Problem-1

* Source code

//module declaration with inputs and outputs

module p1(Y,present,clock,reset,input\_bit);

//Conventionally output is declared first, then input.

output Y;

input clock, reset,input\_bit;

output [2:0] present;

reg [2:0] next;

reg[2:0] present;

parameter [2:0] S0=3'b000;

parameter [2:0] S1=3'b001;

parameter [2:0] S2=3'b010;

parameter [2:0] S3=3'b011;

parameter [2:0] S4=3'b100;

initial

begin

present = 3'b000;

next = 3'b000;

end

always @(posedge clock)

// sequential logic

begin

if (reset) present <= S0;

else present <= next;

end

// combinational logic

always @(present or input\_bit) // combinational logic

begin

case(present)

S0: if (reset) next = S0;

else if(input\_bit) next = S1;

else next = S0;

S1: if (reset) next = S0;

else if(input\_bit) next = S2;

else next = S0;

S2: if (reset) next = S0;

else if(input\_bit) next = S2;

else next = S3;

S3: if (reset) next = S0;

else if(input\_bit) next = S4;

else next = S0;

S4: if (reset) next = S0;

else if(input\_bit) next = S2;

else next = S0;

endcase

end

// output logic described using continuous assignment

assign Y = (present == S4);

endmodule

* Test bench for p1.v

module tb\_p1();

reg clock, reset,input\_bit;

wire Y;

wire[2:0] present;

integer i;

p1 UUT(Y,present,clock,reset,input\_bit);

reg[20:1] Sequence = 20'b11101101001001101101;

initial reset = 1'b1;

always #10 clock = ~clock;

initial

begin

clock = 1'b1;

reset = 1'b0;

for (i=20;i>0;i=i-1)

begin

input\_bit = Sequence[i]; #20

$monitor("Sequence Bit = %b | Output = %b | reset = %b\n time=", input\_bit, Y, reset, $time);

