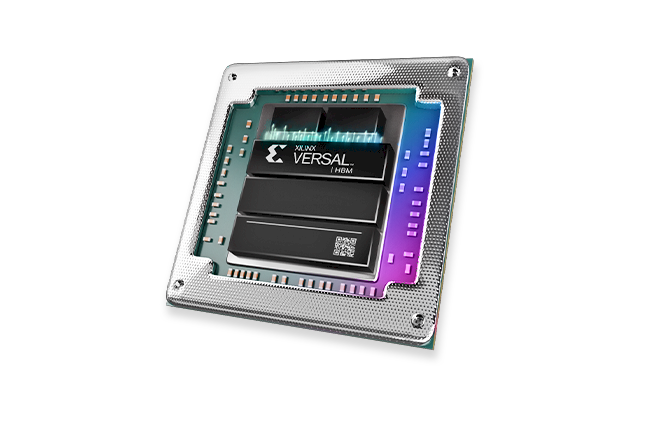
**EC204**

**Digital System Design Lab**

**Lab – 4**



**Utkarsh R Mahajan**

**201EC164**

1] 2 to 4 decoder using

(a) concurrent signal assignment statements

*module* twofourdecoderA (

input [1:0] X,

output [3:0] Y

);

assign Y[0] = ~X[0] & ~X[1];

assign Y[1] = ~X[1] & X[0];

assign Y[2] = X[1] & ~X[0];

assign Y[3] = X[1] & X[0];

*endmodule*

*module* testfunctionA();

    reg [1:0] X;

    wire [3:0]Y;

    integer a;

twofourdecoderA test(.X(X), .Y(Y));

initial

begin

    for(a=0;a<4;a=a+1)

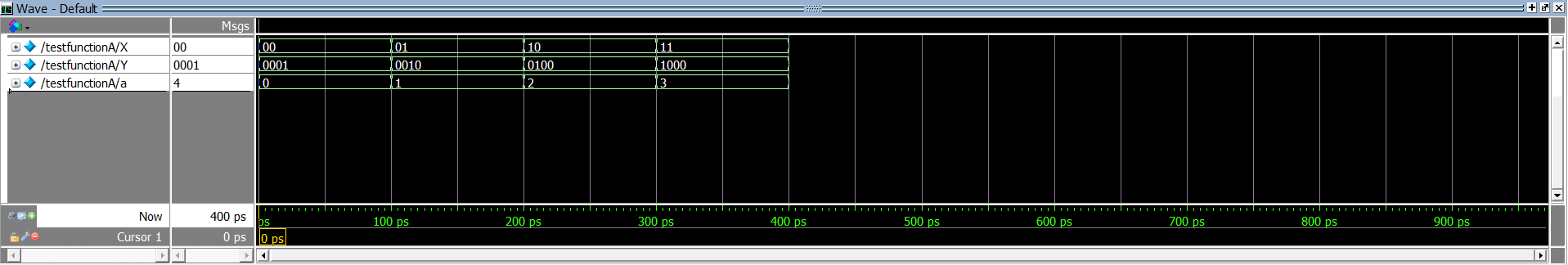
    begin

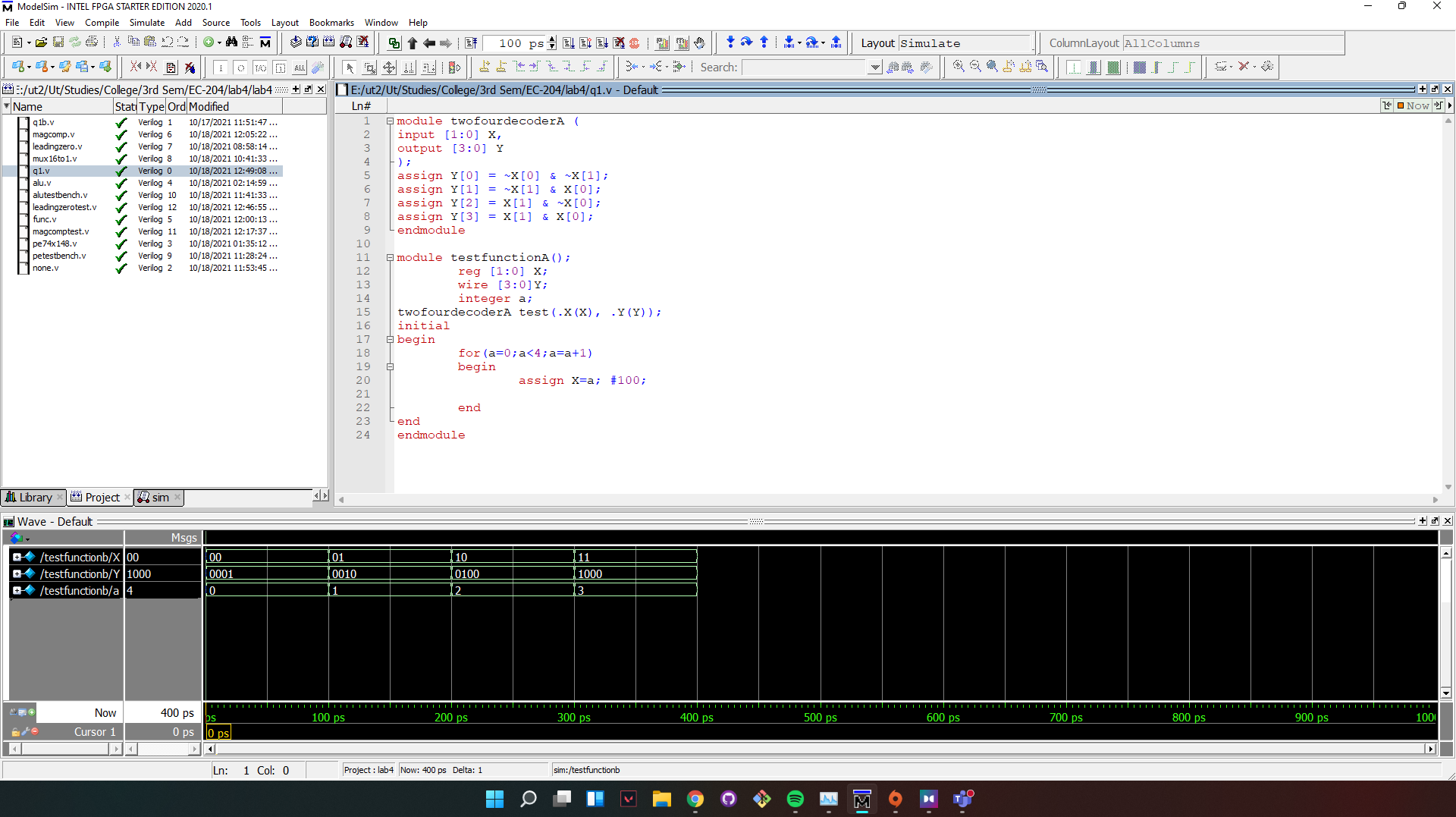
        assign X=a; #100;

