EC204

Digital System Design Lab

Lab – 9

Utkarsh R Mahajan

201EC164

1] A digital system has a single input X and a single output Y. The system is to analyse each sequence of four binary digits and produce the corresponding 2's complement of the 4-bit sequence. Assume each 4 bit sequence is occurring with the LSB first. Draw the minimal state diagram for the system and model in Verilog.

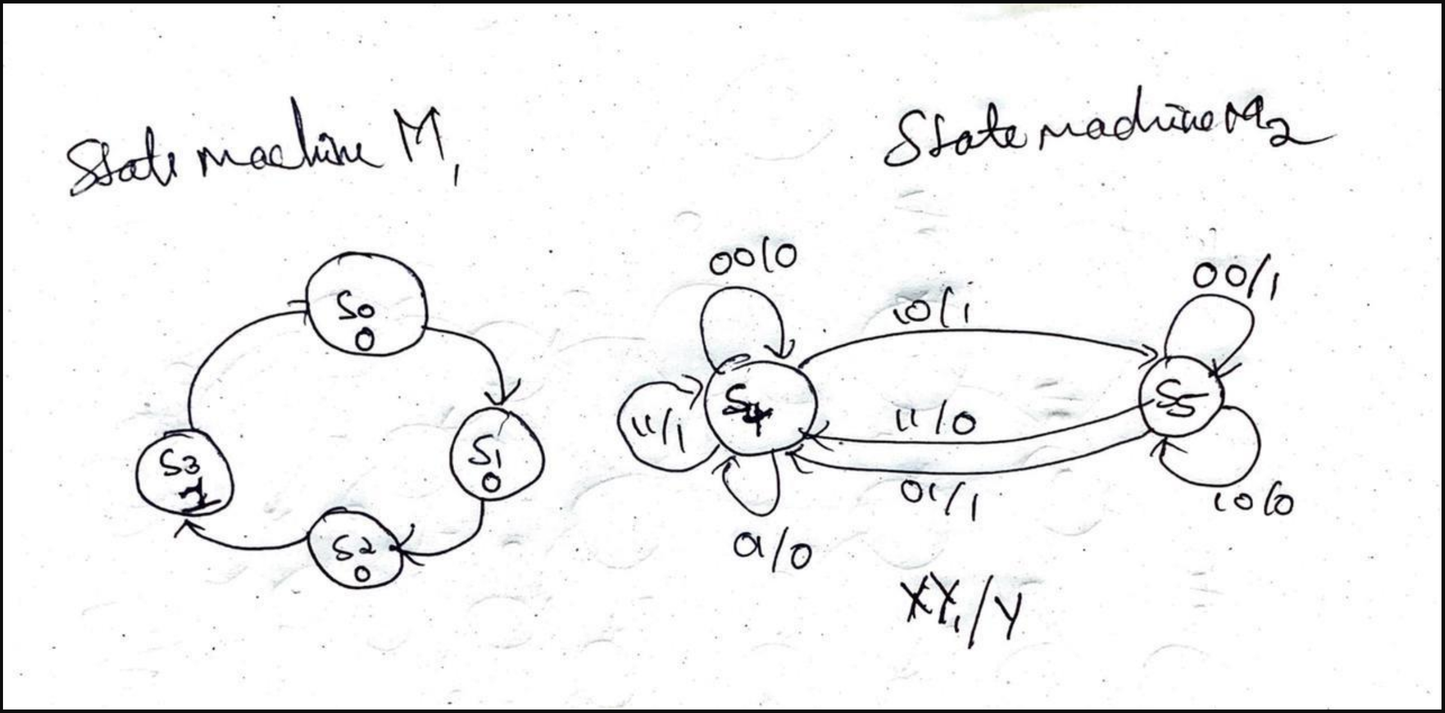
e.g. X = 0101000111000100

Z = 0110000110110111

->

To simplify, we will make use of 2 state machines and then link them together.