end

end

endmodule

* Transcript

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 20

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 40

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 60

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 80

# Sequence Bit = 1 | Output = 1 | reset = 0

# time= 100

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 120

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 140

# Sequence Bit = 0 | Output = 1 | reset = 0

# time= 160

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 180

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 200

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 220

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 240

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 260

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 280

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 300

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 320

# Sequence Bit = 1 | Output = 1 | reset = 0

# time= 340

# Sequence Bit = 0 | Output = 0 | reset = 0

# time= 360

# Sequence Bit = 1 | Output = 0 | reset = 0

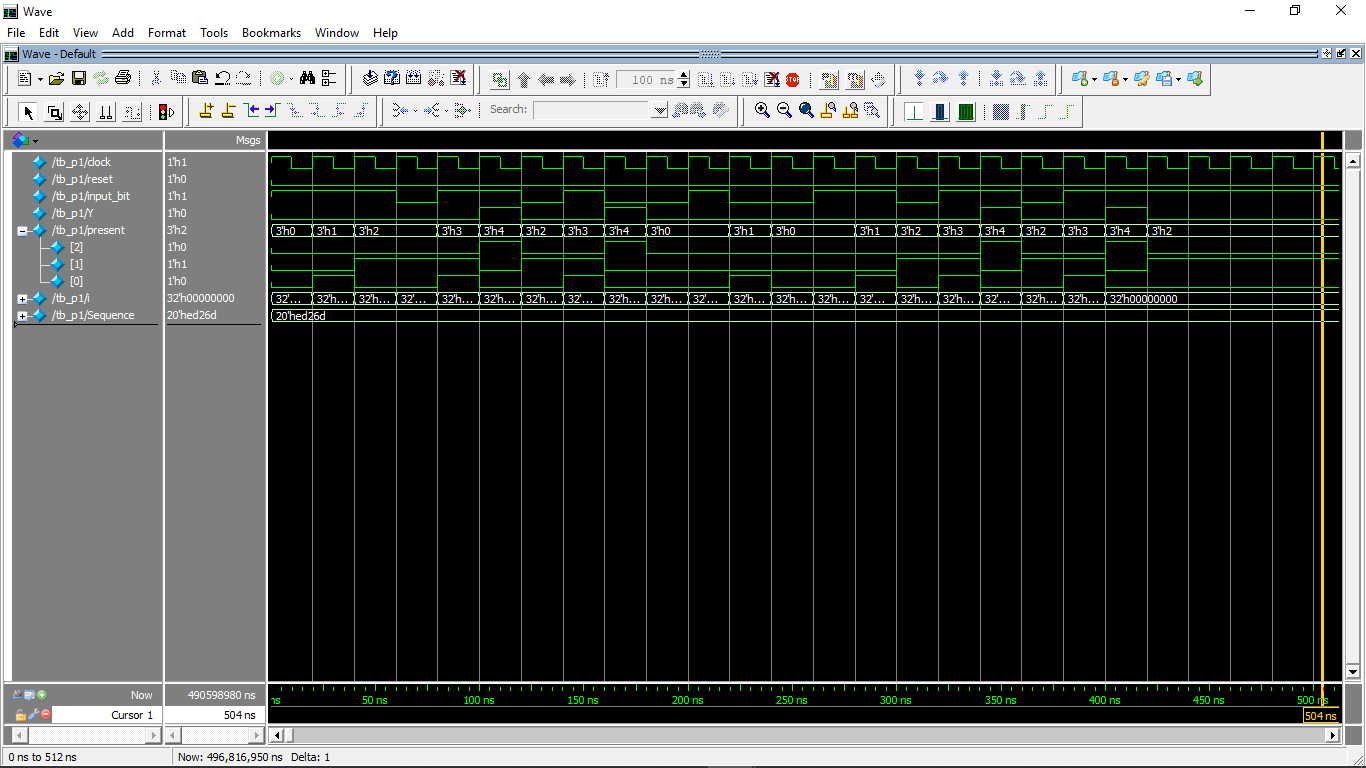
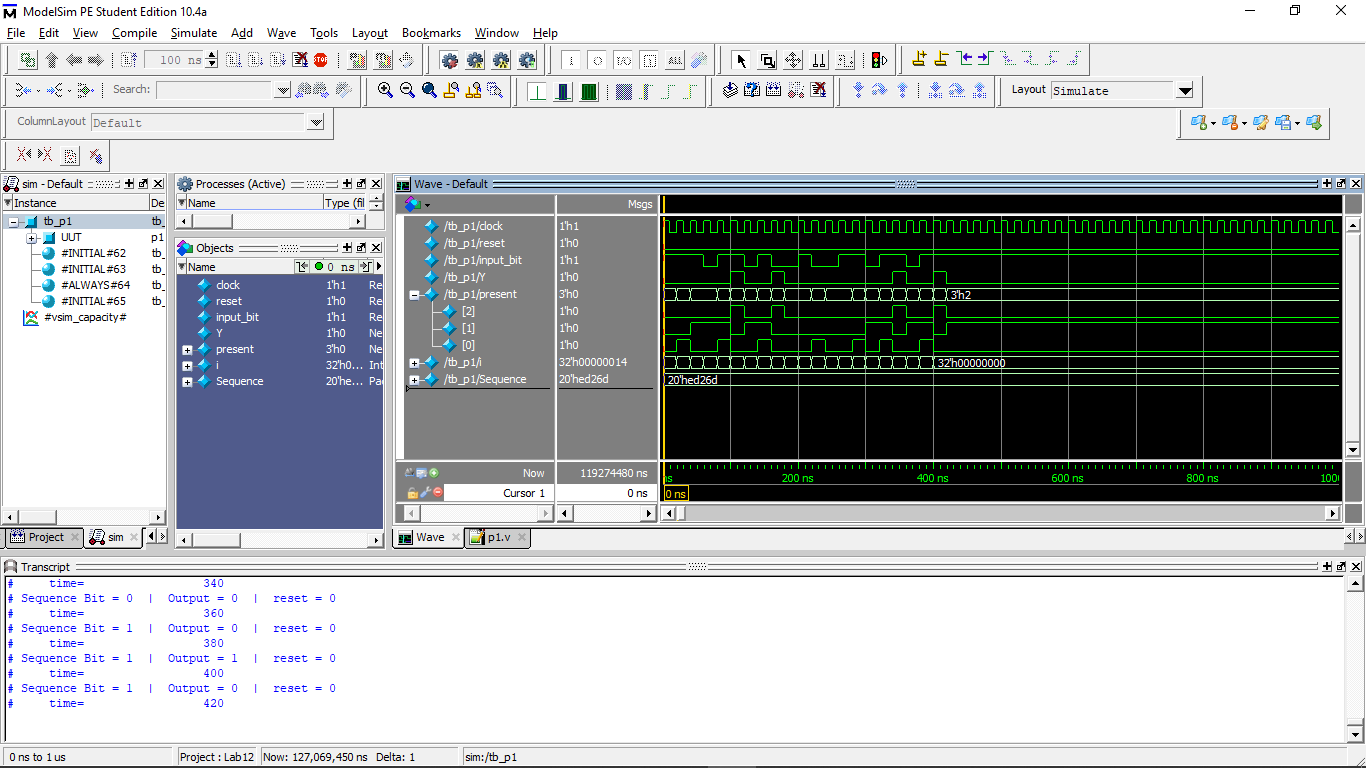
# time= 380

