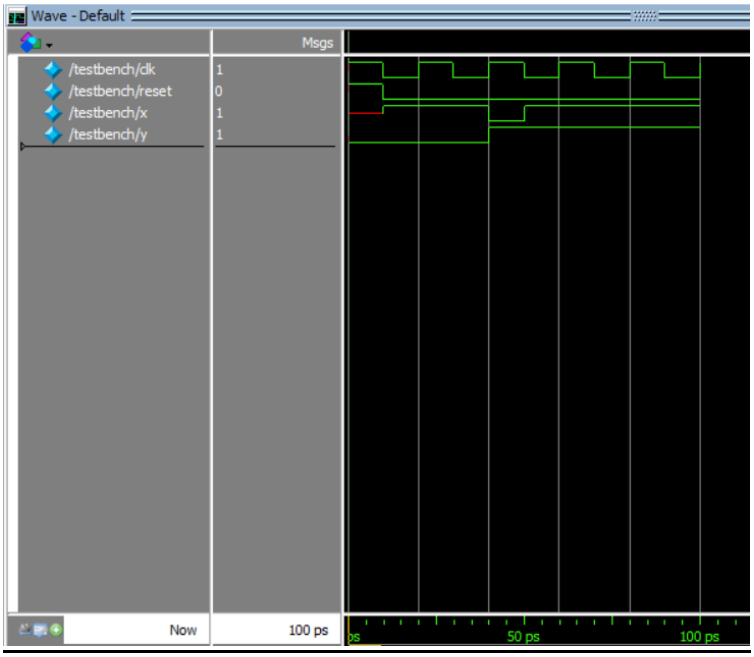
```
module FSM32Ex(input logic clk, input logic reset, input logic x, output logic y);
	typedef enum logic [1:0] {S0, S1, S2} statetype;
	statetype state, nextstate;
	// state register
	always_ff @(posedge clk, posedge reset)
		if (reset) state <= S0;
		else state <= nextstate;
	// next state logic
	always_comb
		case (state)
			S0:	if (x)  nextstate = S1;
				else	nextstate = S0;
			S1:	if (x)  nextstate = S2;
				else	nextstate = S0;
			S2:	if (x)  nextstate = S2;
				else	nextstate = S0;
			default: nextstate <= S0;
		endcase
	// output logic
	assign y = (state == S2);
endmodule
```

<u>Description:</u> The FSM output is TRUE whenever the input is TRUE followed by another TRUE input and FALSE otherwise.

The schematic diagram *(resulting from synthesizing the code on Quartus)*:



The simulation results on ModelSim:

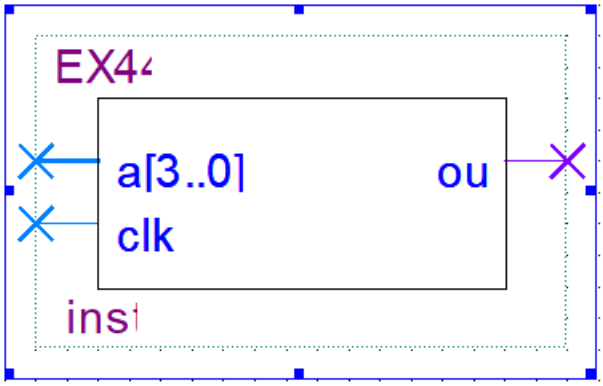


Problem 4.44 (Abdelrahman Nasr):