After analysing the sequence we get the following state diagrams:



From the following state diagram, we get the following:

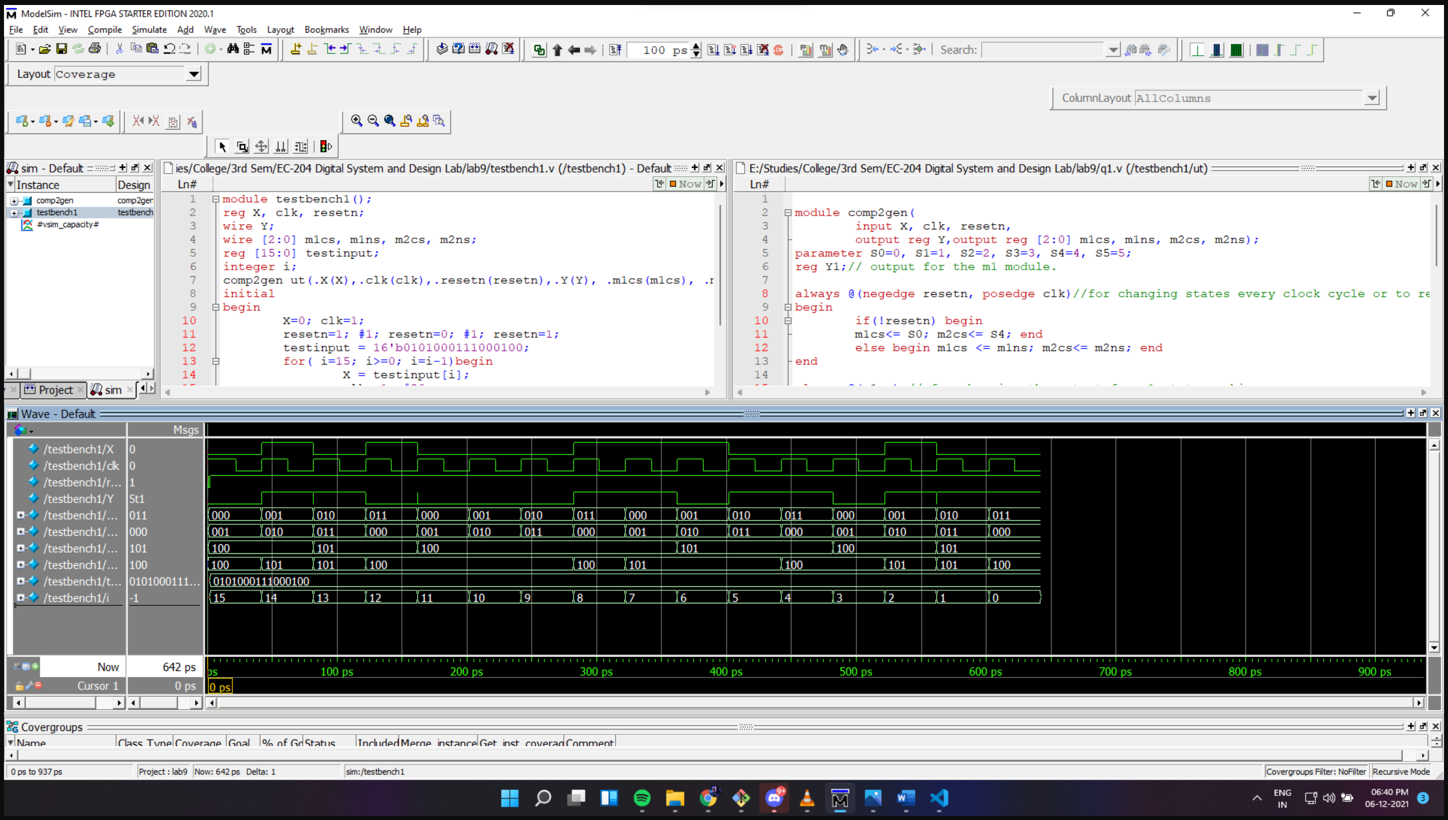
Verilog Code:

```
module comp2gen(
    input X, clk, resetn,
    output reg Y,output reg [2:0] m1cs, m1ns, m2cs, m2ns);
parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5;
reg Y1;// output for the m1 module.

always @(negedge resetn, posedge clk)//for changing states every clock cycle or to reset.
begin
    if(!resetn) begin
    m1cs<= S0; m2cs<= S4; end
    else begin m1cs <= m1ns; m2cs<= m2ns; end
end
```

```
always @(m1cs) // for changing the output for m1 state machine.
begin
    m1ns =m1cs; Y1= 0; //default values for m1ns and output Y1 of the m1 state machine.
    case(m1cs)
    S0: m1ns = S1;
    S1: m1ns = S2;
    S2: m1ns = S3;
    S3: begin
        m1ns = S0;
        Y1=1;
        end
    endcase
end

always @(*) // for changing the output for m2 state machine.
begin
    m2ns=m2cs; Y=0; //default values for m2ns and output Y of the m2 state machine.
    case(m2cs)
    S4: begin
    Y=X;
    if(X&&~Y1) m2ns=S5; else m2ns=S4;
    end
    S5: begin
    Y=~X;
    if(Y1) m2ns=S4; else m2ns=S5;
    end
    default:m2ns=S4;
    endcase
end
endmodule
```