# Sequence Bit = 1 | Output = 1 | reset = 0

# time= 400

# Sequence Bit = 1 | Output = 0 | reset = 0

# time= 420



Problem 2

* Source Code

module p2(Q,upper,reset,clk); //counter module with inputs and outputs

//conventionally outputs are declared first, then input.

output reg [3:0]Q;

output upper;

input reset;

input clk;

always@(posedge clk)

begin

if(reset) //reset

Q = 4'b0000;

else

Q = Q + 4'b0001; //increment by 1

end

assign upper = Q[3];

endmodule

* Test Bench

module tb\_p2(); //Test Bench

reg clk, reset;

wire upper;

wire [3:0]Q;

p2 UUT(Q, upper, reset, clk);

initial begin //clock definition

clk = 0;

forever #5 clk = ~clk;

end

initial begin //simulation

$monitor("COUNT = %b", Q, " | UPPER = %b", upper);

reset = 1;

#20;

reset = 0;

//stop after 500 time units

#500 $finish;

end

endmodule

* Transcript

# COUNT = xxxx | UPPER = x

# COUNT = 0000 | UPPER = 0

# COUNT = 0001 | UPPER = 0

# COUNT = 0010 | UPPER = 0

# COUNT = 0011 | UPPER = 0

# COUNT = 0100 | UPPER = 0

# COUNT = 0101 | UPPER = 0

# COUNT = 0110 | UPPER = 0

# COUNT = 0111 | UPPER = 0

# COUNT = 1000 | UPPER = 1

# COUNT = 1001 | UPPER = 1

# COUNT = 1010 | UPPER = 1

# COUNT = 1011 | UPPER = 1

# COUNT = 1100 | UPPER = 1

# COUNT = 1101 | UPPER = 1

# COUNT = 1110 | UPPER = 1

# COUNT = 1111 | UPPER = 1

# COUNT = 0000 | UPPER = 0

# COUNT = 0001 | UPPER = 0

# COUNT = 0010 | UPPER = 0

# COUNT = 0011 | UPPER = 0

# COUNT = 0100 | UPPER = 0

# COUNT = 0101 | UPPER = 0

# COUNT = 0110 | UPPER = 0

# COUNT = 0111 | UPPER = 0

# COUNT = 1000 | UPPER = 1

# COUNT = 1001 | UPPER = 1

# COUNT = 1010 | UPPER = 1

# COUNT = 1011 | UPPER = 1

# COUNT = 1100 | UPPER = 1

# COUNT = 1101 | UPPER = 1

# COUNT = 1110 | UPPER = 1

# COUNT = 1111 | UPPER = 1

# COUNT = 0000 | UPPER = 0

# COUNT = 0001 | UPPER = 0

# COUNT = 0010 | UPPER = 0

# COUNT = 0011 | UPPER = 0

# COUNT = 0100 | UPPER = 0

# COUNT = 0101 | UPPER = 0

# COUNT = 0110 | UPPER = 0

# COUNT = 0111 | UPPER = 0

# COUNT = 1000 | UPPER = 1

# COUNT = 1001 | UPPER = 1

# COUNT = 1010 | UPPER = 1

# COUNT = 1011 | UPPER = 1

# COUNT = 1100 | UPPER = 1

# COUNT = 1101 | UPPER = 1

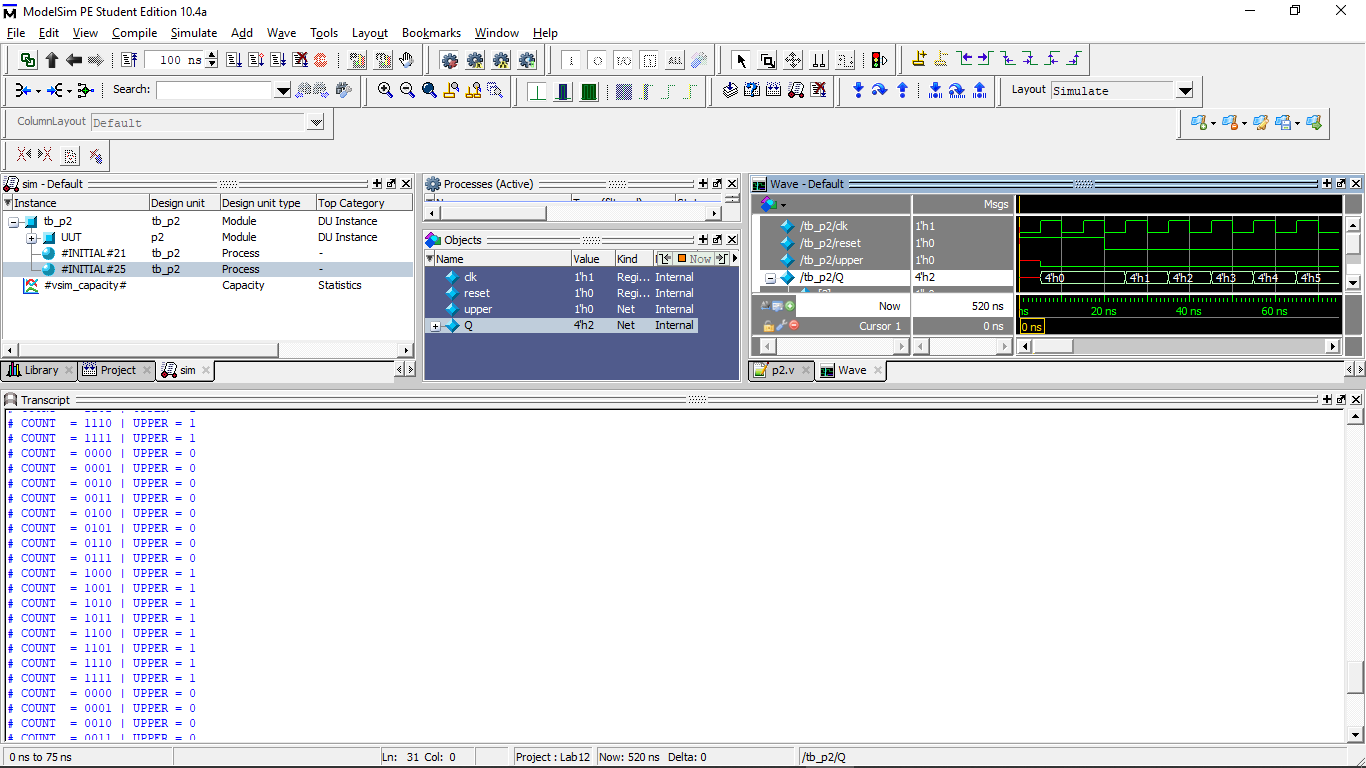
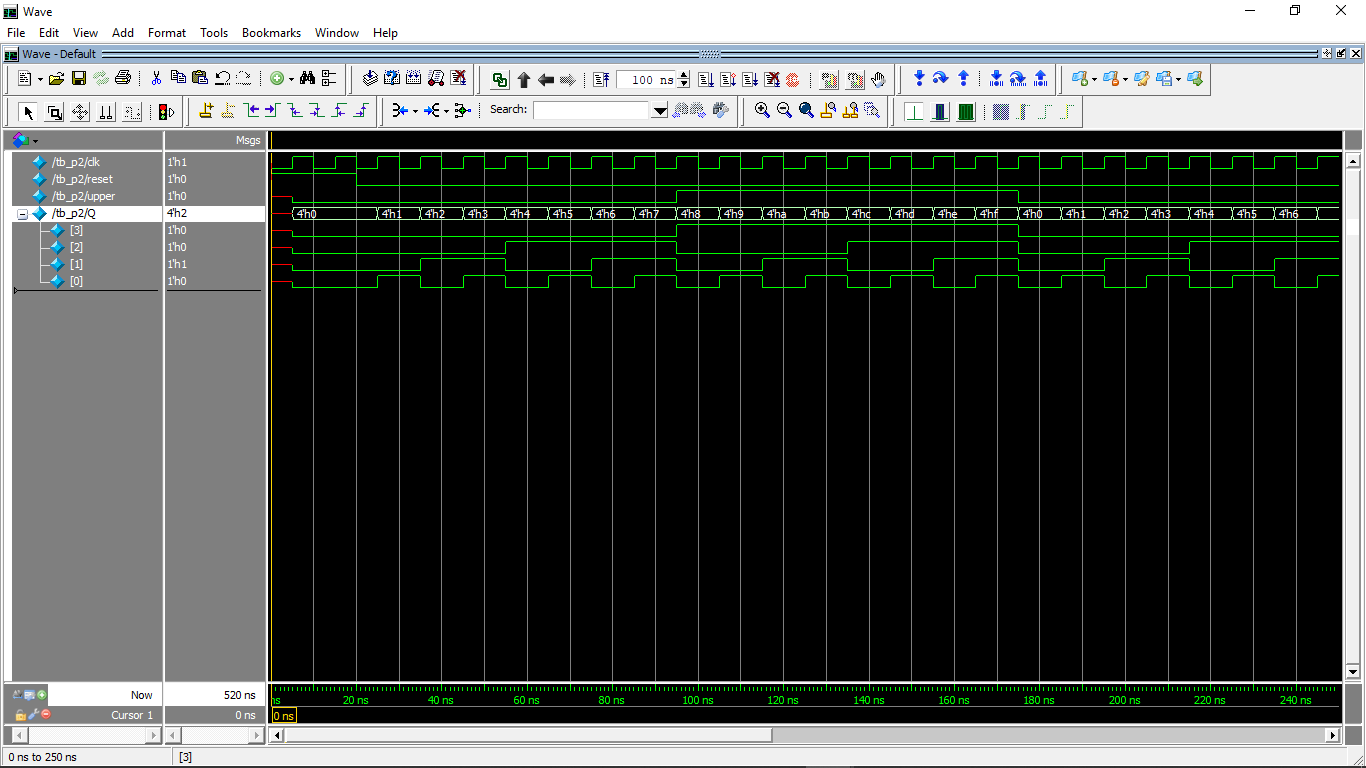
# COUNT = 1110 | UPPER = 1