    end

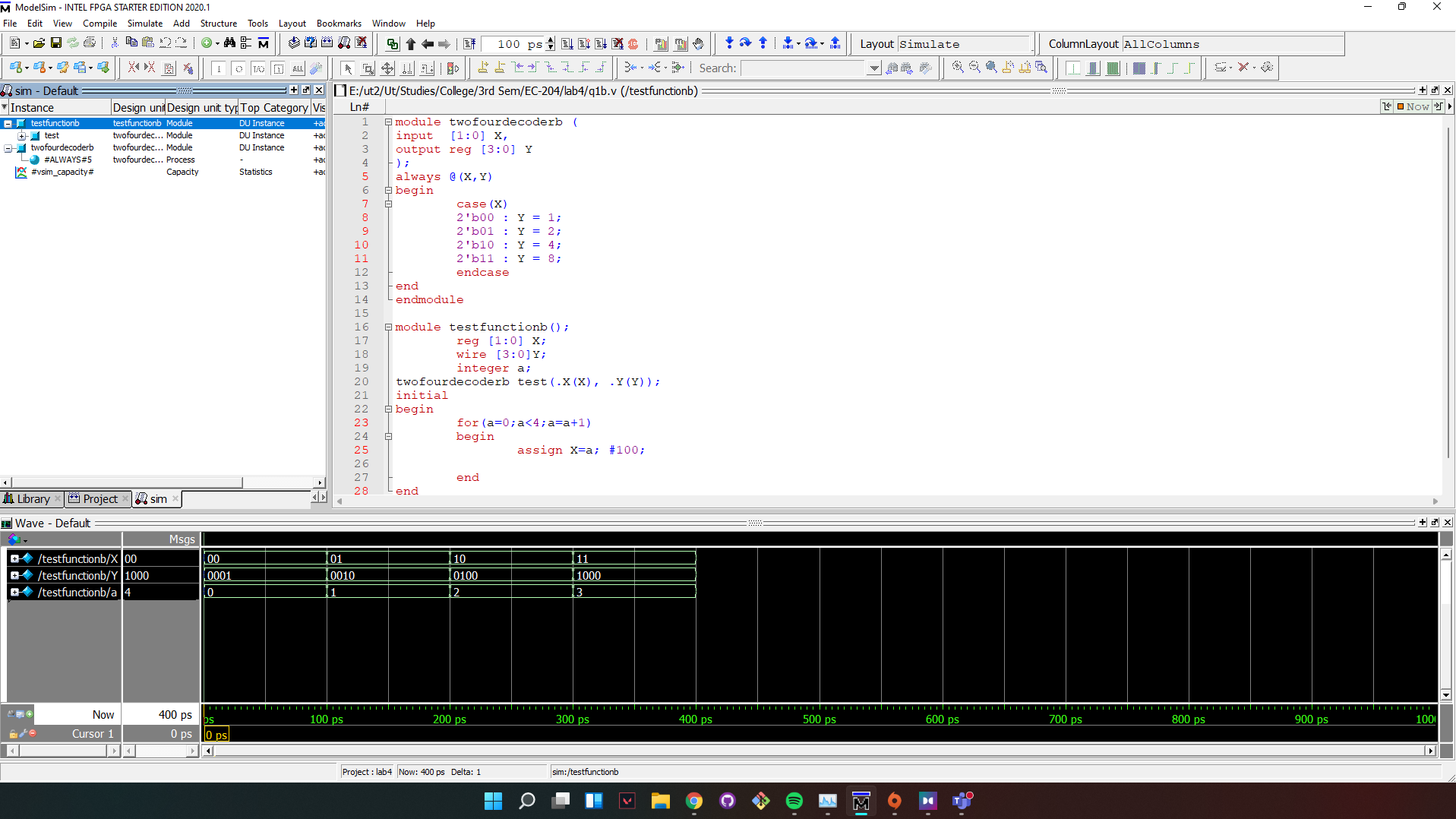
end

*endmodule*

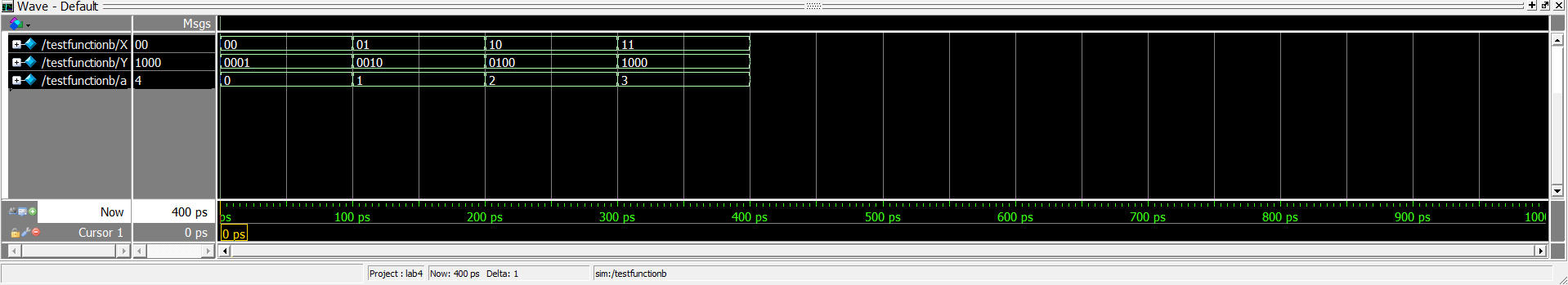
**Wave:**

****

**(b) using case statement**

****

**Wave:**

****

*module* twofourdecoderb (

input  [1:0] X,

output reg [3:0] Y

);

always @(X,Y)

begin

    case(X)

    2'b00 : Y = 1;

    2'b01 : Y = 2;

    2'b10 : Y = 4;

    2'b11 : Y = 8;

    endcase

end

*endmodule*

*module* testfunctionb();

    reg [1:0] X;

    wire [3:0]Y;

    integer a;

twofourdecoderb test(.X(X), .Y(Y));

initial

begin

    for(a=0;a<4;a=a+1)

    begin

        assign X=a; #100;

    end

end

*endmodule*

2] 16:1 multiplexer

*module* muxtestbench();

    reg [3:0] S;

    reg [15:0]I;

    wire O;

    integer a;

mux164to1 test(.S(S),.I(I), .O(O));

initial

begin

    I=16'b0100110000010010;

    for(a=1;a<16;a=a+2)

    begin

        assign S=a;

        #100;

    end

    I=16'b1010001000000111;

    for(a=1;a<16;a=a+2)

    begin

        assign S=a;

        #100;

    end

    I=16'b1011000010001110;

    for(a=0;a<16;a=a+2)

    begin

        assign S=a;

        #100;

    end

    I=16'b1011110011010001;

    for(a=1;a<16;a=a+2)

    begin

        assign S=a;

        #100;

    end

    I=16'b1001101100111010;

    for(a=0;a<16;a=a+2)

    begin

        assign S=a;

        #100;

    end

end

*endmodule*

*module* mux164to1 (

input [3:0]S, [15:0]I,

output reg O

);

