Cairo University
Faculty of Computers and Information

CS331

Assignment 2

Computer organization And architecture

7th Dec 2020

Project Team

Esraa AbuBaker Mubarek 20180040

Marina Moheb Nafee 20180208

Task 2 – Problem Solving

Implement SystemVerilog design for FSM of problem 3.31 [Esraa]

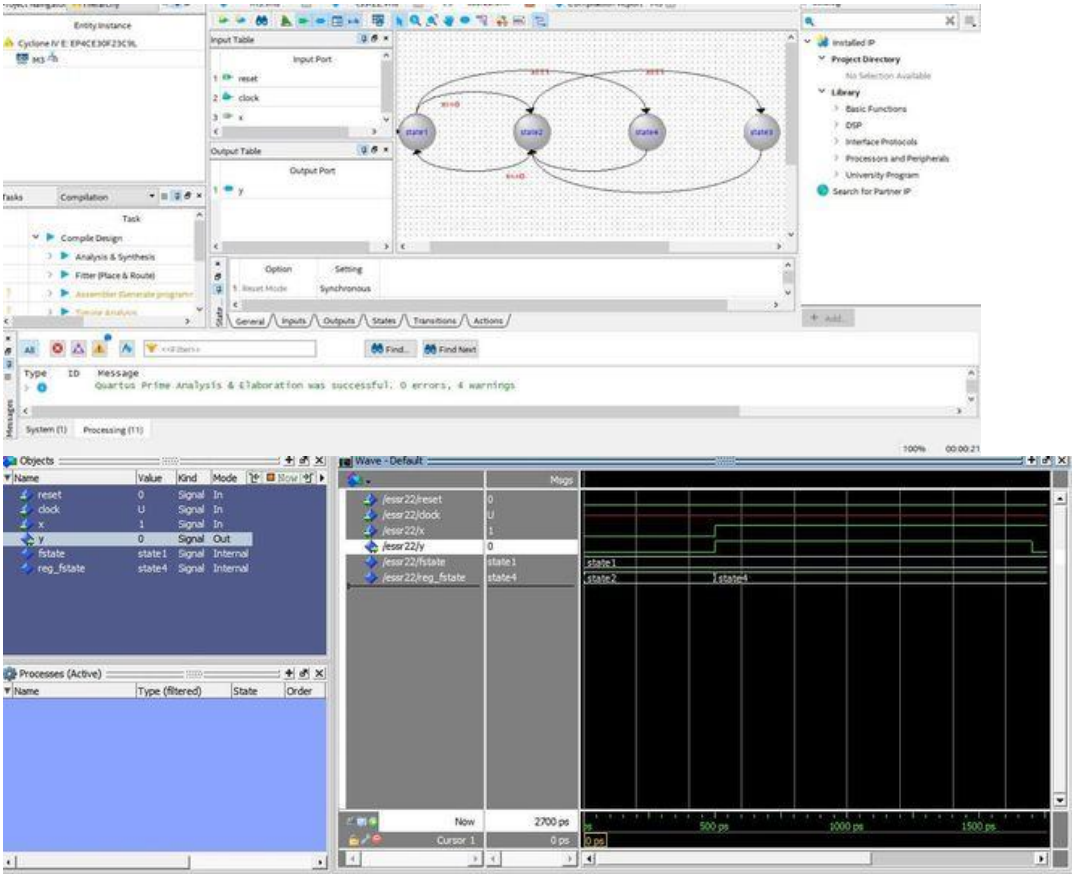
```
module sequanceDetector(q,clk,Reset,x);
output reg q;
input clk,Reset,x;
statetype [1:0] currentstate;
statetype [1:0]nextstate;
typedef enum logic { A=2'b00,B=2'b01,C=2'b10,D=2'b11} statetype;
always@(x,currentstate)
begin
q=(currentstate[0])|(currentstate[1]);
case(currentstate)
A:if(x==0) begin nextstate= B;end
     else begin nextstate= D;end
B:if(x==0) begin nextstate= A;end
     else begin nextstate=C ;end
C:if(x==0|x==1) begin nextstate= B;end
D:if(x==0|x==1) begin nextstate= B;end
endcase
end
always@(posedge clk)
begin
if(~Reset)
currentstate <= nextstate;
else
currentstate =A;
end
endmodule
```