# COUNT = 1111 | UPPER = 1

# COUNT = 0000 | UPPER = 0

# COUNT = 0001 | UPPER = 0

# COUNT = 0010 | UPPER = 0



Problem-3

* Source code

module p3(open,clk,reset, c0,c1);

output[2:0] open;

input clk, reset;

input c0,c1;

reg [2:0] Next;

reg[2:0] state;

parameter [2:0] S0= 3'b000;

parameter [2:0] S1= 3'b001;

parameter [2:0] S2 = 3'b010;

parameter [2:0] S3=3'b011;

parameter [2:0] S4=3'b100;

parameter [2:0] S5=3'b101;

parameter [2:0] S6=3'b110;

parameter [2:0] S7=3'b111;

always @(posedge clk) // sequential

begin

if (reset) state <= S0;

else state <= Next;

end

always @(state,c0,c1) // combinational

begin

case(state)

S0: if (!c0 && c1)

Next = S1;

else if (c0 && !c1)

Next = S7;

else if(c0 && c1)

Next = S2;

S1: if (!c0 && c1)

Next = S2;

else if (c0 && !c1)

Next = S0;

else if(c0 && c1)

Next = S3;

S2: if (!c0 && c1)

Next = S3;

else if (c0 && !c1)

Next = S1;

else if(c0 && c1)

Next = S4;

S3: if (!c0 && c1)

Next = S4;

else if (c0 && !c1)

Next = S2;

else if(c0 && c1)

Next = S5;

S4:if (!c0 && c1)

Next = S5;

else if (c0 && !c1)

Next = S3;

else if(c0 && c1)

Next = S6;

S5:if (!c0 && c1)

Next = S6;

else if (c0 && !c1)

Next = S4;

else if(c0 && c1)

Next = S7;

S6:if (!c0 && c1)

Next = S7;

else if (c0 && !c1)

Next = S5;

else if(c0 && c1)

Next = S0;

S7:if (!c0 && c1)

Next = S0;

else if (c0 && !c1)

Next = S6;

else if(c0 && c1)

Next = S1;

endcase end

// output logic described using continuous assignment

assign open = Next;

endmodule

* Test Bench

module tb\_p3();

reg reset;

wire[2:0] open;

reg clk=0;

reg c0,c1;

p3 uut (open,clk,reset, c0,c1);

always #5 clk = ~clk;

initial

begin

reset=1'b1; #10;

reset=1'b0; c0=1'b0;c1=1'b1;#10;

reset=1'b0; c0=1'b1;c1=1'b0;#10;

reset=1'b0; c0=1'b1;c1=1'b1;#10;

reset=1'b0; c0=1'b0;c1=1'b0;#10;

reset=1'b1; #10;

