EC200

Digital System Design

Assignment

Utkarsh R Mahajan

201EC164

1] Implement a shift and add multiplier in Verilog to multiply two 4 bit numbers. It should be capable of multiplying both positive and negative numbers. Negative numbers are represented in 2'sC representation.

->

Storing the input values in 4 bit registers A and B we will display the output using a 8bit Register P. we know that for signed multiplication when multiplicand is negative we need to ignore carry and use 1 as the msb's while adding to the register P when the multiplier bit is 1 while when multiplier is negative we just need to add two's complement in the last step.

Verilog code:

```
module signedmultiplier(input clk, resetn, start, input [3:0]A, B, output reg [7:0] P, output
reg done);
localparam S0=0, S1=1, S2=2, S3=3, S4=4, S5=5;
reg [2:0] state_d, state_q;
reg [3:0] B_d, B_q;
reg [7:0] A_d, A_q, pdt_d, pdt_q;
assign P = pdt_q;
always @ (posedge clk , negedge resetn)
begin
    if(!resetn) state_q <=S0;
    else begin
    state_q <= state_d;
    pdt_q <=pdt_d;
    A_q <= A_d;
    B_q <= B_d;
    end
end

always @(*) begin
    state_d = state_q;
    done = 1'b0;
    case(state_q)
        S0:if(start) state_d=S1;
        S1: state_d=S2;
        S2: state_d=S3;
        S3: state_d=S4;
        S4: state_d=S5;
        S5: begin
        done= 1'b1;
        if(start)state_d =S1;
        end
        default: state_d=S0;
    endcase
end

always @(*)
```

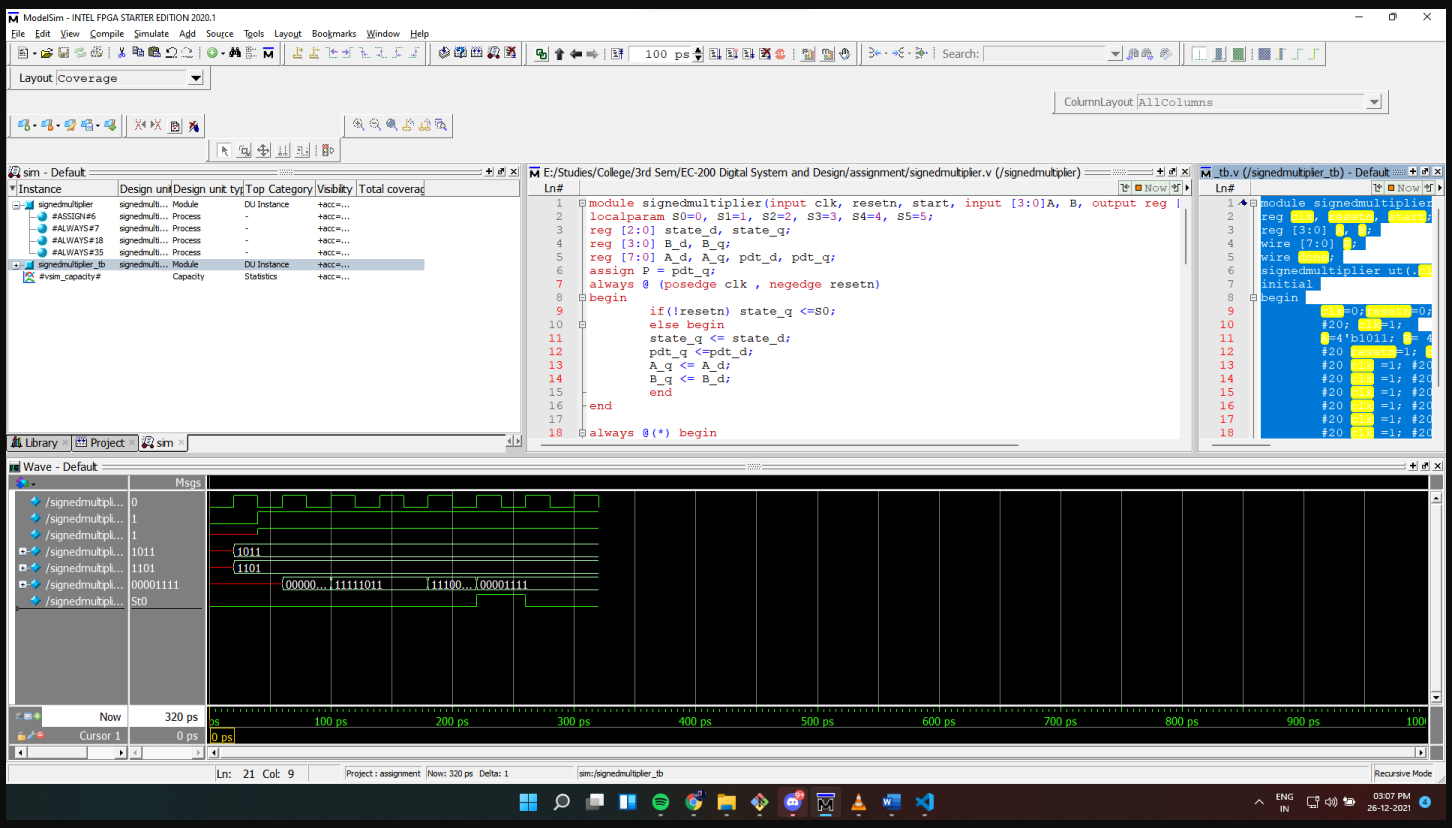
```
begin
pdt_d = pdt_q; A_d = A_q; B_d = B_q;
case (state_q)
S0: begin
pdt_d = {8{1'b0}};
A_d = A;
B_d = B;
end
S1: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[3] == 1'b1) pdt_d = {4'b1111,A_q[3:0]} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S2: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[3] == 1'b1) pdt_d = {3'b111,A_q[4:1], 1'b0} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S3: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[3] == 1'b1) pdt_d = {2'b11,A_q[5:2], 2'b00} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S4: begin
    A_d = A_q << 1;
    B_d = B_q >> 1;
    if (B_q[0] == 1'b1) begin
        if(B[3] == 1'b1) pdt_d = {1'b0, (~A+1'b1),3'b000} +pdt_q;
        else if(A[3] == 1'b1) pdt_d = {1'b1,A_q[6:3], 3'b000} +pdt_q;
        else pdt_d = A_q + pdt_q;
    end
end
default: pdt_d = pdt_q;
endcase
end
endmodule
```