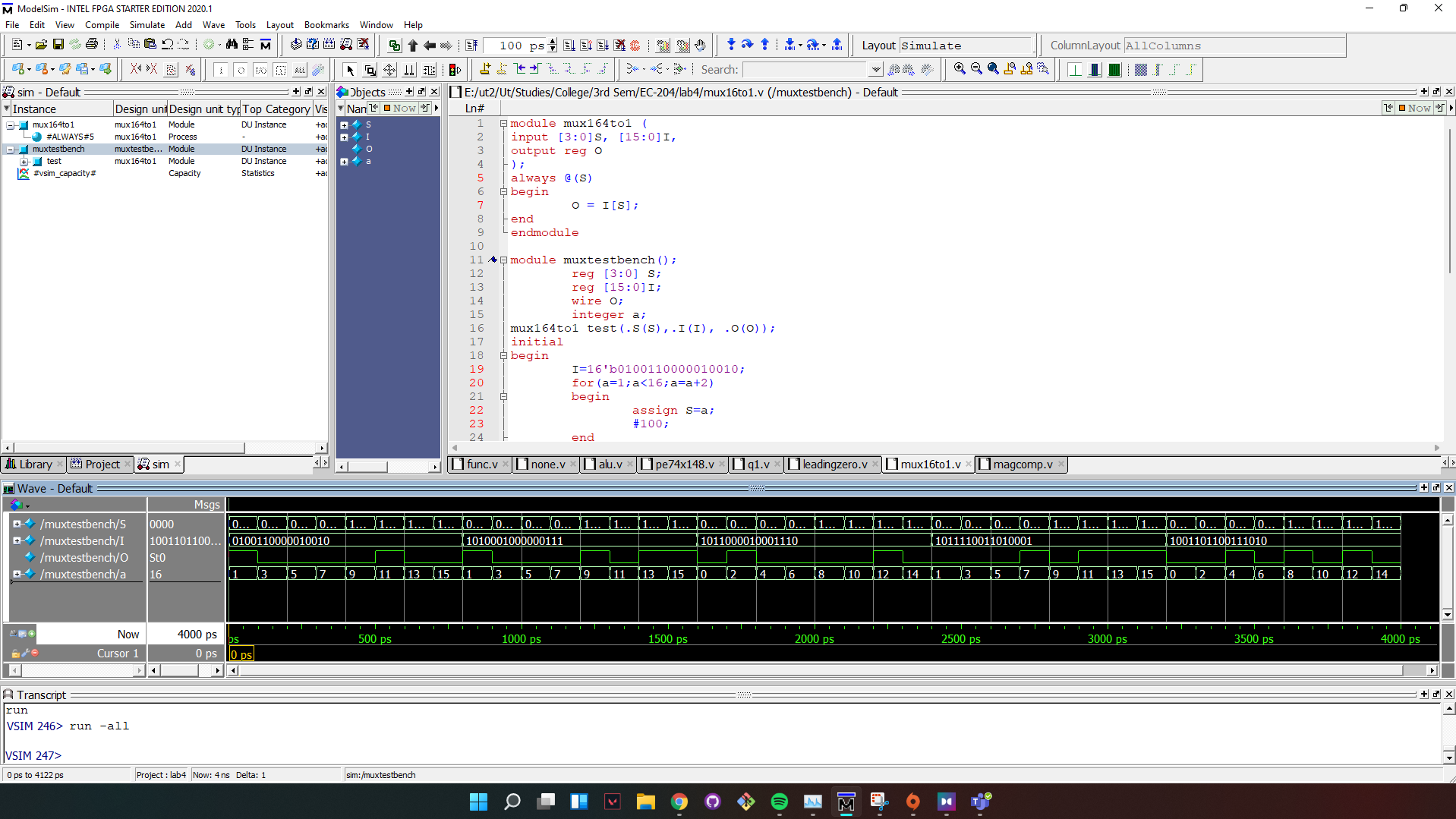
always @(S)

begin

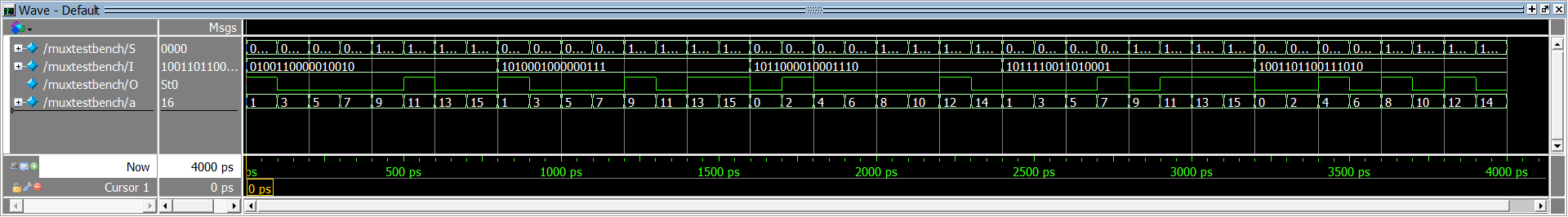
    O = I[S];

end

*endmodule*

****

**Wave:**

****

3] Functionality of 74x148 priority encoder

*module* priorityEncoder74x148 (

input  [7:0] IL,

input EIL,

output reg [2:0] AL,

output reg GSL,

output reg EOL

);

always @(EIL,IL)

begin

    if(EIL==0)

    begin

    casex(IL)

    8'b0???????: begin

    AL = 3'b000;

    GSL = 0;

    EOL = 1;

    end

    8'b10??????: begin

    AL = 3'b001;

    GSL = 0;

    EOL = 1;

    end

    8'b110?????: begin

    AL = 3'b010;

    GSL = 0;

    EOL = 1;

    end

    8'b1110????: begin

    AL = 3'b011;

    GSL = 0;

    EOL = 1;

    end

    8'b11110???: begin

    AL = 3'b100;

    GSL = 0;

    EOL = 1;

    end

    8'b111110??: begin

    AL = 3'b101;

    GSL = 0;

    EOL = 1;

    end

    8'b1111110?: begin

    AL = 3'b110;

    GSL = 0;

    EOL = 1;

    end

    8'b11111110: begin

    AL = 3'b111;

    GSL = 0;

    EOL = 1;

    end

    8'b11111111: begin

    AL = 3'b111;

    GSL = 1;

    EOL = 0;

    end

    endcase

    end else begin

    AL = 3'b111;

    GSL = 1;

    EOL = 1;

    end

end

*endmodule*

*module* petestbench();

reg [7:0] IL;

reg EIL;

wire [2:0] AL;

wire GSL;

wire EOL;

integer a;

priorityEncoder74x148 test(.EIL(EIL),.IL(IL), .AL(AL), .GSL(GSL), .EOL(EOL));

initial

begin

    IL = 8'b01000100;

    EIL =1;

    #100;

    EIL =0;