another solution

```
module essr22 (
   input reset, input clock, input x,
   output y);

   enum int unsigned { state1=0, state3=1, state4=2, state2=3 } fstate, reg_fstate;

   always_ff @(posedge clock)
   begin
      if (clock) begin
```

```
      fstate <= reg_fstate;
   end
end

always_comb begin
   if (reset) begin
      reg_fstate <= state1;
      y <= 1'b0;
   end
   else begin
      y <= 1'b0;
      case (fstate)
         state1: begin
            if ((x == 1'b0))
               reg_fstate <= state2;
            else if ((x == 1'b1))
               reg_fstate <= state4;
            // Inserting 'else' block to prevent latch inference
            else
               reg_fstate <= state1;

            if ((x == 1'b1))
               y <= 1'b1;
            else if ((x == 1'b0))
               y <= 1'b0;
            // Inserting 'else' block to prevent latch inference
            else
               y <= 1'b0;
         end
         state3: begin
            reg_fstate <= state2;
         end
         state4: begin
            reg_fstate <= state2;

            y <= 1'b1;
         end
         state2: begin
            if ((x == 1'b1))
               reg_fstate <= state3;
            else if ((x == 1'b0))
               reg_fstate <= state1;
```

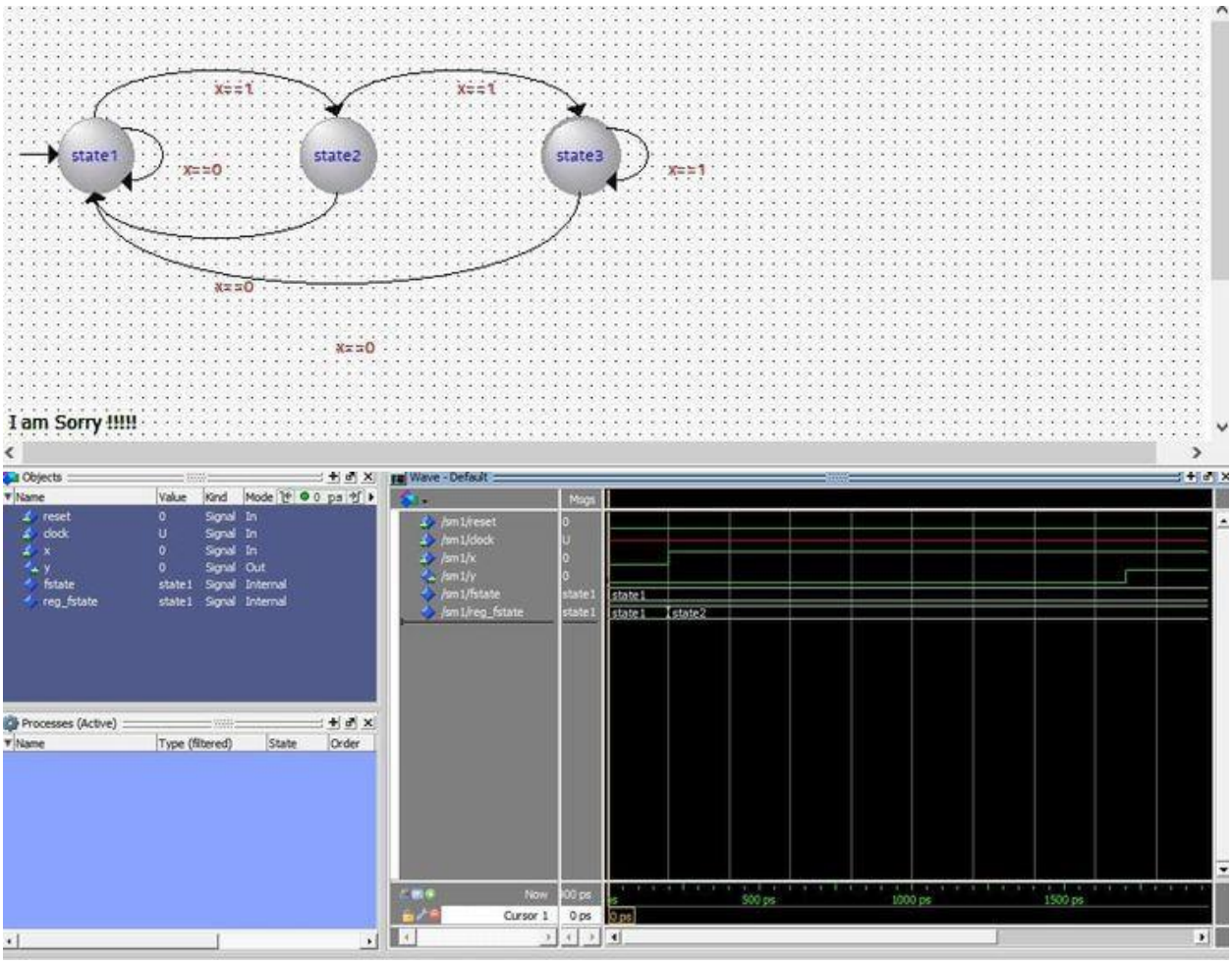
```
            // Inserting 'else' block to prevent latch inference
            else
               reg_fstate <= state2;

            if ((x == 1'b1))
               y <= 1'b1;
            // Inserting 'else' block to prevent latch inference
            else
               y <= 1'b0;
         end
         default: begin
            y <= 1'bx;
            $display ("Reach undefined state");
         end
      endcase
   end
end
endmodule
```