Testbench Code:

```
module testbench1();
reg X, clk, resetn;
wire Y;
wire [2:0] m1cs, m1ns, m2cs, m2ns;
reg [15:0] testinput;
integer i;

comp2gen ut(.X(X),.clk(clk),.resetn(resetn),.Y(Y), .m1cs(m1cs), .m1ns(m1ns), .m2cs(m2cs),
.m2ns(m2ns));
initial
begin
    X=0; clk=1;
    resetn=1; #1; resetn=0; #1; resetn=1;
    testinput = 16'b0101000111000100;
    for( i=15; i>=0; i=i-1)begin
        X = testinput[i];
        clk =1; #20;
clk =0; #20;
    end

end
endmodule
```



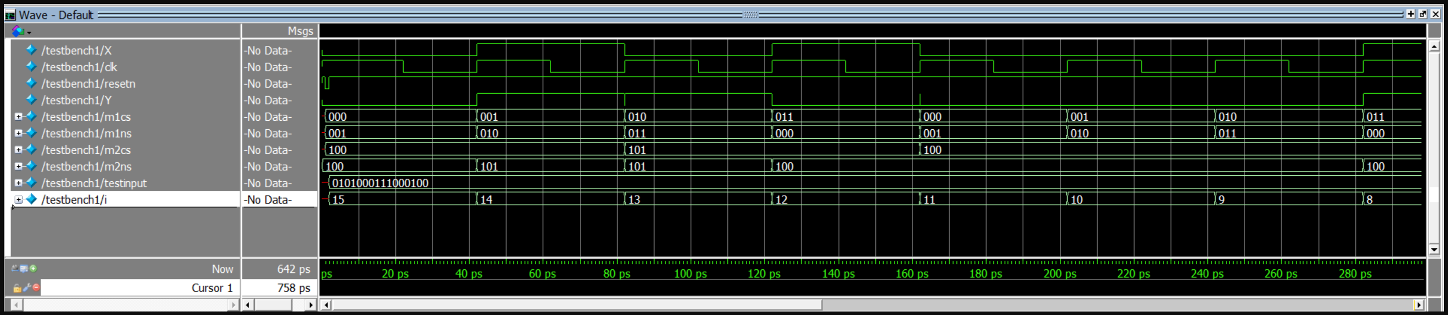
ies/College/3rd Sem/EC-204 Digital System and Design Lab/lab9/testbench1.v (/testbench1) - Default