    IL =8'b00000000;

    for(a=0; a<256; a=a+3)  begin

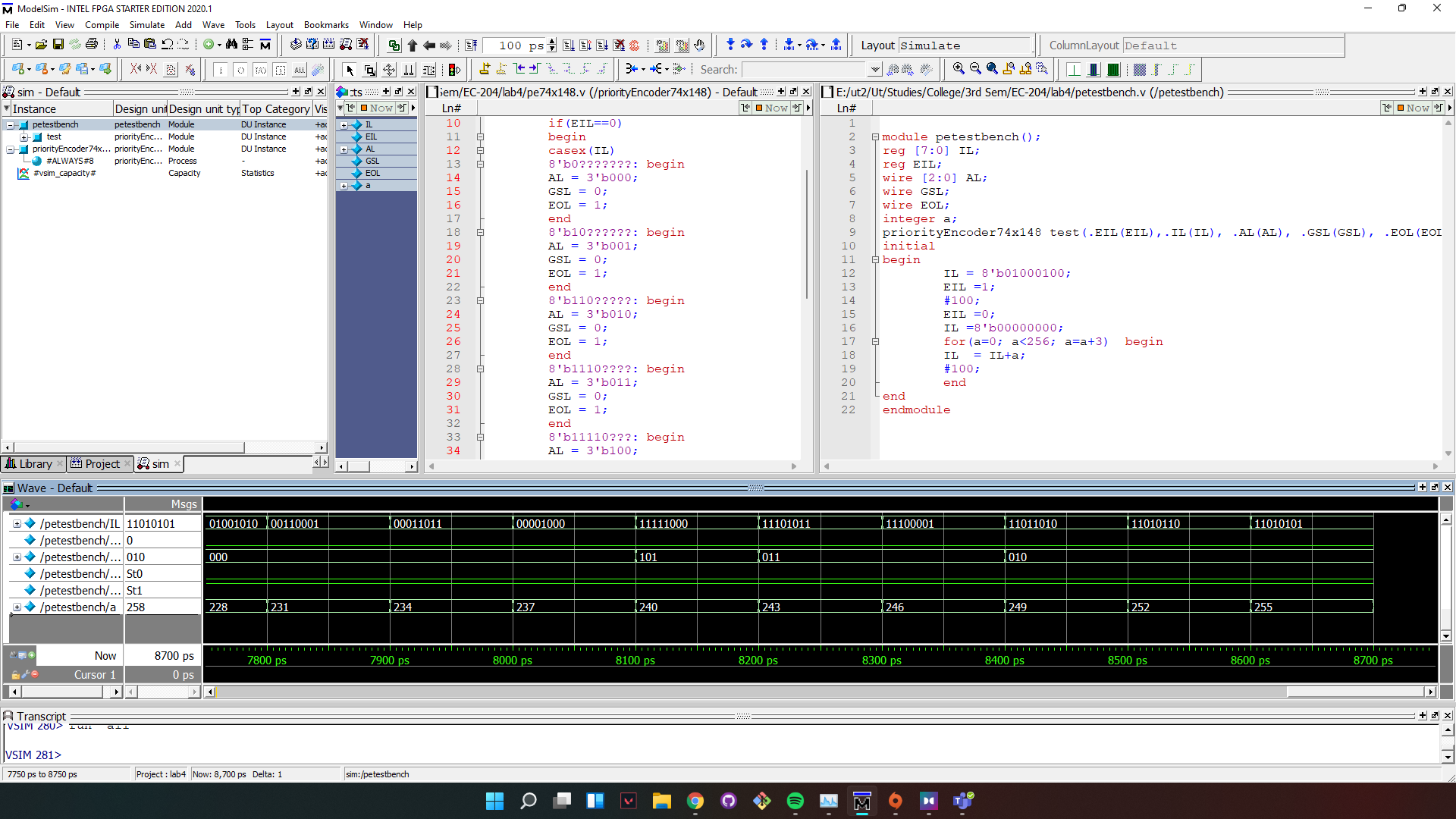
    IL  = IL+a;

    #100;

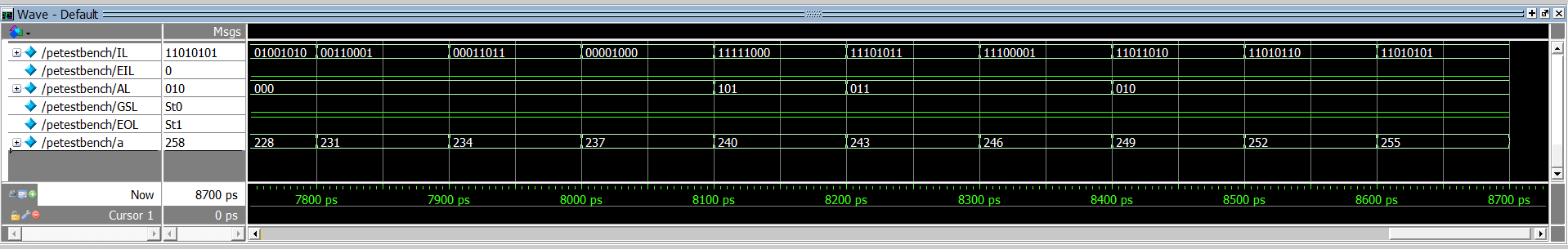
    end

end

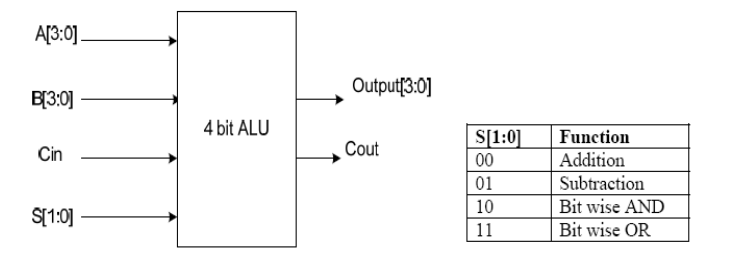
*endmodule*



**Wave:**



4] ALU design: Design a 4 bit ALU that is capable of performing addition, subtraction, bitwise AND and bit-wise OR instructions on 4 bit operands.

****

*module* alu (

input [1:0] S,

input [3:0] A, B,

input Cin,

output reg [3:0] Output,

output reg Cout

);

always @ (S, A, B)

begin

case (S)

    2'b00: begin

    {Cout, Output} = A + B;

    end

    2'b01: begin

    {Cout, Output} = A - B;

    end

    2'b10: begin

    Output = A & B;

    Cout =0;

    end

    2'b11: begin

    Output = A | B;

    Cout = 0;

    end

endcase

end

*endmodule*

*module* alutestbench ();

reg [1:0]S;

reg [3:0] A, B;

reg Cin;

wire [3:0] Output;

wire Cout;

integer i;

alu test(.S(S), .A(A), .B(B), .Cin(Cin), .Output(Output), .Cout(Cout));

initial

begin

    for(i=0;i<4;i=i+1)begin

    S= i;

    if(S[0]==0) Cin =1;

    else Cin =0;

    A= 4'b1110;

    B= 4'b0110;

    #100;

    A= 4'b0100;

    B= 4'b0111;

    #100;

    A= 4'b1010;

    B= 4'b1111;

    #100;

    A= 4'b1101;

    B= 4'b0000;

    #100;

    A= 4'b0011;

    B= 4'b1100;

    #100;

    A= 4'b1011;