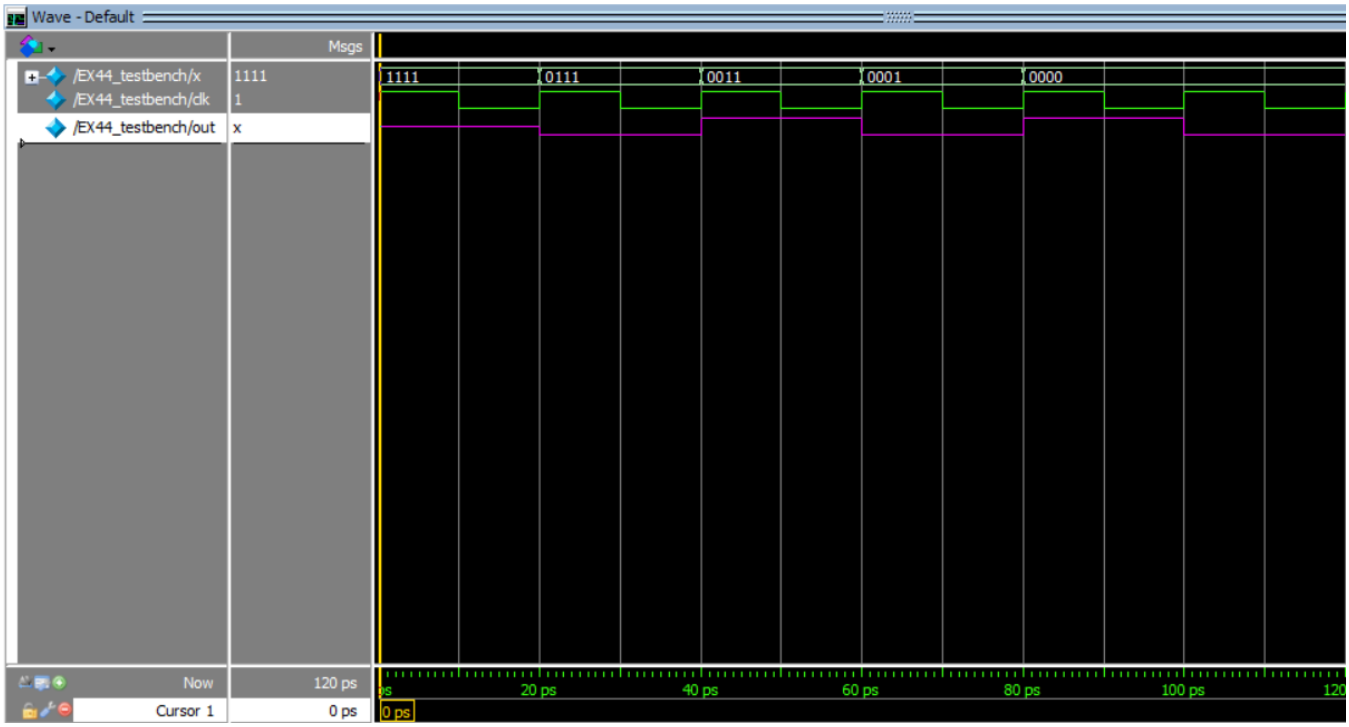
```
module register(input logic in, input logic clock, output logic out);
always_ff @(posedge clock)
        out <= in;
endmodule


module EX44(input logic [3:0] a, input logic clk, output logic out);
        logic [3:0] b;
        logic y;
        // inputs to registers
        register r1(a[3], clk, b[3]);
        register r2(a[2], clk, b[2]);
        register r3(a[1], clk, b[1]);
        register r4(a[0], clk, b[0]);
        // XORing
        assign y = ((b[3] ^ b[2]) ^ b[1]) ^ b[0];
        // storing output
        register rout(y, clk, out);
endmodule
```

Description: The four inputs are passed first to registers to store their values, then all of them are XORed using 3 XOR gates and storing the output in a register too.

The schematic diagram *(resulting from synthesizing the code on Quartus)*:



The simulation results on ModelSim:



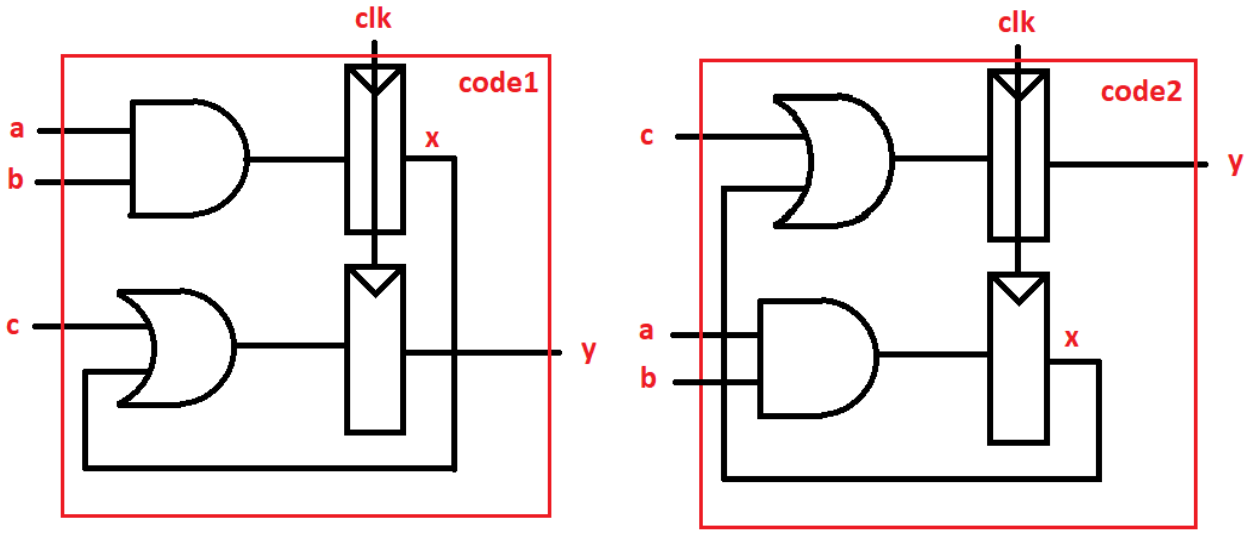
Note: **The output is TRUE only and only if there are odd number of TRUE inputs and false otherwise**

Problem 4.46 (Bahaa EL-Deen Osama):