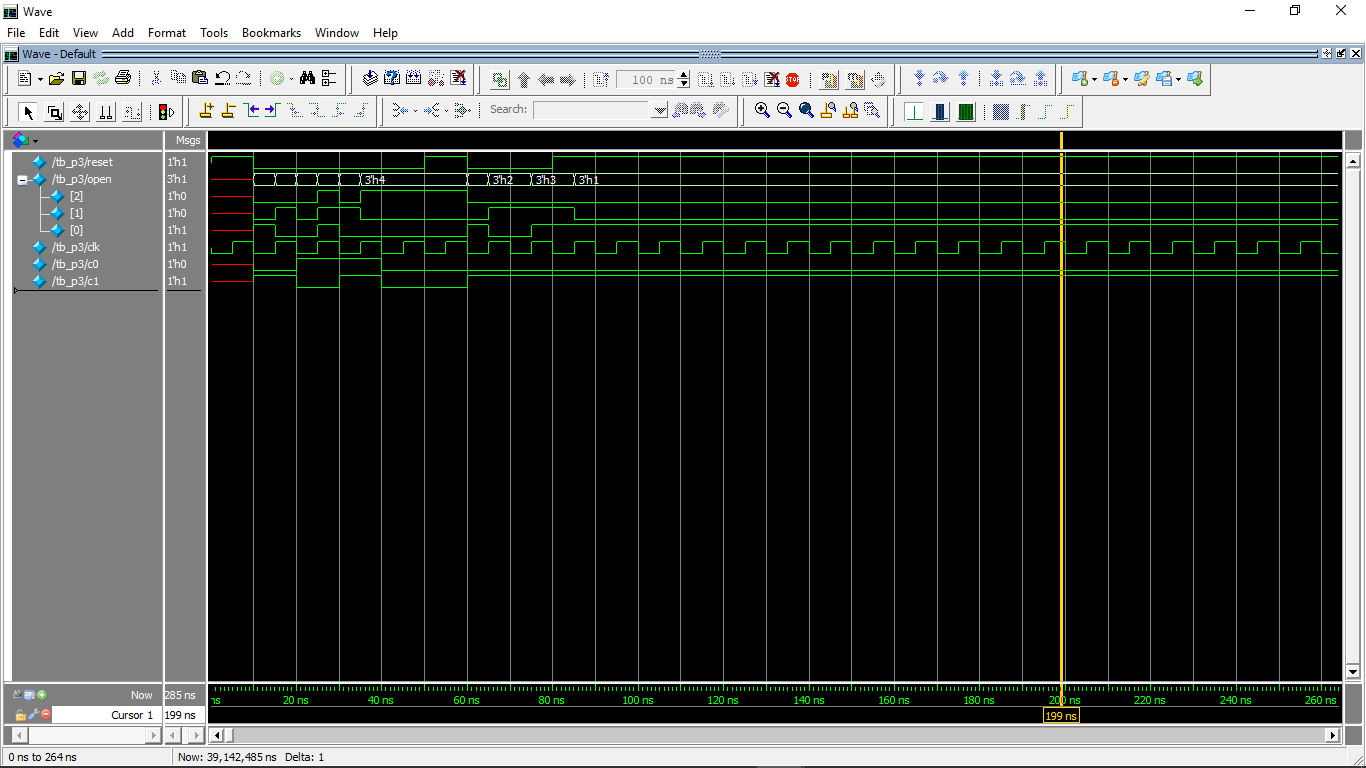
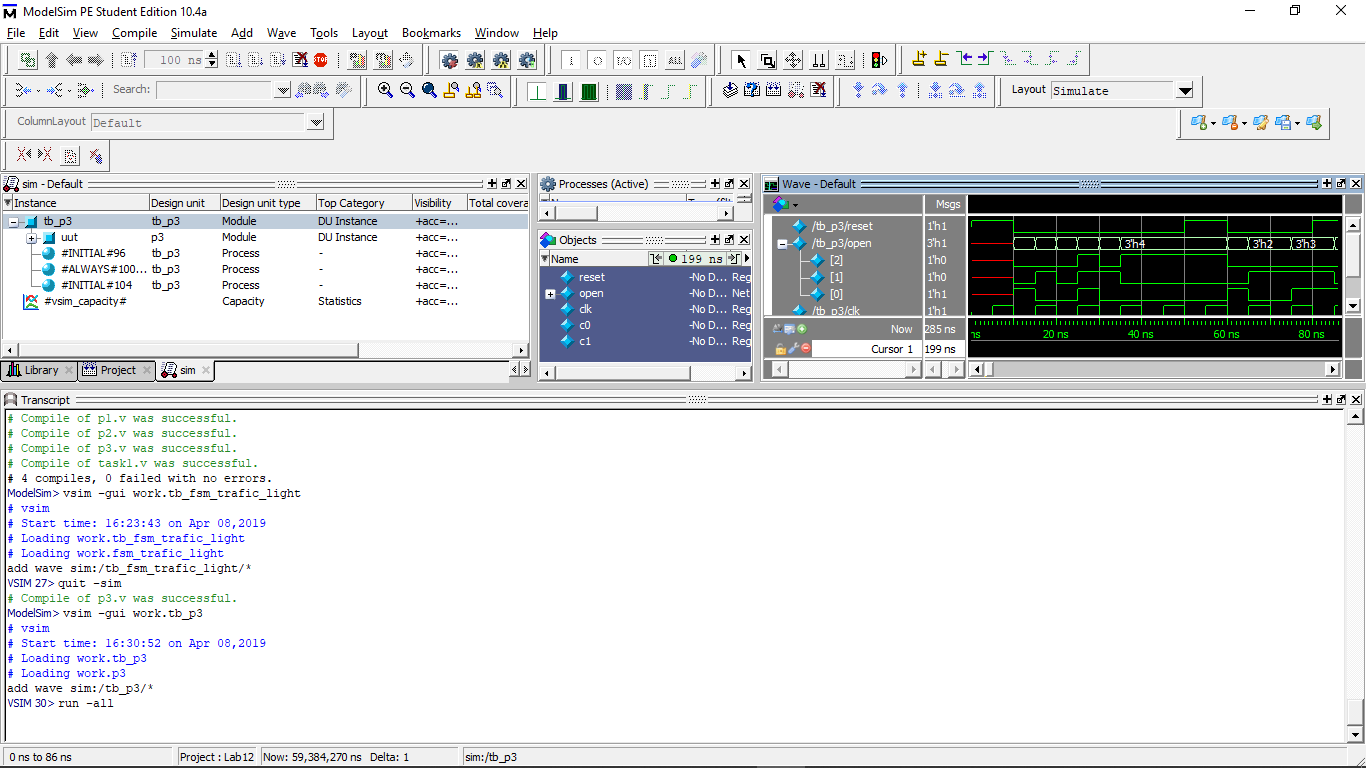
reset=1'b0; c0=1'b0;c1=1'b1;#10;