Implement SystemVerilog design for FSM of problem 3.32 [Marina]

```
module SM1 (
  input reset, input clock, input x,
  output y);

  enum int unsigned { state1=0, state2=1, state3=2 } fstate, reg_fstate;

  always_ff @(posedge clock)
  begin
    if (clock) begin
      fstate <= reg_fstate;
    end
  end

  always_comb begin
    if (reset) begin
      reg_fstate <= state1;
      y <= 1'b0;
    end
    else begin
      y <= 1'b0;
      case (fstate)
        state1: begin
          if ((x == 1'b0))
            reg_fstate <= state1;
          else if ((x == 1'b1))
            reg_fstate <= state2;
          // Inserting 'else' block to prevent latch inference
          else
            reg_fstate <= state1;
        end
        state2: begin
          if ((x == 1'b1))
            reg_fstate <= state3;
          else if ((x == 1'b0))
            reg_fstate <= state1;
```

```
            // Inserting 'else' block to prevent latch inference
            else
               reg_fstate <= state2;
         end
         state3: begin
            if ((x == 1'b1))
               reg_fstate <= state3;
            else if ((x == 1'b0))
               reg_fstate <= state1;
            // Inserting 'else' block to prevent latch inference
            else
               reg_fstate <= state3;

            if ((x == 1'b1))
               y <= 1'b1;
            // Inserting 'else' block to prevent latch inference
            else
               y <= 1'b0;
         end
         default: begin
            y <= 1'bx;
            $display ("Reach undefined state");
         end
      endcase
   end
 end
endmodule
```



Exercise 4.42 [Esraa]

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity ex4_42 is
 port(clk, reset, x: in STD_LOGIC;
 q: out STD_LOGIC);
end;
architecture synth of ex4_42 is type statetype is (S00, S01, S10, S11);
 signal state, nextstate: statetype;
begin
 process(clk, reset) begin
 if reset then state <= S00;
 elsif rising_edge(clk) then
 state <= nextstate;
 end if;
 end process;
 process(all) begin
```