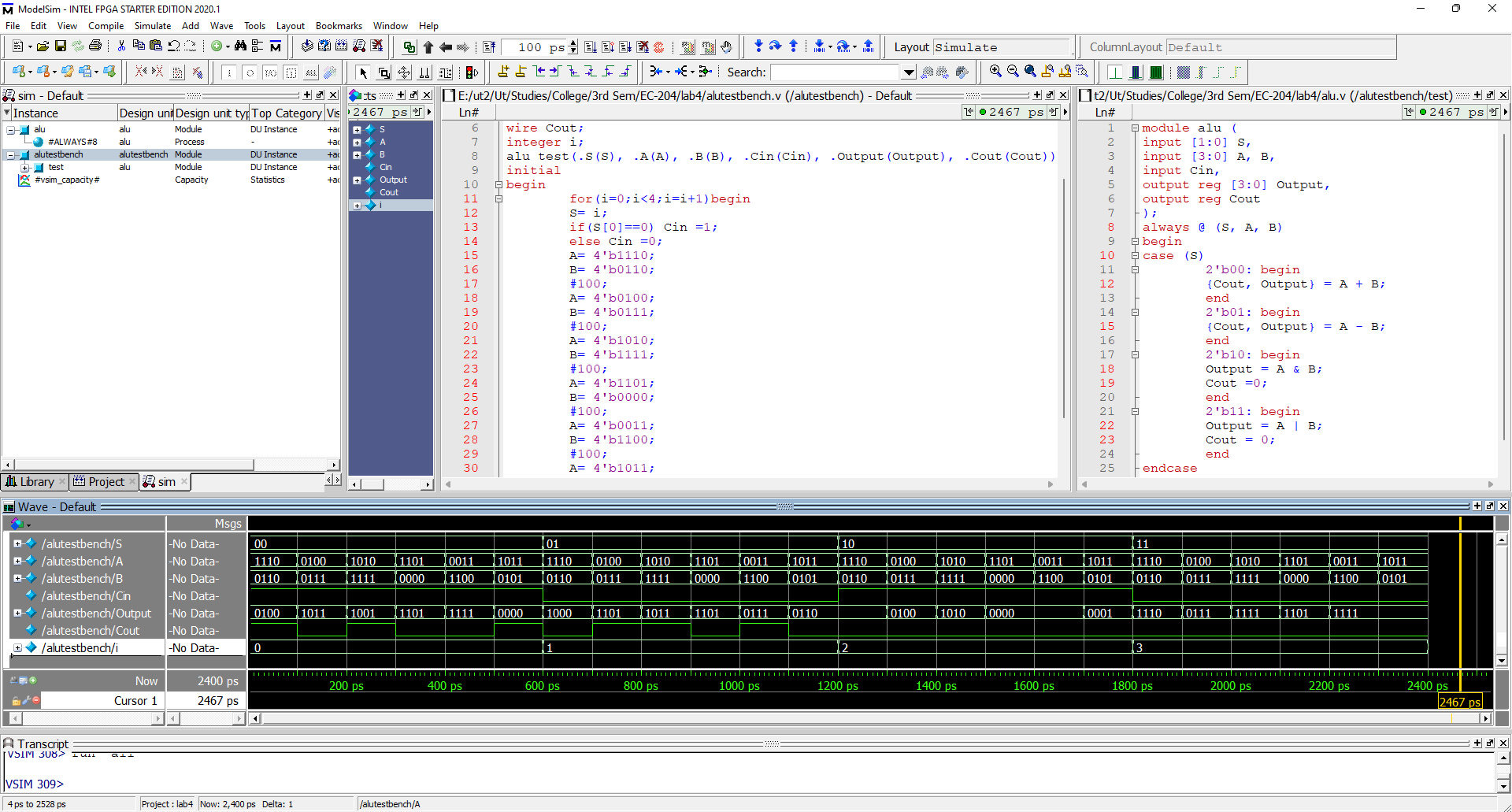
    B= 4'b0101;

    #100;

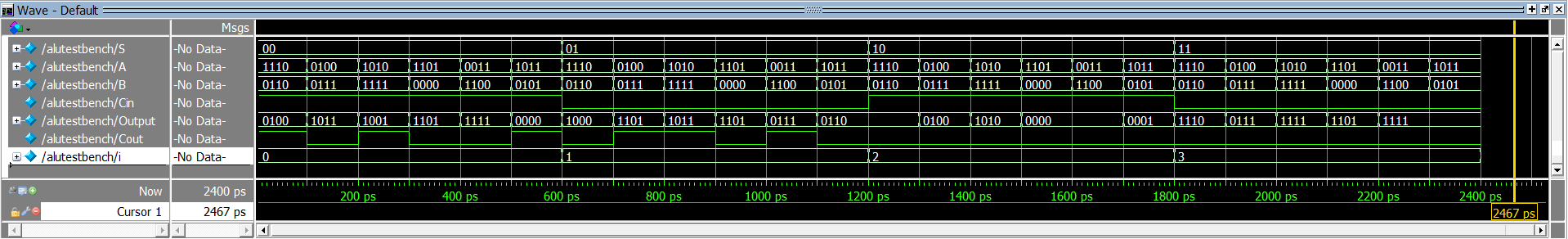
    end

end

*endmodule*

****

**Wave:**

****

5] Count the number of 1s in a 32 bit number

*module* NumberOfOnes(input [31:0] data, output reg [5:0] count);

    integer k;

    always @(data)

    begin

    count = 0;

    for (k=0; k < 31; k=k+1)

        count = count + data[k];

    end

*endmodule*

*module* NumberOfOnestest();

    reg [31:0] a;

    wire [5:0]count;

    integer i;

NumberOfOnes f(.data(a), .count(count));

initial

begin

    for(i=0;i<429496729;i=i+42900000)