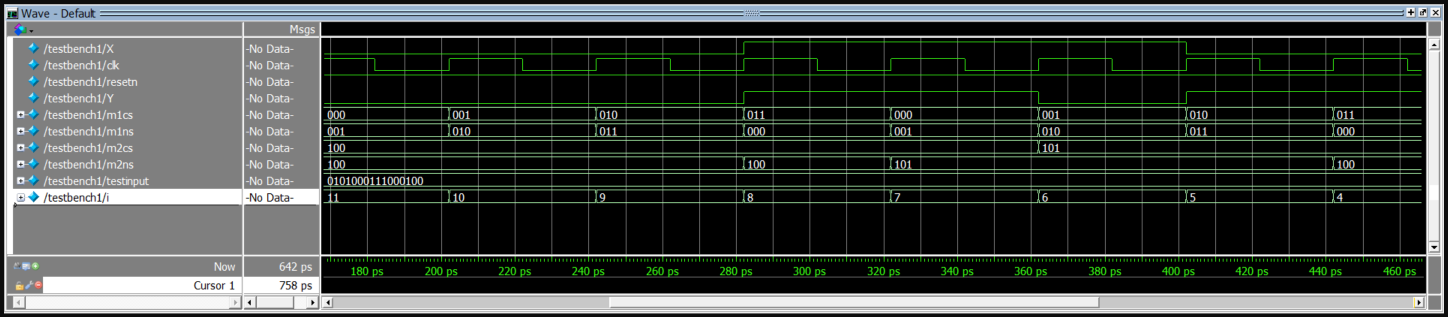
```
module testbench1();
reg X, clk, resetn;
wire Y;
wire [2:0] m1cs, m1ns, m2cs, m2ns;
reg [15:0] testinput;
integer i;
comp2gen ut(.X(X),.clk(clk),.resetn(resetn),.Y(Y), .m1cs(m1cs), .r
initial
begin
        X=0; clk=1;
        resetn=1; #1; resetn=0; #1; resetn=1;
        testinput = 16'b0101000111000100;
        for( i=15; i>=0; i=i-1)begin
                X = testinput[i];
```

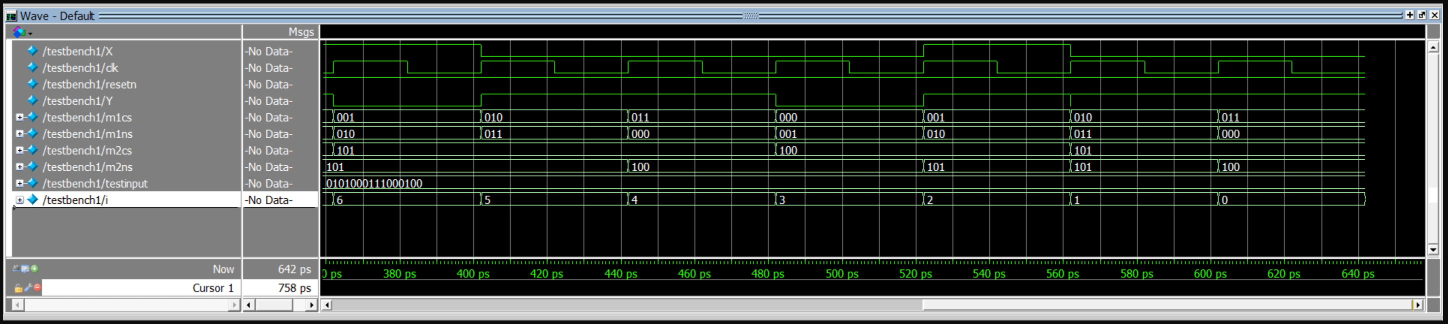
E:/Studies/College/3rd Sem/EC-204 Digital System and Design Lab/lab9/q1.v (/testbench1/ut)

```

module comp2gen(
        input X, clk, resetn,
        output reg Y,output reg [2:0] m1cs, m1ns, m2cs, m2ns);
parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5;
reg Y1;// output for the m1 module.

always @(negedge resetn, posedge clk)//for changing states every clock cycle or to re
begin
        if(!resetn) begin
        m1cs<= S0; m2cs<= S4; end
        else begin m1cs <= m1ns; m2cs<= m2ns; end
end

```

Output wave:






Wave - Default
Msgs
/testbench1/X -No Data-
/testbench1/clk -No Data-
/testbench1/resetn -No Data-
/testbench1/Y -No Data-
/testbench1/m1cs -No Data- 001 010 011 000 001 010 011
/testbench1/m1ns -No Data- 010 011 000 001 010 011 000
/testbench1/m2cs -No Data- 101 100 101
/testbench1/m2ns -No Data- 101 100 101 101 100
/testbench1/testinput -No Data- 0101000111000100
/testbench1/i -No Data- 6 5 4 3 2 1 0
Now 642 ps
Cursor 1 758 ps
0 ps 380 ps 400 ps 420 ps 440 ps 460 ps 480 ps 500 ps 520 ps 540 ps 560 ps 580 ps 600 ps 620 ps 640 ps

2] Implement a serial adder in Verilog to add two 8 bit numbers using a single full adder and registers

-> for the serial adder we will make use of a D flip flop for storing the carry of each bit. 3 shift registers, 2 for taking inputs and 1 for storing the output sequentially. For the first 8 cycles of clock, it will take input and for the next 8 cycles the output will be computed and shifted to the output shift register. The resetn is a negative edge triggered reset which will set everything to default values.