```
 case state is
 when S00 => if x then
 nextstate <= S11;
else nextstate <= S01;
end if;
 when S01 => if x then
 nextstate <= S10;
else nextstate <= S00;
end if;
 when S10 => nextstate <= S01;
 when S11 => nextstate <= S01;
 when others => nextstate <= S00;
 end case;
 end process;
 q <= '0' when (state = S00) else '1';
end;
```

Exercise 4.44 [Marina]

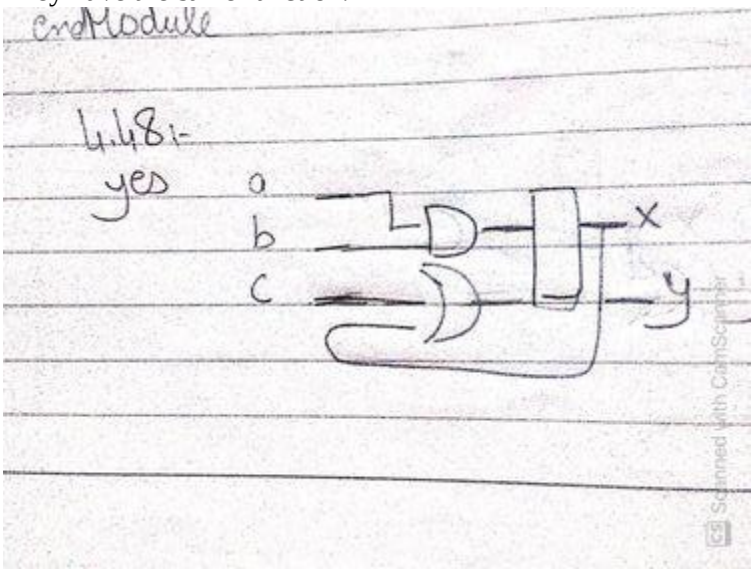
```
module a(input logic clk, a, b, c, d,
output logic q);
logic reg1, reg2, reg3, reg4;
always_ff @(posedge clk)
begin
reg1 <= a;
reg2 <= b;
reg3 <= c;
reg4 <= d;
q <= ((reg1 ^ reg2) ^ reg3) ^ reg4;
end
endmodule

module d (input logic clk, a, b, c, d,
                        output logic q);
  logic reg1, reg2, reg3, reg4;
always_ff @(posedge clk)
begin
  reg1 <= a;
  reg2 <= b;
  reg3 <= c;
  reg4 <= d;
   q <= (reg1 ^ reg2) ^ (reg3 ^ reg4);
end
endmodule
```

Exercise 4.46 [Esraa]

Tristate busses can have multiple drivers, so they should be declared as a net.

Exercise 4.48 [Marina]

They have the same function.



Exercise 4.50 [Esraa]

Changes are in bold

(a) Error : in always statement d not included

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

(b) Error: in always statement b not included.

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

endmodule

(c) Error : always statement needs to respond to any changes in s not just positive edge

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

(d) Error : in always statement , filpflops have a special always statement , error in the equal sign and begin and end statement .

```
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) Error: Reset missing in input declaration ,out1 and out2 aren't assigned for all cases and also it would be best to separate the next state logic from the current state.

```
module FSM(input logic clk,
                    input logic a,
                    input logic Reset,
                  output logic out1, out2);
           logic currentstate,nextstate;
      // currentstate logic
     always_ff @(posedge clk,posedge Reset)
      if (Reset)
         currentstate <=1'b0;
     else
        currentstate <=nextstate;
         // nextstate logic
always_comb
 case(currentstate)
     1'b0: if (a) nextstate = 1'b1;
    else nextstate = 1'b0;
    1'b1: if (~a) nextstate = 1'b1;
    else nextstate = 1'b0;
 endcase
 // output logic (combinational)
```

```
always_comb
 if (currentstate == 0) {out1, out2}={1'b1,1'b0};
 else          { out1, out2}={1'b0,1'b1};
endmodule
```

Exercise 4.50 [Marina]

(f) Error: If must have. Else and it is combinational circuit

```
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) Error: state S2 is missing and default is missing

```
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) Error: two statement have the same value so ~ is missing on the first tristate.

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

(i) Error: q, cannot be assigned in multiple always or assignment statements.

```
module floprs(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
```