Verilog Code for Testbench:

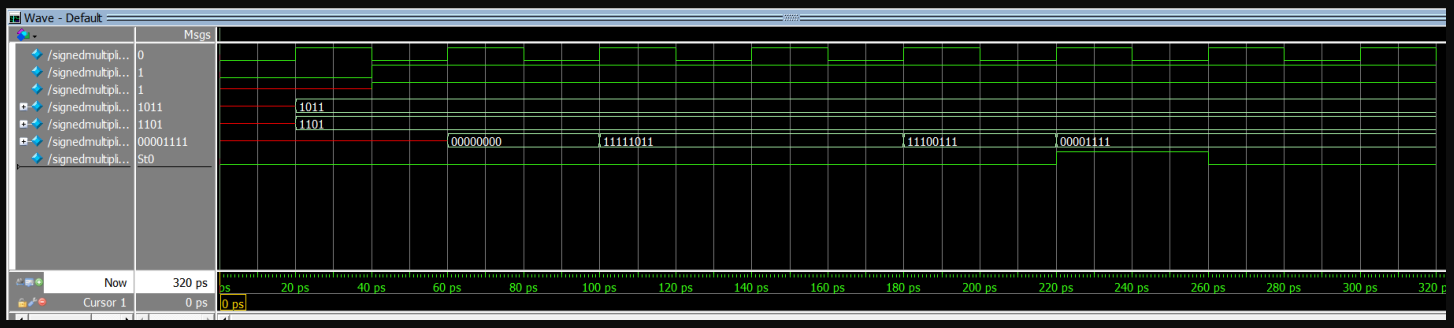
```
module signedmultiplier_tb();
reg clk, resetn, start;
```

```
reg [3:0] A, B;
wire [7:0] P;
wire done;
signedmultiplier ut(.clk(clk), .resetn(resetn), .start(start), .A(A), .B(B), .P(P),
.done(done));
initial
begin
    clk=0;resetn=0;
    #20; clk=1;
    A=4'b1011; B= 4'b1101;
    #20 resetn=1; clk=0; start=1'b1;
    #20 clk =1; #20 clk=0;
    #20 clk =1; #20 clk=0;
    #20 clk =1; #20 clk=0;
    #20 clk =1; #20 clk=0;
    #20 clk =1; #20 clk=0;
    #20 clk =1; #20 clk=0;
    #20 clk =1; #20 clk=0;
end
endmodule
```