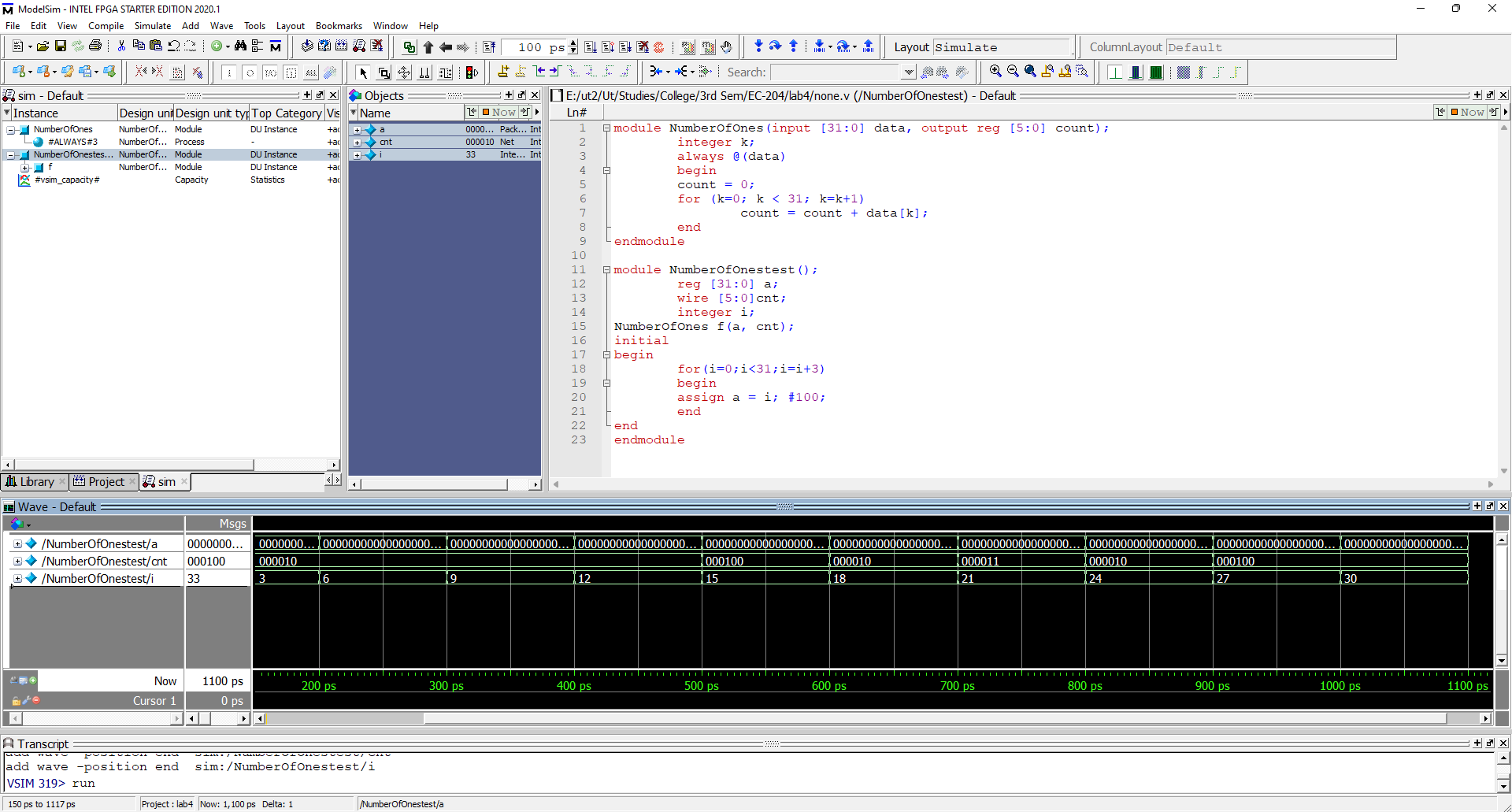
    begin

    assign a = i; #100;

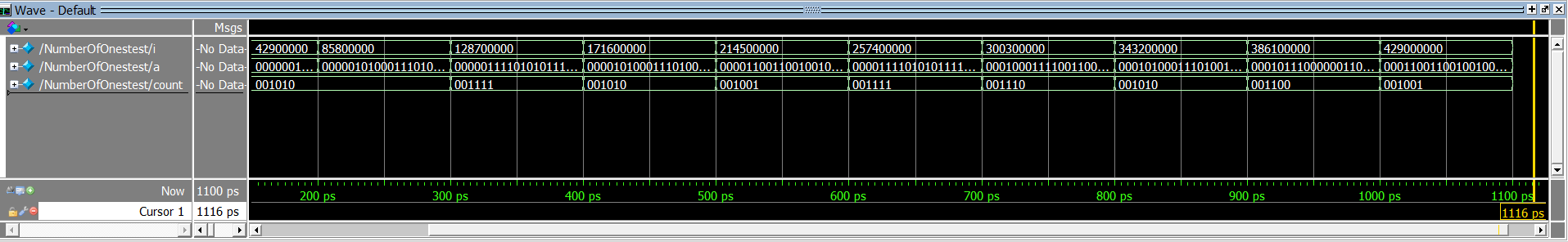
    end

end

*endmodule*

****

**Wave:**

****

6] Implement F(v,w,x,y,z) = ∑ (0, 2, 3, 4, 8, 21, 22, 29, 31)

*module* func(

input v, w, x, y, z,

output f

);

assign f = (~v & ~w & ~y & ~z) | (~v & ~x & ~y & ~z) | (~v & ~w & ~x & y)

    | (v & x & ~y & z) | (v & ~w & x & y & ~z ) | (v & w & x & z);

*endmodule*

*module* functest();

reg [4:0] a;

wire f;

integer i;

func test(.v(a[0]),.w(a[1]),.x(a[2]),.y(a[3]),.z(a[4]), .f(f));

initial

begin

    for(i=0;i<32;i=i+1)begin

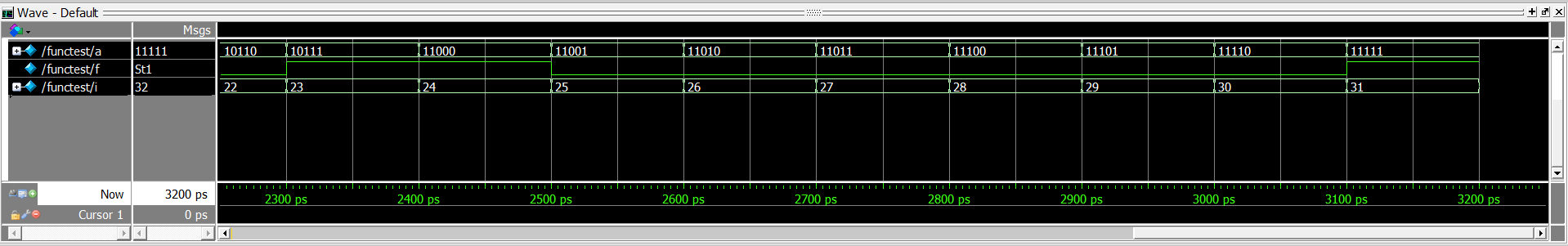
    a = i;