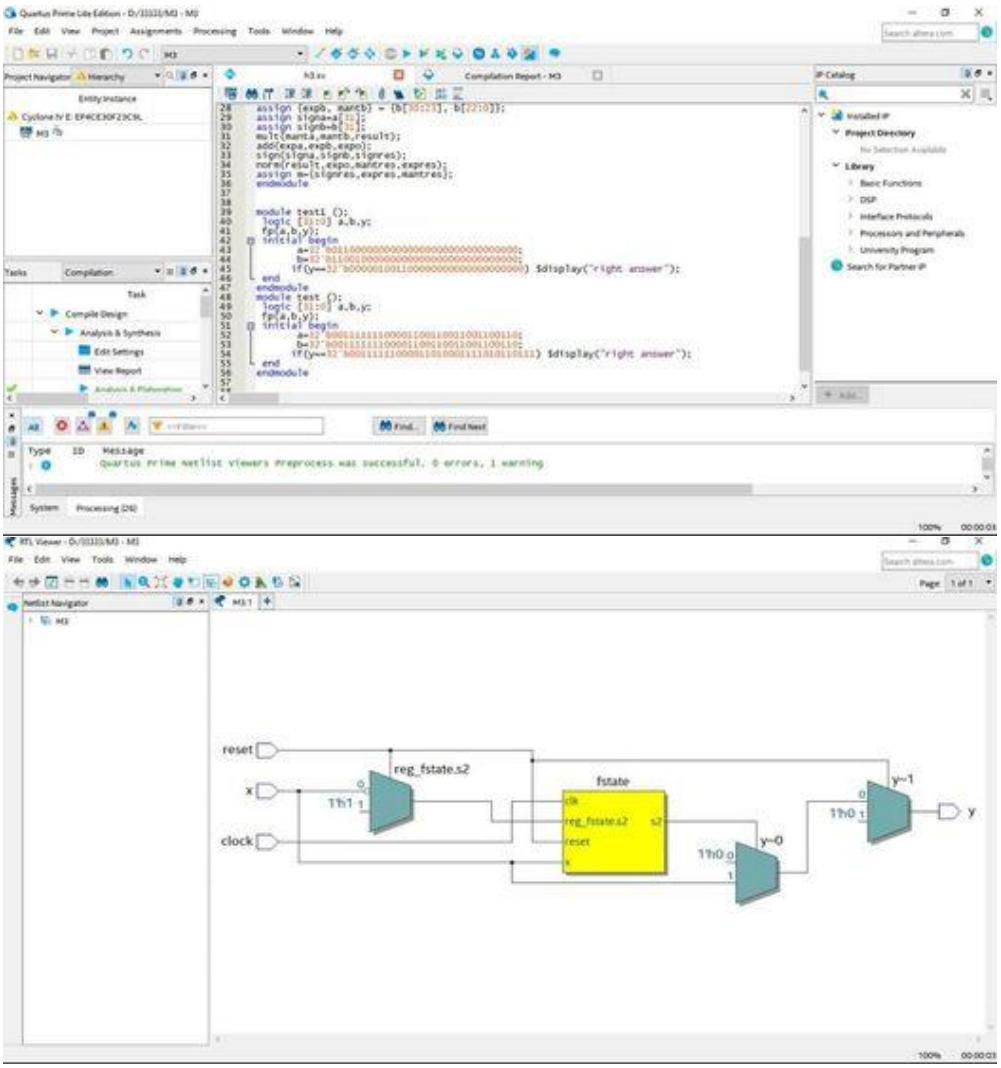
(j) Error: combinational module so statements (=) should be used in the always statement.

```
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 add #(parameter x=8)(input logic [x-1:0] exp1,exp2,output logic[x-1:0] y);
assign y=exp1+exp2-127;
endmodule

module  mult #(parameter x=23)(input logic [x-1:0] mant1,mant2,output logic[47:0] y);
logic[x:0]m1,m2;
assign ma={1'b1,mant1};
assign mb={1'b1,mant1};
assign y = mant1 * mant2;
endmodule

module  sign #(parameter x=1)(input logic sign1,sign2,output logic m);
assign m=sign1 ^ sign2;
endmodule
```

```
module  norm #(parameter x=8)(input logic [47:0] result,input logic[x-1:0] expo,output logic[22:0]
y,output logic[x-1:0]expres);
assign y = result[47] ?result[46:24] :result[45:23];
assign expres=expo-1;
endmodule

module fp(input logic [31:0] a, b,output logic [31:0] m);
logic [7:0] expa, expb, expres,expo;
logic [23:0] manta, mantb,mantres;
logic signa,signb,signres;
logic [47:0] result;
assign {expa, manta} = {a[30:23], a[22:0]};
assign {expb, mantb} = {b[30:23], b[22:0]};
assign signa=a[31];
assign signb=b[31];
mult(manta,mantb,result);
add(expa,expb,expo);
sign(signa,signb,signres);
norm(result,expo,mantres,expres);
assign m={signres,expres,mantres};
endmodule


module test1 ();
 logic [31:0] a,b,y;
 fp(a,b,y);
 intial begin
      a=32'b01100000000000000000000000000000;
      b=32'b11001000000000000000000000000000;
     if(y==32'b00000100110000000000000000000000) $display("right answer");
 end
endmodule
module test ();
 logic [31:0] a,b,y;
 fp(a,b,y);
 intial begin
      a=32'b00111111100001100110011001100110;
      b=32'b00111111100001100110011001100110;
     if(y==32'b00111111000011010001111010110111) $display("right answer");
 end
endmodule
```