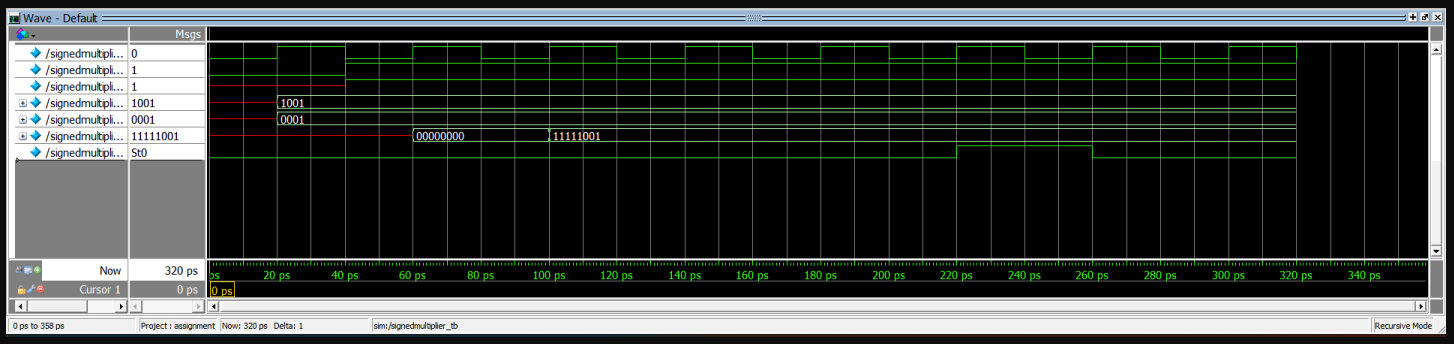
Output:



Waveform:



Different output:



2] Implement the division algorithm discussed in class in Verilog.