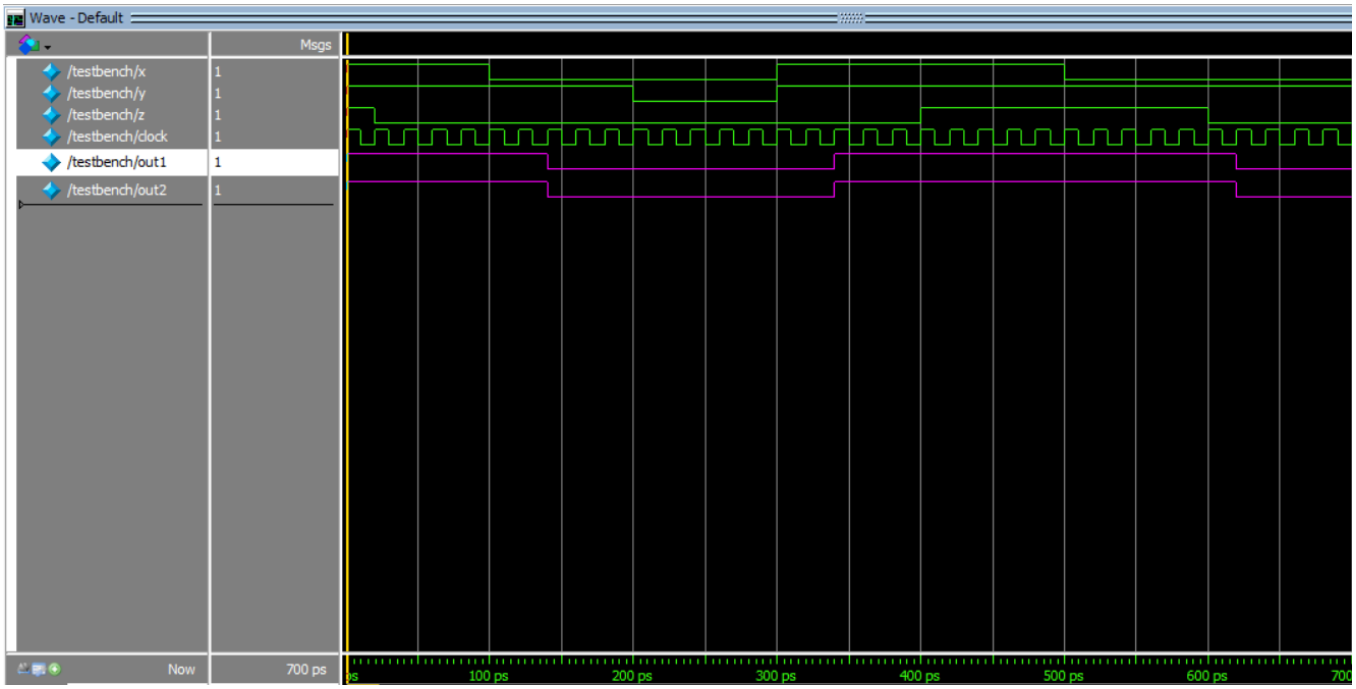
There can be several drivers for a signal declared as tri.

Problem 4.48 (Abdelrahman Nasr):

Yes, they both have the same functionality



Testbench simulation for different values:



Problem 4.50 (Bahaa El-Deen Osama):

(a)

```
module latch(input logic clk ,input logic [3:0] d, output reg [3:0] q);
        always @(clk)
                if (clk) q <= d;
endmodule
```

Errors:

1. Output reg[3:0]q must be output logic reg[3:0].
2. Always @(clk) must be always_latch.

Fixed Code:

```
module latch(input logic clk ,input logic [3:0] d, output logic reg [3:0] q);
        always_latch
                if (clk) q <= d;
endmodule
```

(b)

```
module gates(input logic [3:0] a, b, output logic [3:0] y1, y2, y3, y4, y5);
        always @(a)
        begin
        y1 =a & b;
        y2 =a | b;
        y3 =a ^ b;
        y4 = ~(a & b);
        y5 = ~(a | b);
        end
endmodule
```

Error: always @(a) must be always_comb .

Fixed Code:

```
module gates(input logic [3:0] a, b, output logic [3:0] y1, y2, y3, y4, y5);
        always_comb
        begin
```

```
        y1 =a & b;
        y2 =a | b;
        y3 =a ^ b;
        y4 = ~(a & b);
        y5 = ~(a | b);
        end
endmodule
```

(c)

```
module mux2(input logic [3:0] d0, d1, input logic s, output logic [3:0] y);
        always @(posedge s)
        if (s) y <= d1;
        else y <= d0;
endmodule
```

Error: always @(posedge s) must be always_comb

Fixed Code:

```
module mux2(input logic [3:0] d0, d1, input logic s, output logic [3:0] y);
        always_comb
        if (s)  y = d1;
        else   y = d0;
endmodule
```

(d)

```
module twoflops(input logic clk, input logic d0, d1, output logic q0, q1);
        always @(posedge clk)
        q1 = d1;
        q0 = d0;
endmodule
```

Error: always@(posedge clk) must be always_ff@(posedge clk)

Fixed Code:

```
module twoflops(input logic clk , input logic d0, d1, output logic q0, q1);
        always_ff @(posedge clk)
        begin
        q1 <= d1;
        q0 <= d0;
```

```
        end
endmodule
```

(e)

```
module FSM(input logic clk, input logic a, output logic out1, out2);
        logic state;
        // next state logic and register (sequential)
        always_ff @(posedge clk)
                if (state == 0) begin
                if (a) state <= 1;
        end else begin
        if (~a) state <= 0;
        end
        always_comb
        // output logic (combinational)
        if (state == 0) out1 = 1;
        else out2 = 1;
endmodule
```

Error: module messing state logic and input declaration.