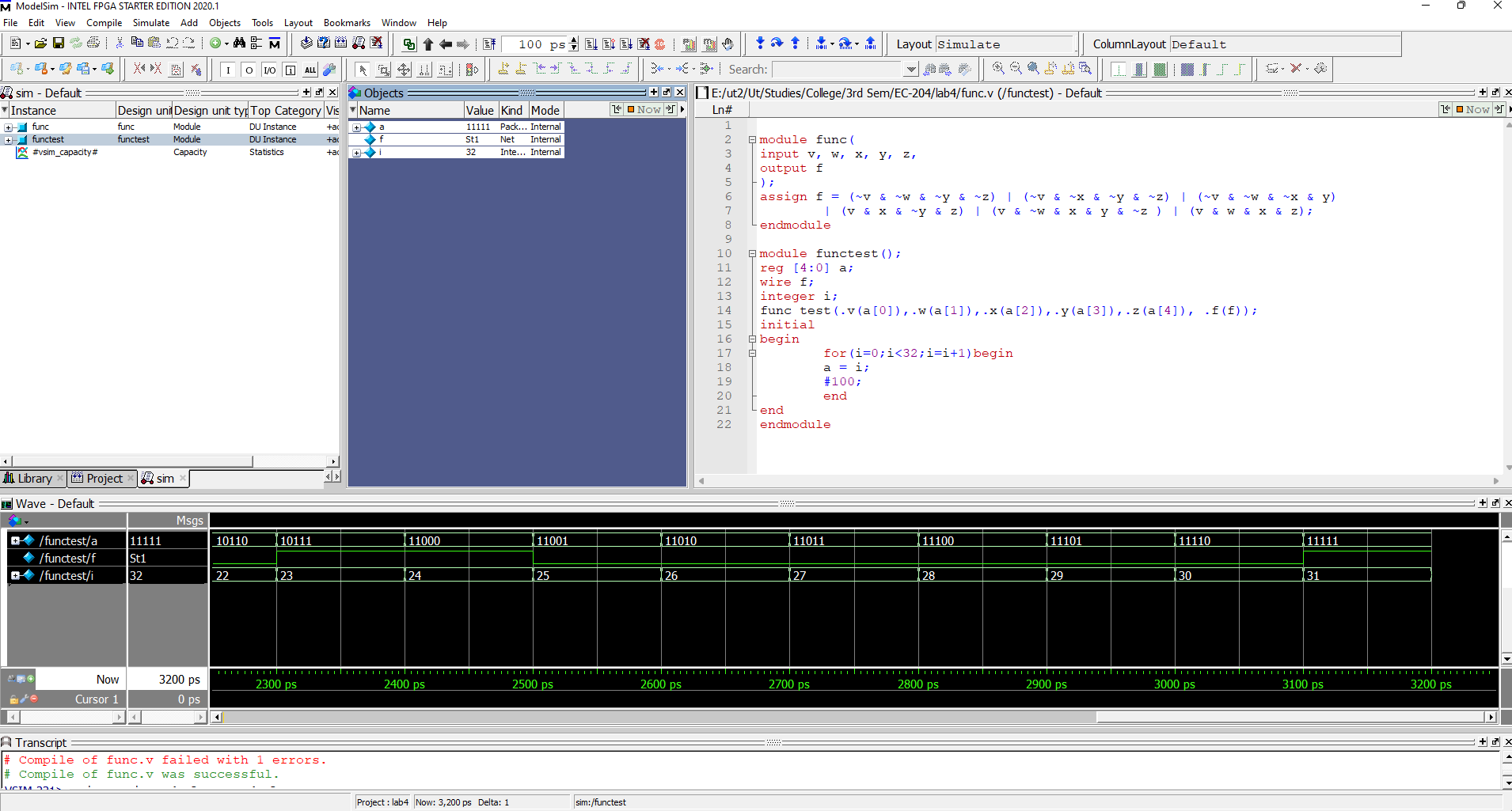
    #100;

    end

end

*endmodule*

****

****

7] 16 bit magnitude comparator

*module* magcomp (input [15:0] A, B,

output reg P, Z, N);

always @(A,B)

begin

    if(A > B) begin

    P=1; Z=0; N=0;

    end else if(A==B) begin

    P=0; Z=1; N=0;

    end else begin

    P=0; Z=0; N=1;

    end

end

*endmodule*

*module* magcomptest();

reg [15:0] A, B;

wire P, Z, N;

magcomp test(.A(A),.B(B),.P(P),.Z(Z),.N(N));

initial

begin

    A=16'b1000110011001010;

    B=16'b0101010011011100;

    #100;

    A=16'b0010111001100011;

    B=16'b0101110110001110;

    #100;

    A=16'b1110011001110110;

    B=16'b1000100000111001;

    #100;

    A=16'b0110100111011110;

    B=16'b0110100111011110;

    #100;

    A=16'b0110000000100011;

    B=16'b0000010110001000;

    #100;

    A=16'b1101011111101001;

    B=16'b0001110011000011;

    #100;

    A=16'b1010011011010001;

    B=16'b0110100001100110;

    #100;

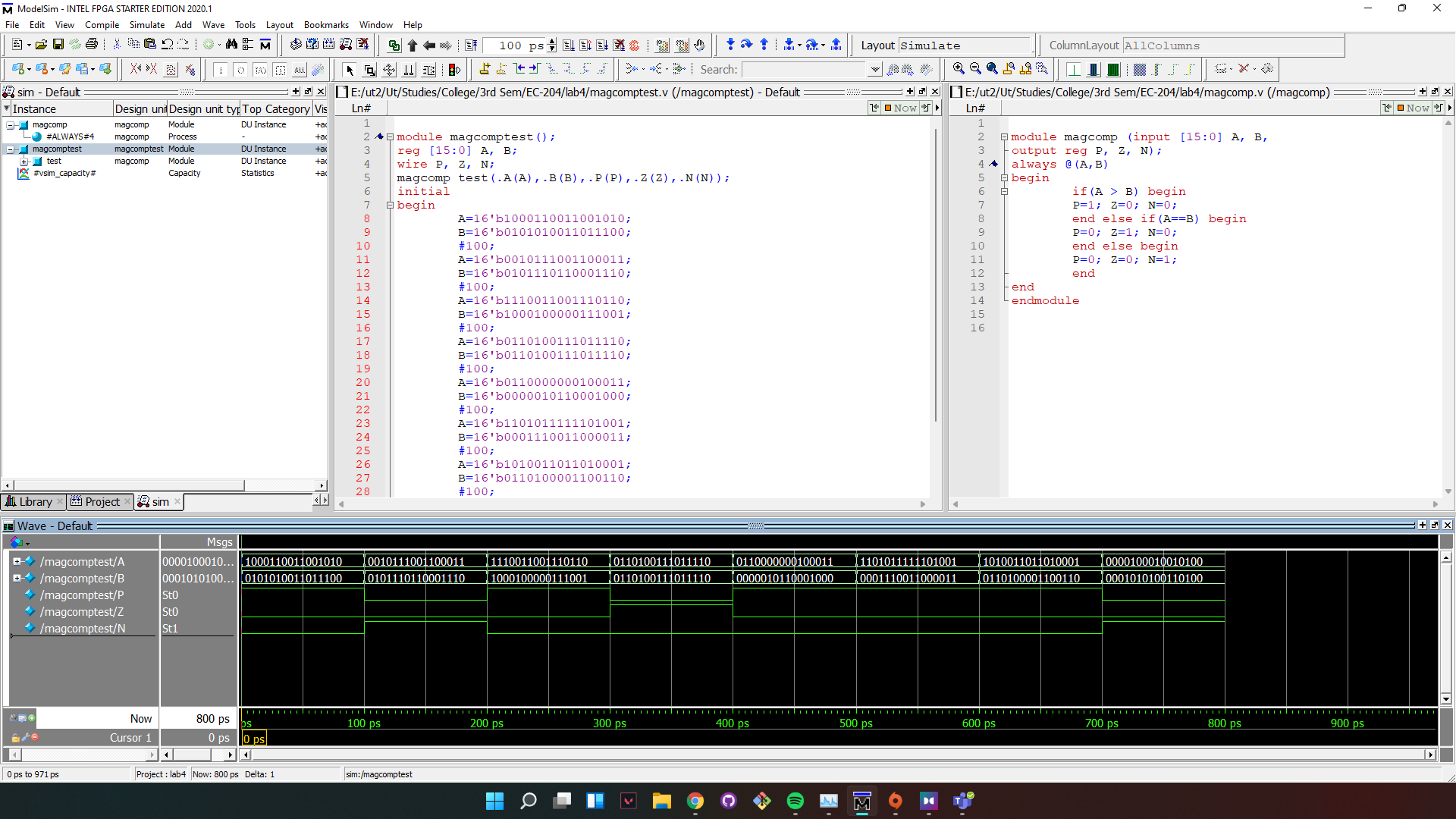
    A=16'b0000100010010100;