->

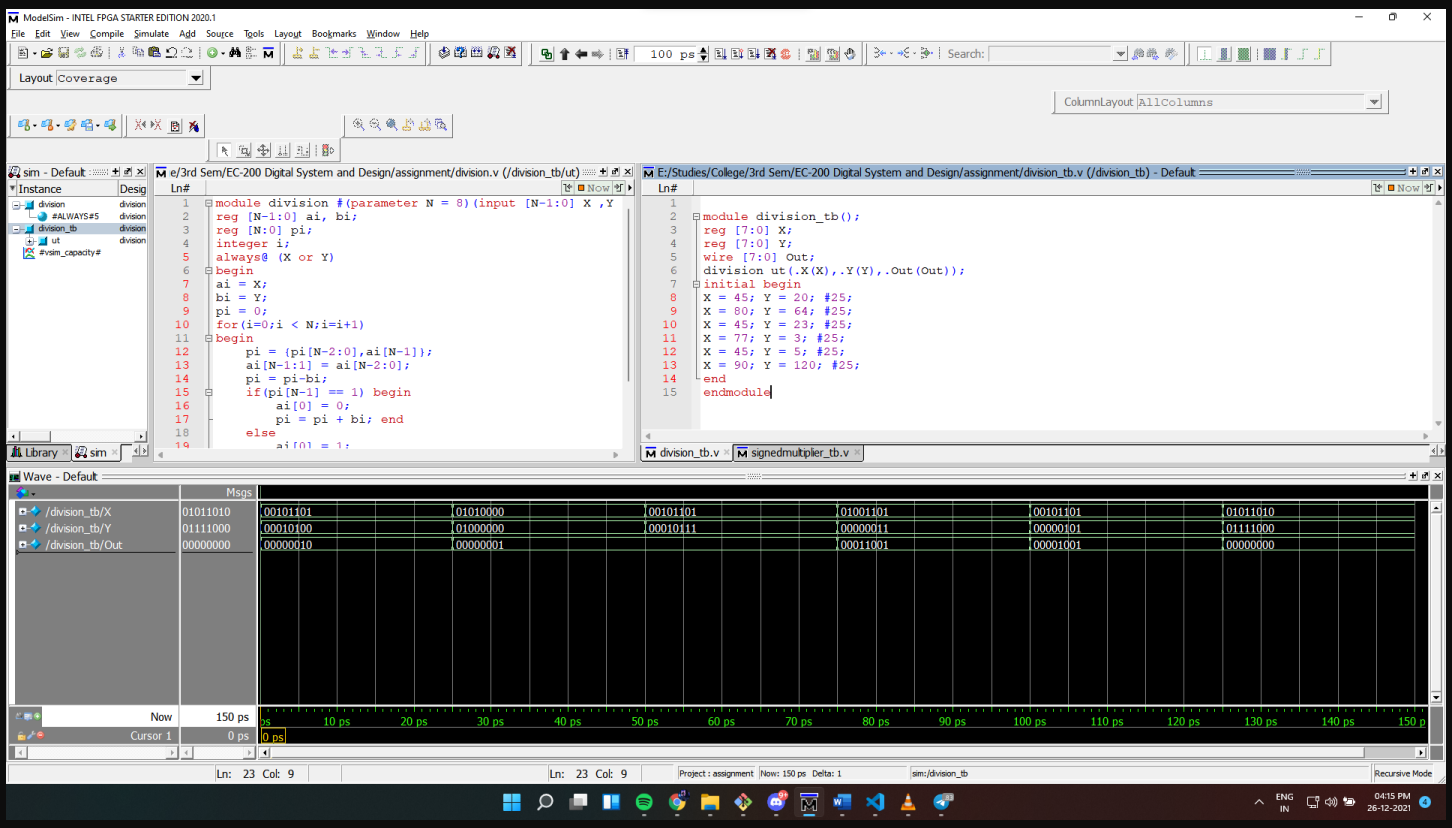
Verilog Code:

```
module division #(parameter N = 8)(input [N-1:0] X ,Y ,output reg [N-1:0] Out);
reg [N-1:0] ai, bi;
reg [N:0] pi;
integer i;
always@ (X or Y)
begin
ai = X;
bi = Y;
pi = 0;
for(i=0;i < N;i=i+1)
begin
    pi = {pi[N-2:0],ai[N-1]};
    ai[N-1:1] = ai[N-2:0];
    pi = pi-bi;
    if(pi[N-1] == 1) begin
        ai[0] = 0;
        pi = pi + bi; end
    else
        ai[0] = 1;
    end
    Out = ai;
end
endmodule
```

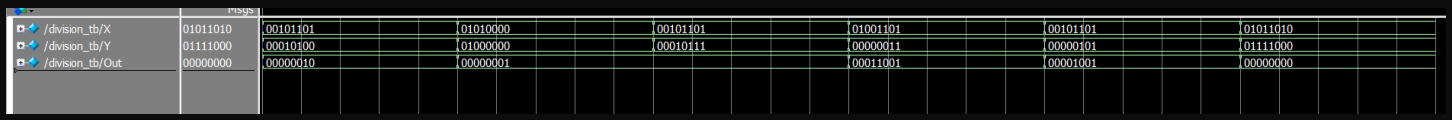
Verilog Code for the testbench:

```
module division_tb();
reg [7:0] X;
reg [7:0] Y;
wire [7:0] Out;
division ut(.X(X),.Y(Y),.Out(Out));
initial begin
X = 45; Y = 20; #25;
X = 80; Y = 64; #25;
X = 45; Y = 23; #25;
X = 77; Y = 3; #25;
X = 45; Y = 5; #25;
X = 90; Y = 120; #25;
end
endmodule
```

Output:



Waveform:



3] Write a VERILOG code to compute the dot product of two sequences

->

We will use shift registers for inputs and outputs and then use done signal to show that the output has been processed.