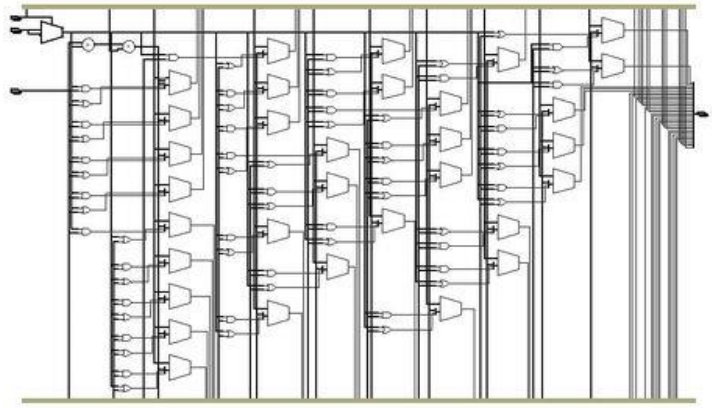


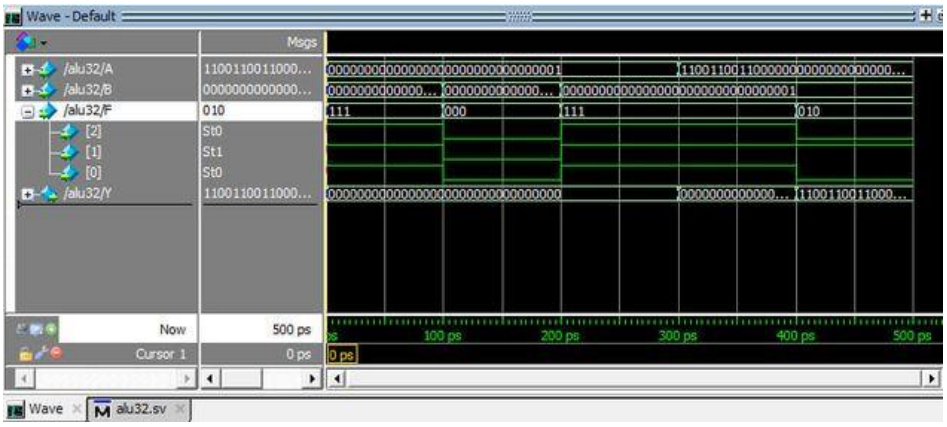
Task 4 – ALU

Exercise 5.9

```
module ALU32(input logic [31:0] A, B,
                        input logic [2:0] F,
                           output logic [31:0] Y);
    logic [31:0] S, BB;
     assign BB = F[2] ? ~B : B;
     assign S = A + BB + F[2];
  always_comb
```

```
        begin
         case (F[1:0])
              2'b00: Y <= A & BB;
              2'b01: Y <= A | BB;
              2'b10: Y <= S;
              2'b11: Y <= S[31];
         endcase
      end
endmodule
```