reset=1'b0; c0=1'b0;c1=1'b1;#10;

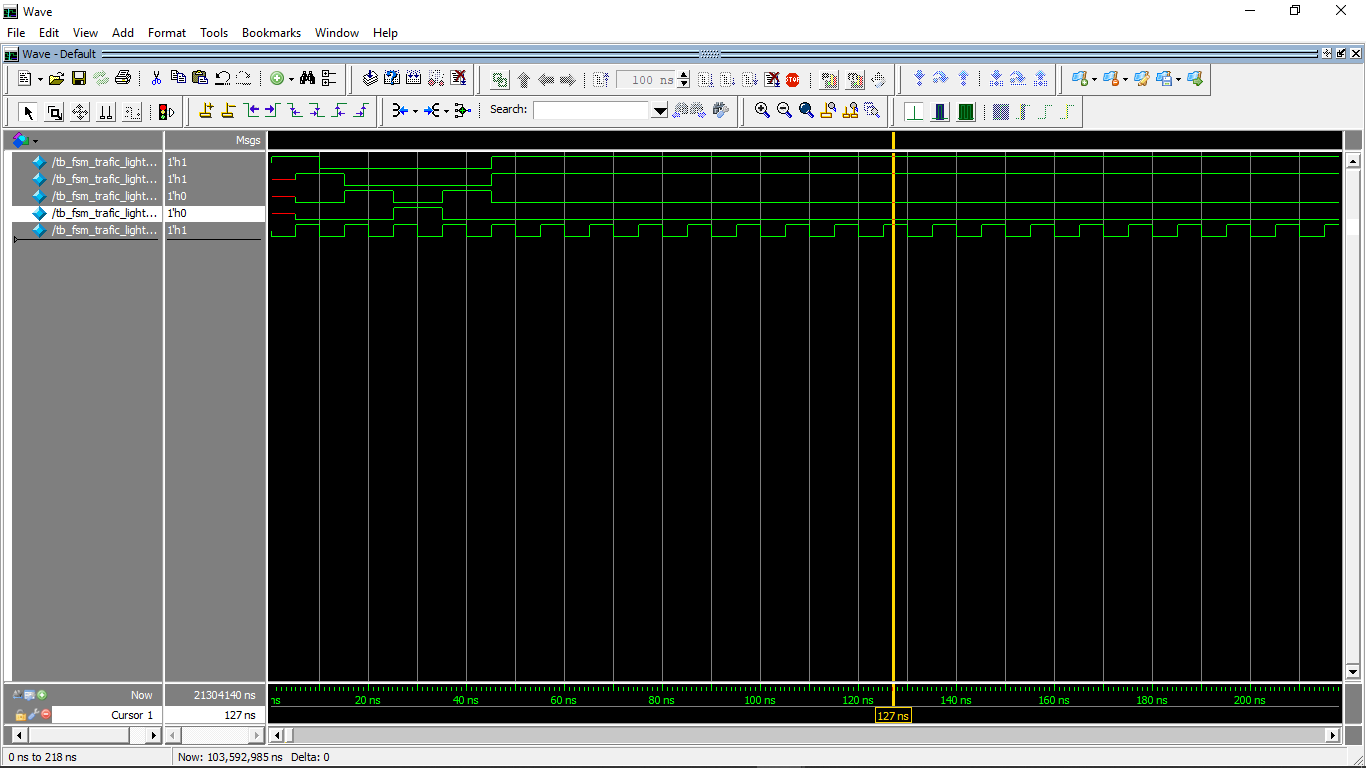
reset=1'b1; #10;