Fixed Code:

```
module FSM(input logic clk, input logic reset ,input logic a, output logic out1, out2);
        logic state ,nextstate;

        //  state register
        always_ff @(posedge clk , posedge reset)
           if (reset)    state  <= 1'b0;                    //first state
           else          state  <= nextstate;               //second state

        // next state logic
        always_comb
        case(state)
                1'b0 :          if (a)    nextstate =1'b1;
                                else      nextstate = 1'b0;
                1'b1:           if(a)      nextstate =1'b1;
                                else       nextstate  =1'b0;

        // output logic (combinational)
        Always_comb
        if (state == 0)   {out1,out2}    =  {1b'1  , 1'b0};
        else              {out1,out2}    =  { 1b'0 ,1'b1};
endmodule
```

Problem 4.50 (Abdelrahman Nasr):

(f)

```
module priority(input logic [3:0] a, output logic [3:0] y);
        always_comb
                if (a[3]) y = 4'b1000;
                else if (a[2]) y = 4'b0100;
                else if (a[1]) y = 4'b0010;
                else if (a[0]) y = 4'b0001;
endmodule
```

Errors:

3. priority is a reserved keyword on SystemVerilog
4. There must be a default statement for the if condition to prevent conflicts

Fixed Code:

```
module priority_check(input logic [3:0] a, output logic [3:0] y);
        always_comb
                if (a[3])   y = 4'b1000;
                else if (a[2]) y = 4'b0100;
                else if (a[1]) y = 4'b0010;
                else if (a[0]) y = 4'b0001;
                else       y = 4'b0000;
endmodule
```

(g)

```
module divideby3FSM(input logic clk, input logic reset, output logic out);
        logic [1:0] state, nextstate;
        parameter S0 = 2'b00;
        parameter S1 = 2'b01;
        parameter S2 = 2'b10;
        // State Register
        always_ff @(posedge clk, posedge reset)
                if (reset) state <= S0;
                else state <= nextstate;
        // Next State Logic
        always @(state)
                case (state)
                        S0: nextstate = S1;
                        S1: nextstate = S2;
                        S2: nextstate = S0;
                endcase
        // Output Logic
```

```
        assign out = (state == S2);
endmodule
```

Errors:

1. always @(state) should be always_comb to avoid missing signals in the sensitivity list
2. the case statement is missing the default case

Fixed Code:

```
module divideby3FSM(input logic clk, input logic reset, output logic out);
        logic [1:0] state, nextstate;
        parameter S0 = 2'b00;
        parameter S1 = 2'b01;
        parameter S2 = 2'b10;
        // State Register
        always_ff @(posedge clk, posedge reset)
                if (reset) state <= S0;
                else state <= nextstate;
        // Next State Logic
        always_comb
                case (state)
                        S0: nextstate = S1;
                        S1: nextstate = S2;
                        S2: nextstate = S0;
                        default nextstate = S0;
                endcase
        // Output Logic
        assign out = (state == S2);
endmodule
```

(h)

```
module mux2tri(input logic [3:0] d0, d1, input logic s, output tri [3:0] y);
        tristate t0(d0, s, y);
        tristate t1(d1, s, y);
endmodule
```

Error: It's a logical error because s parameter of t0 instance should be passed inverted

Fixed Code:

```
module mux2tri(input logic [3:0] d0, d1, input logic s, output tri [3:0] y);
        tristate t0(d0, ~s, y);
        tristate t1(d1, s, y);
endmodule
```

(i)

```
module floprsen(input logic clk, input logic reset, input logic set, input logic [3:0] d, output logic [3:0] q);
        always_ff @(posedge clk, posedge reset)
                if (reset) q <= 0;
                else q <= d;
        always @(set)
                if (set) q <= 1;
endmodule
```

Error: The output is assigned multiple times which cause conflicts to the output's signal

Fixed Code:

```
module floprsen(input logic clk, input logic reset, input logic set, input logic [3:0] d, output logic [3:0] q);
        always_ff @(posedge clk, posedge reset, posedge set)
                if (reset) q <= 0;
                else if (set) q<=1;
                else q <= d;
endmodule
```

(j)

```
module and3(input logic a, b, c, output logic y);
        logic tmp;
        always @(a, b, c)
        begin
                tmp <= a & b;
                y <= tmp & c;
        end
endmodule
```

Errors:

1. always_comb should be used instead of always @(a, b, c) to avoid missing signals in the sensitivity list
2. The blocking assignment should be used with always_comb