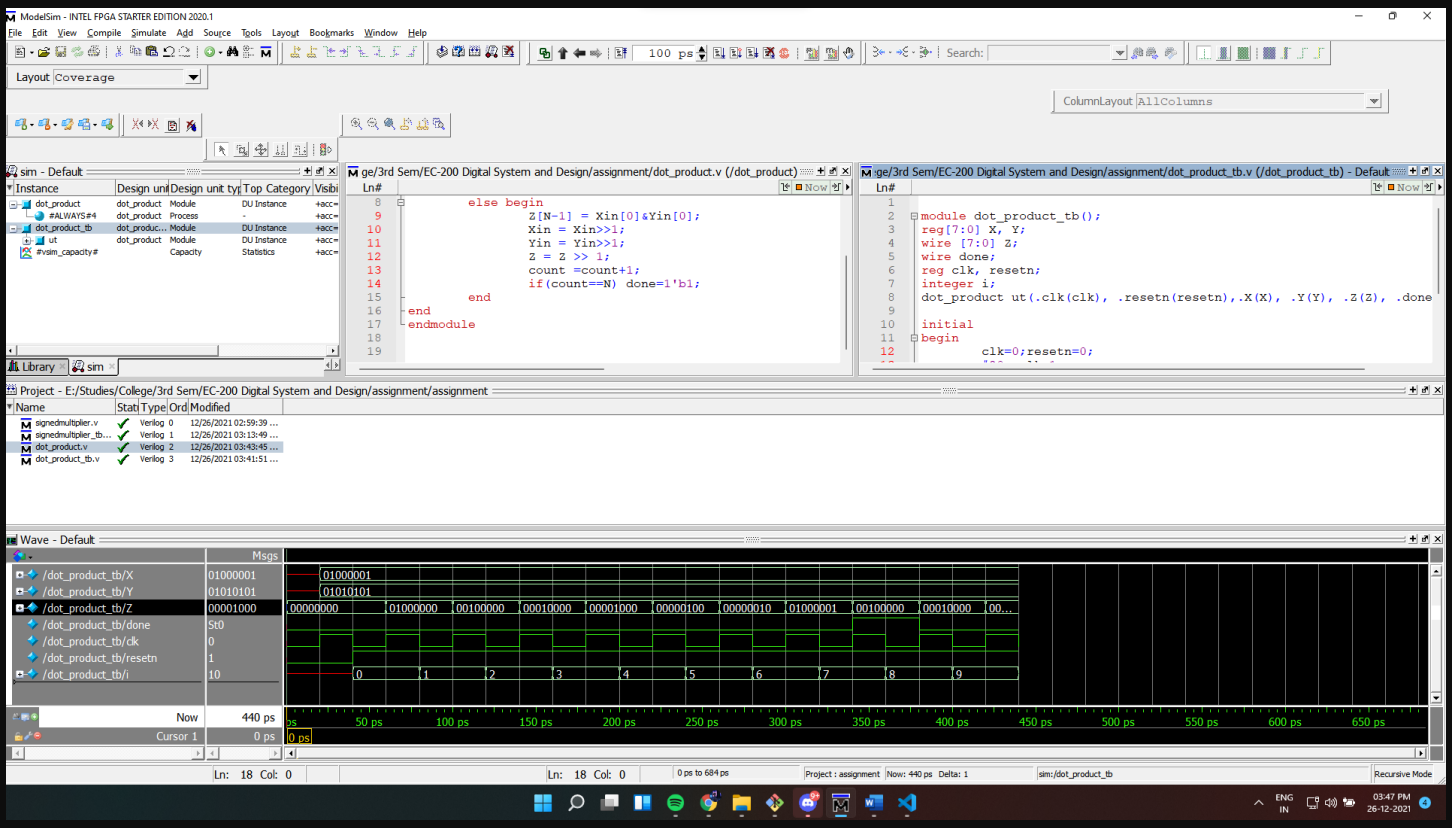
Verilog Code:

```
module dot_product #(parameter N=8)(input clk, resetn, input[N-1:0] X, Y, output reg [N-1:0]
Z, output reg done);
reg [N-1:0] Xin, Yin;
integer count;
always @(posedge clk, negedge resetn)
begin
    done =1'b0;
    if(!resetn) begin  Z=0;  Xin=X; Yin=Y; count=0; end
    else begin
        Z[N-1] = Xin[0]&Yin[0];
        Xin = Xin>>1;
        Yin = Yin>>1;
        Z = Z >> 1;
        count =count+1;
        if(count==N) done=1'b1;
    end
end
endmodule
```