end

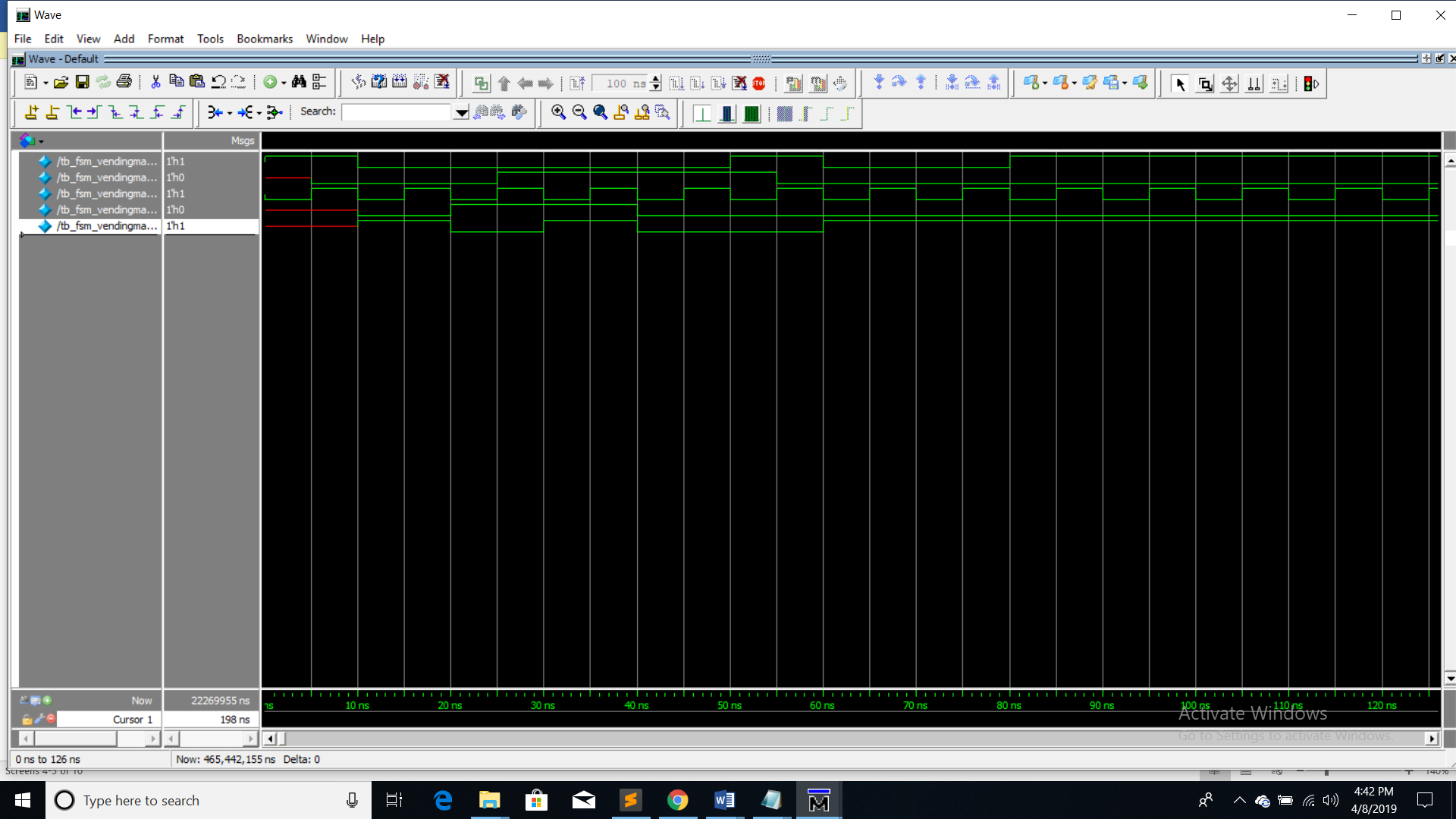
endmodule

-

Task-1



TASK-2



TASK-3