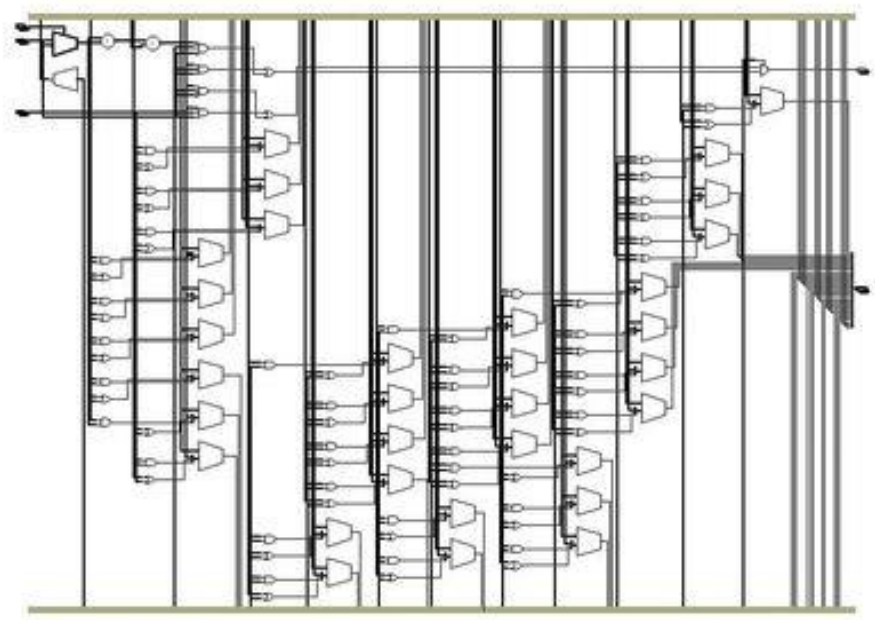


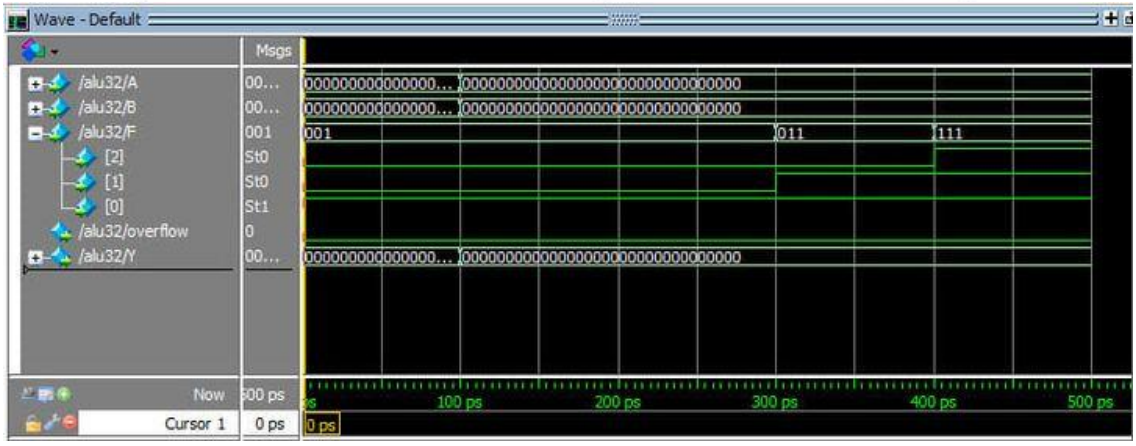
Exercise 5.10

(b)

(C) module ALUwithOverflow(input logic [31:0] A, B,
input logic [2:0] F,
output logic Overflow,

```
                              output logic [31:0] Y);
    logic [31:0] S, BB;
     assign BB = F[2] ? ~B : B;
     assign S = A + BB + F[2];
  always_comb
      begin
       case (F[1:0])
            2'b00: Y <= A & BB;
            2'b01: Y <= A | BB;
            2'b10: Y <= S;
            2'b11: Y <= S[31];
       endcase
      end
         always_comb
          case (F[2])
          1'b0: Overflow = A[31] & B[31] & ~S[31] |
                           ~A[31] & ~B[31] & S[31];
          1'b1: Overflow = ~A[31] & B[31] & S[31] |
                           A[31] & ~B[31] & ~S[31];
          default: Overflow = 1'b0;
          endcase
endmodule
```





Exercise 5.11

```
module ALUwithZero(input logic [31:0] A, B,
                   input logic [2:0] F,
                     output logic Overflow,
                     output logic zero,
                      output logic [31:0] Y);
    logic [31:0] S, BB;
     assign BB = F[2] ? ~B : B;
     assign S = A + BB + F[2];
  always_comb
      begin
       case (F[1:0])
            2'b00: Y <= A & BB;
```

```
            2'b01: Y <= A | BB;
            2'b10: Y <= S;
            2'b11: Y <= S[31];
        endcase
       end
     assign zero = (Y==0);
          always_comb
          case (F[2:1])
          2'b01: Overflow = A[31] & B[31] & ~S[31] |
                            ~A[31] & ~B[31] & S[31];
          2'b11: Overflow = ~A[31] & B[31] & S[31] |
                              A[31] & ~B[31] & ~S[31];
          default: Overflow = 1'b0;
          endcase
endmodule
```