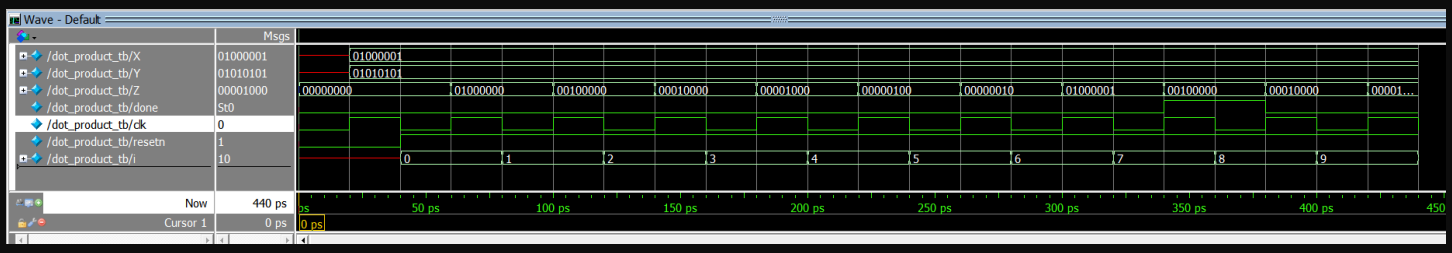
Verilog code for testbench:

```

module dot_product_tb();
reg[7:0] X, Y;
wire [7:0] Z;
wire done;
reg clk, resetn;
integer i;
dot_product ut(.clk(clk), .resetn(resetn),.X(X), .Y(Y), .Z(Z), .done(done));

initial
begin
    clk=0;resetn=0;
    #20; clk=1;
    X = 8'b01000001;
    Y = 8'b01010101;
    #20 resetn=1; clk=0;
    for(i=0; i<10; i=i+1) begin
        #20 clk =1; #20 clk=0;
        end
end
endmodule
```

Output:



Waveform:



4] Write a VERILOG code to compute the factorial of a number. You may need to make certain assumptions during the design of this system. State the assumptions used.

->

The assumptions are that both the inputs are unsigned and no overflow occurs in the output.

Verilog Code:

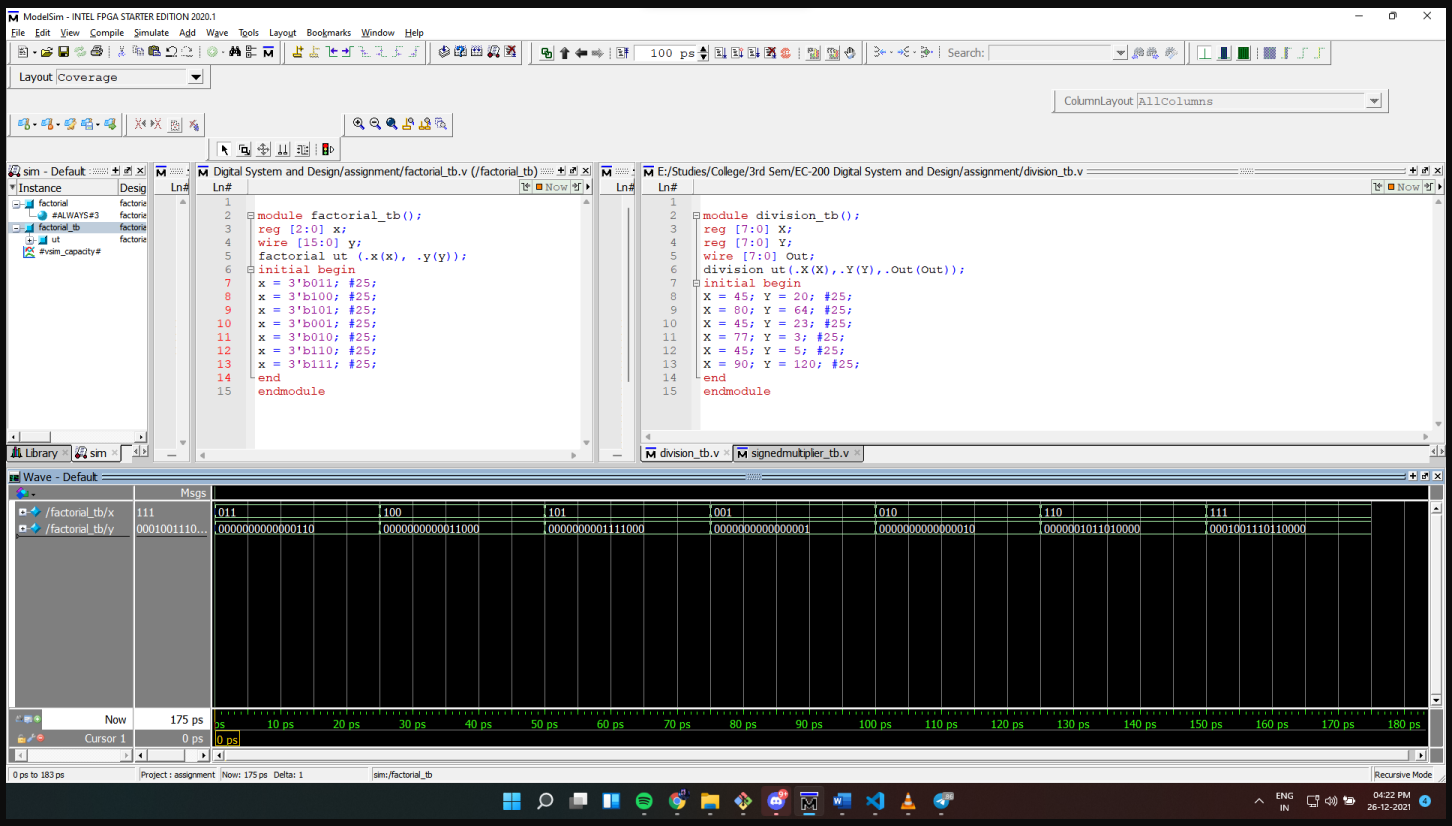
```
module factorial (input [2:0]x, output reg [15:0] y);
integer i;
always @ (x)
begin
```

```
    y = 1;
    i = 0;
    for(i=1;i<=x;i = i + 1) begin
    y = i * y;
    end
end
endmodule
```