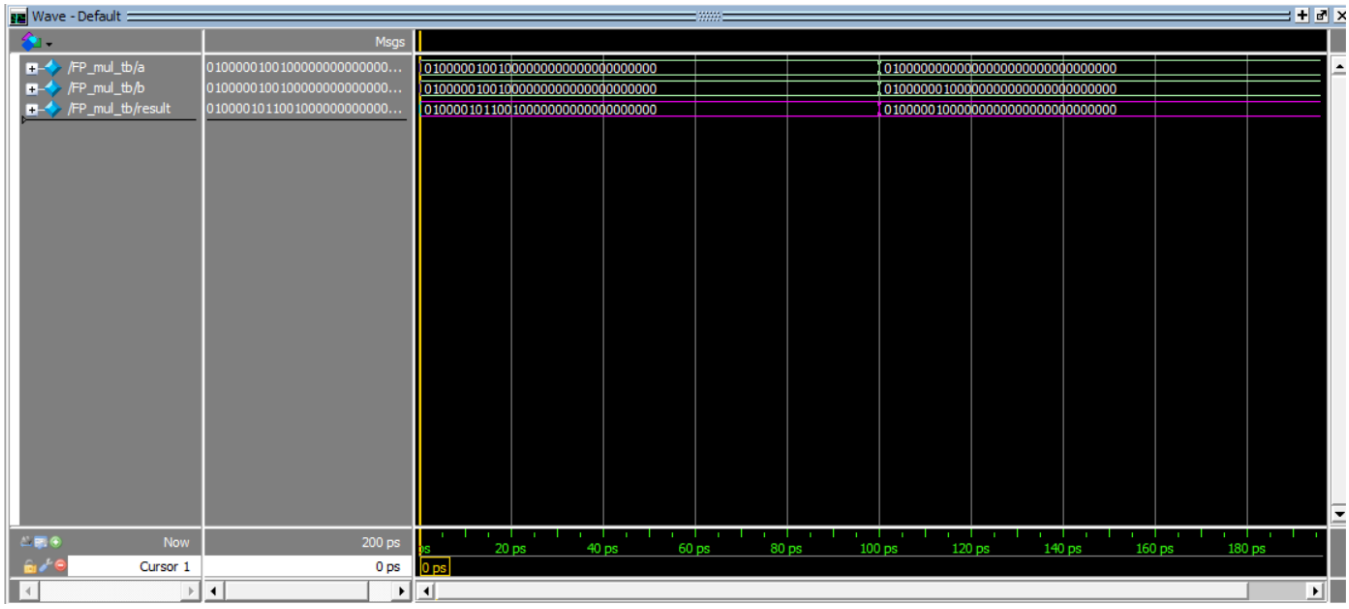
Fixed Code:

```
module and3(input logic a, b, c, output logic y);
        logic tmp;
        always_comb
        begin
                tmp = a & b;
                y = tmp & c;
        end
endmodule
```

Task 3 – FP Multiplier

```
module FP_mul(input logic [31:0] x, y, output logic [31:0] m);
	// internal variables
	logic [23:0] mantx, manty;
	logic [7:0] exponent, expx, expy;
	logic [22:0] fraction;
	logic [47:0] result;
	// extracting to exponent and mantisa
	assign expx  = x[30:23];
	assign mantx = {1'b1, x[22:0]};
	assign expy  = y[30:23];
	assign manty = {1'b1, y[22:0]};
	// multiplying mantisas and calculating the new fraction and exponent
	assign result = mantx * manty;
	assign fraction = result[47] ? result[46:24] : result[45:23];
	assign exponent = result[47] ? (expx + expy - 126) : (expx + expy - 127);
	// the result of multiplication
	assign m = {1'b0, exponent, fraction};
endmodule
```

Simulation:

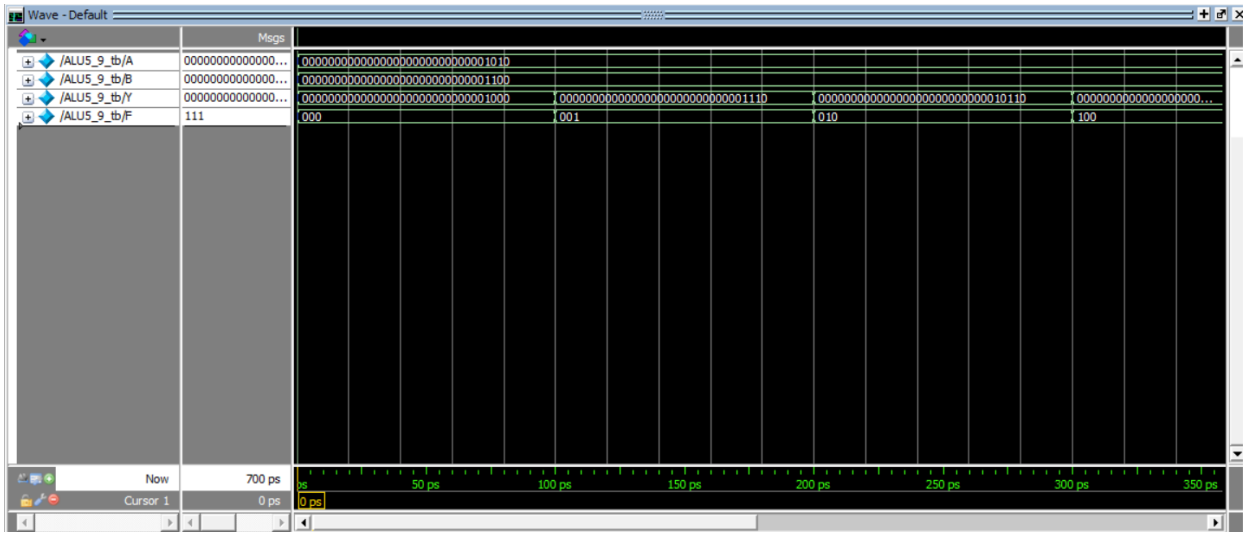


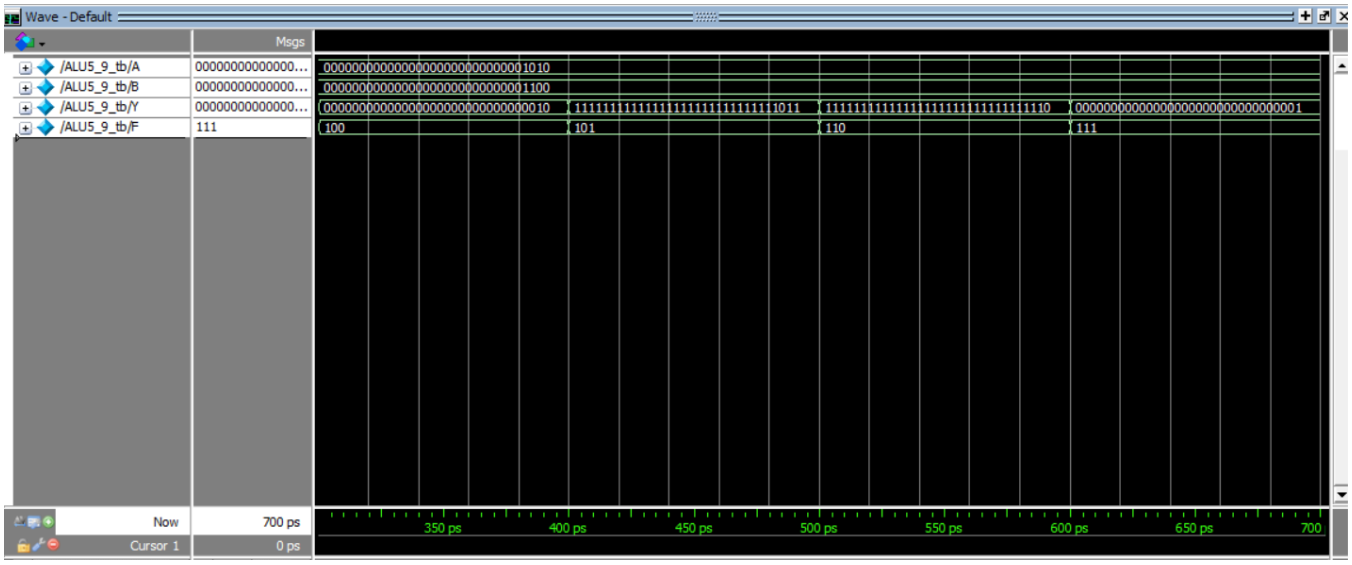
Task 4 - ALU

Problem 5.9:

```
module ALU5_9(input logic [31:0]a, input logic [31:0]b, input logic [2:0]f, output logic
[31:0]y);

    // ALU operations:
    // ---------------
    // 000 = A AND B
    // 001 = A OR B
    // 010 = A + B
    // 011 = Not Used
    // 100 = A AND B'
    // 101 = A OR B'
    // 110 = A - B
    // 111 = SLT

       logic[31:0] sum, B_output;

       assign B_output = (f[2]) ? ~b : b;
       assign sum = a+B_output+f[2];

       always_comb
             case(f[1:0])
          2'b00: y <= a & B_output;       // AND operation
          2'b01: y <= a | B_output;       // OR operation
          2'b10: y <= sum;                // Sumation
          2'b11: y <= sum[31];
          default: y<= sum[31];
             endcase

endmodule
```

Simulation:



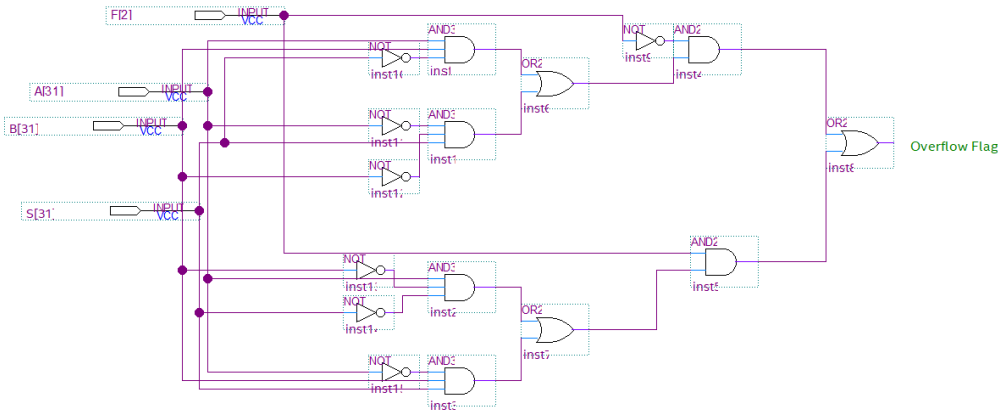


Problem 5.10:

(a) The overflow equation:

~f[2] & (a & b & ~s[31] | ~a & ~b & s[31]) | f[2] & (~a & b & s[31] | a & ~b & ~s[31]);

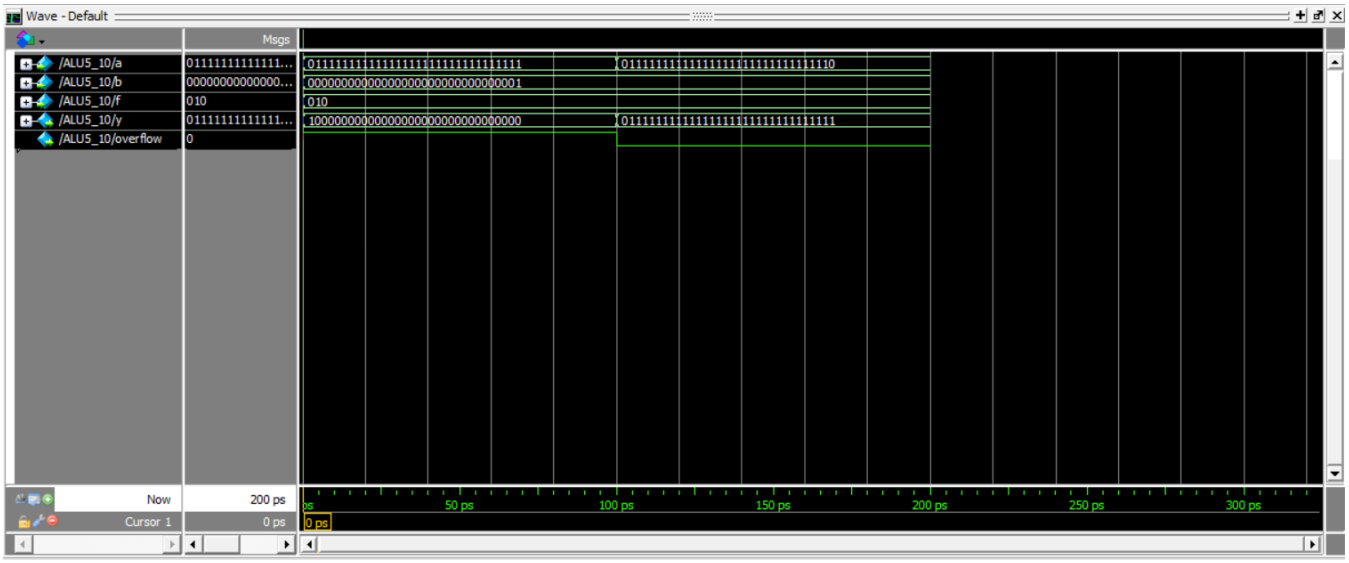
(b) Circuit Diagram:




```
module ALU5_10(input logic [31:0]a, input logic [31:0]b, input logic [2:0]f, output logic
[31:0]y, output logic overflow);

    // ALU operations:
    // ---------------
    // 000 = A AND B
    // 001 = A OR B
    // 010 = A + B
    // 011 = Not Used
    // 100 = A AND B'
    // 101 = A OR B'
    // 110 = A - B
    // 111 = SLT

       logic[31:0] sum, B_output;
```

```
        assign B_output = (f[2]) ? ~b : b;
        assign sum = a+B_output+f[2];

        always_comb
              case(f[1:0])
           2'b00: y <= a & B_output;       // AND operation
           2'b01: y <= a | B_output;       // OR operation
           2'b10: y <= sum;                // Sumation
           2'b11: y <= sum[31];
           default: y<= sum[31];
              endcase

        always_comb
              case (f[2])
                    1'b0: overflow = a[31] & b[31] & ~sum[31] | ~a[31] & ~b[31] & sum[31];
                    1'b1: overflow = ~a[31] & b[31] & sum[31] | a[31] & ~b[31] & ~sum[31];
                    default: overflow = 0;
              endcase

endmodule
```

Simulation:

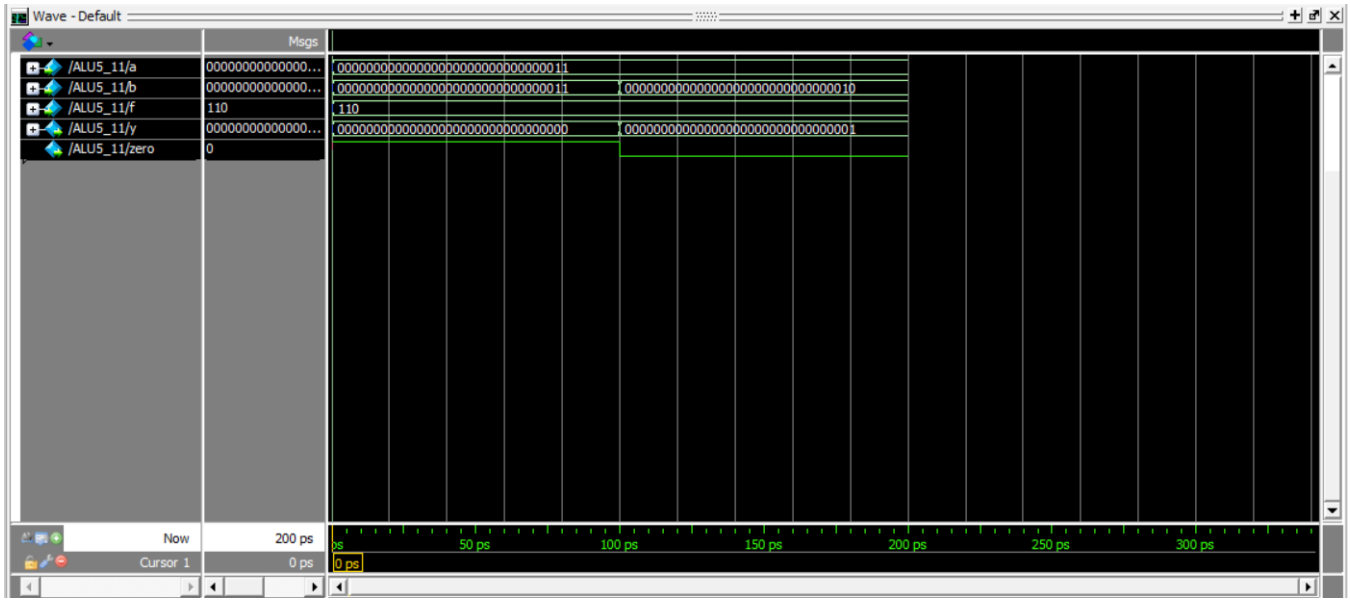


Problem 5.11:

```
module ALU5_11(input logic [31:0]a, input logic [31:0]b, input logic [2:0]f, output logic
[31:0]y, output logic zero);

    // ALU operations:
    // ---------------
    // 000 = A AND B
    // 001 = A OR B
    // 010 = A + B
```

```
    // 011 = Not Used
    // 100 = A AND B'
    // 101 = A OR B'
    // 110 = A - B
    // 111 = SLT

       logic[31:0] sum, B_output;

       assign B_output = (f[2]) ? ~b : b;
       assign sum = a+B_output+f[2];

       always_comb
             case(f[1:0])
          2'b00: y <= a & B_output;      // AND operation
          2'b01: y <= a | B_output;      // OR operation
          2'b10: y <= sum;               // Sumation
          2'b11: y <= sum[31];
          default: y<= sum[31];
             endcase

       assign zero = (y==32'b0);

endmodule
```

Simulation:



Problem 5.12 (Testbenches):

```
module ALU5_9_tb();

       logic[31:0] A, B, Y;
       logic[2:0] F;

       // instantiate device under test
```

```
	ALU5_9 dut(A, B, F, Y);

	initial begin
		A<=32'b00000000000000000000000000001010;
		B<=32'b00000000000000000000000000001100;

		F<=3'b000; #100;
		F<=3'b001; #100;
		F<=3'b010; #100;

		F<=3'b100; #100;
		F<=3'b101; #100;
		F<=3'b110; #100;
		F<=3'b111; #100;
	end

endmodule


module ALU5_10_tb();

	logic[31:0] A, B, Y;
	logic[2:0] F;

	// Flags
	logic VF;

	// instantiate device under test
	ALU5_11 dut(A, B, F, Y, VF);

	initial begin
		A<=32'b01111111111111111111111111111111;
		B<=32'b00000000000000000000000000000001;
		F<=3'b010; #100;

		A<=32'b01111111111111111111111111111110;
		F<=3'b010; #100;
	end

endmodule


module ALU5_11_tb();

	logic[31:0] A, B, Y;
	logic[2:0] F;

	// Flags
	logic ZF;

	// instantiate device under test
	ALU5_11 dut(A, B, F, Y, ZF);

	initial begin
		A<=32'b00000000000000000000000000000011;
		B<=32'b00000000000000000000000000000011;
		F<=3'b110; #100;

		B<=32'b00000000000000000000000000000010;
		F<=3'b110; #100;
	end

endmodule
```