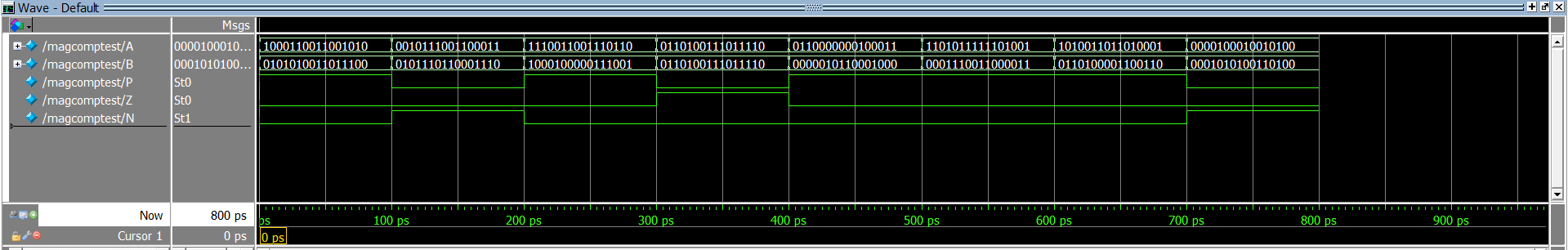
    B=16'b0001010100110100;

    #100;

end

*endmodule*





8] Count the number of leading 0s in an 8 bit number

*module* leadingzero (

input [7:0] A,

output reg [3:0]B);

integer k;

always @(A)

begin

    B=0; k=7;

    while(A[k]==0)

    begin

        B=B+1;

        k= k-1;

    end

end

*endmodule*

*module* leadingzerotest ();

reg [7:0]A;

wire [3:0]B;

integer i;

leadingzero test(.A(A), .B(B));

initial

begin

    for(i=0; i<256;i=i+17) begin

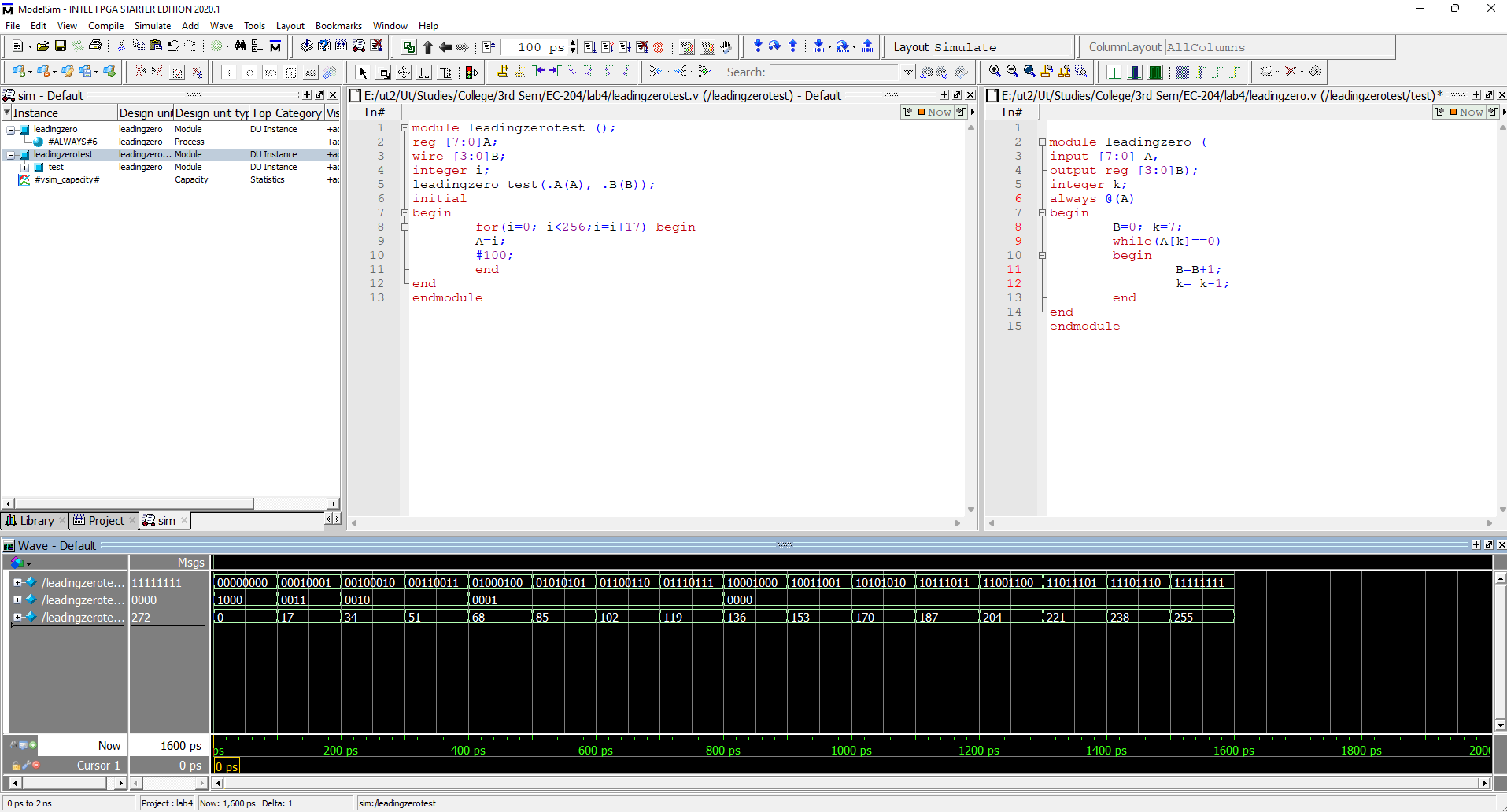
    A=i;

    #100;

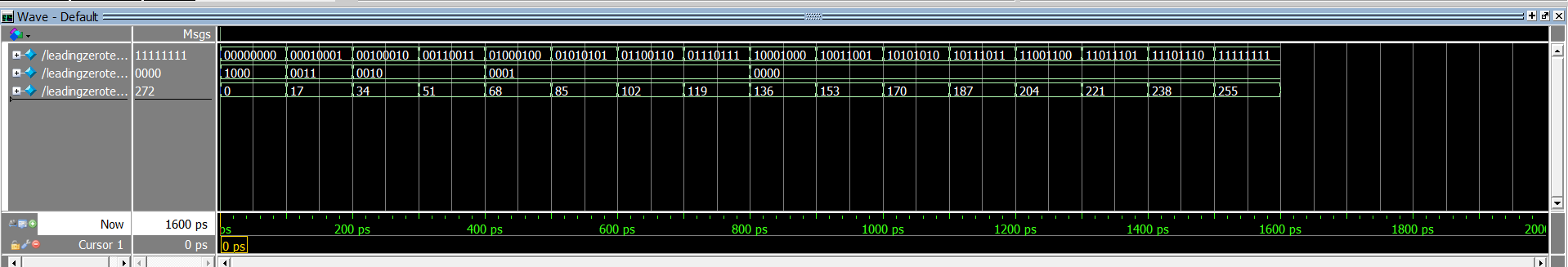
    end

end

*endmodule*

****

**Wave:**

****