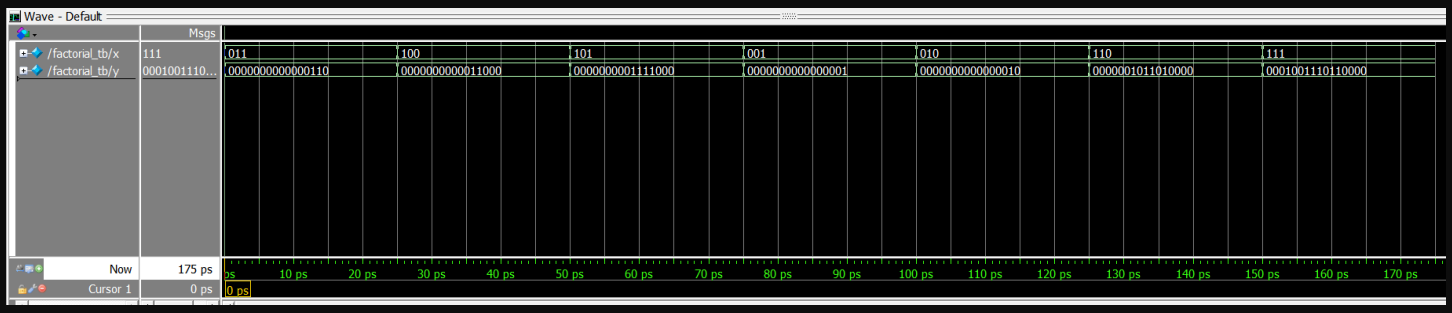
Verilog Code for testbench:

```
module factorial_tb();
reg [2:0] x;
wire [15:0] y;
factorial ut (x, y);
initial begin
x = 3'b011; #25;
x = 3'b100; #25;
x = 3'b101; #25;
x = 3'b001; #25;
x = 3'b010; #25;
x = 3'b110; #25;
x = 3'b111; #25;
end
endmodule
```

Output:



Waveform:


Wave - Default
Msgs
/factorial_tb/x
/factorial_tb/y
111
0001001110...
011
100
101
001
010
110
111
0000000000000110
0000000000011000
0000000001111000
0000000000000001
0000000000000010
0000001011010000
0001001110110000
Now
175 ps
Cursor 1
0 ps
0 ps
10 ps
20 ps
30 ps
40 ps
50 ps
60 ps
70 ps
80 ps
90 ps
100 ps
110 ps
120 ps
130 ps
140 ps
150 ps
160 ps
170 ps