Register count stores the counts of cycles that have taken place so that after complete calculation of the addition of 8bit numbers it will be active.

```
// sum output is at output reg sum and the carry of the sum is at output reg C.
module serial_adder(
    input ain, bin, clk, start, resetn,
    output reg [7:0] sum, output reg done, output reg c);
//wires for connecting different modules
wire [7:0] rxQ, ryQ, rsQ;wire fas, fac, dffc;
reg [3:0] count =4'b1111;
reg [7:0] D =0;
reg load = 0;
//connecting wires from modules to output registers.
assign sum = rsQ;
assign c = fac;
shiftreg regX(.D(D), .load(load), .shiftR(!done),.lin(ain), .clk(clk), .Q(rxQ));
shiftreg regY(.D(D), .load(load), .shiftR(!done),.lin(bin), .clk(clk), .Q(ryQ));
shiftreg regSum(.D(D), .load(load), .shiftR(!done),.lin(fas), .clk(clk), .Q(rsQ));
fulladder fa(.a(rxQ[0]), .b(ryQ[0]), .cin(dffc), .sum(fas), .cout(fac));
dff da(.d(fac), .reset(load), .clk(clk), .q(dffc));
//for decrementing counter and making a reset button for entire module.
always @(posedge clk, negedge resetn)
begin
    if(!resetn) begin
    load<= 1;
    count<=15;
    done<=0;
    end else if(start) begin
    load<= 0;
    if(count >0) begin done <= 0;
    count<= count -1;
    end
    else done <=1;
    end
end
endmodule
//single bit full adder module required.
module fulladder(input a, b, cin, output sum, cout);
    assign sum = cin^(a^b);
    assign cout = (a&b) | cin&(a^b);
endmodule
// D flip flop
module dff(input d, reset, clk, output reg q);
    always@(posedge clk, posedge reset)
    begin
    if(reset) q=0;
    else begin
        q=d;
    end
    end
endmodule
//8 bit shift register with load support.
module shiftreg (input [7:0]D, input load, shiftR, lin, clk, output reg [7:0]Q);
always @(load)
begin
    if(load) Q<=D;
end
always @(posedge clk)
begin
    if(shiftR) Q <= {lin, Q[7:1]};
end
endmodule
```

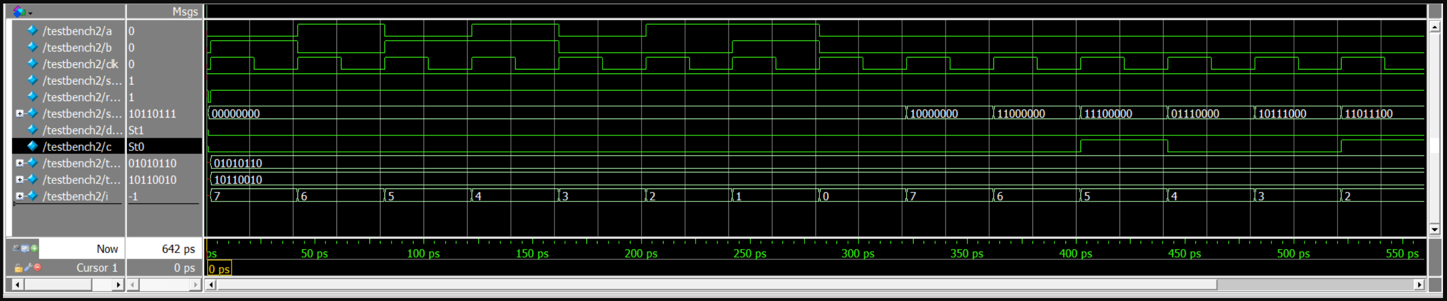
TestBench:

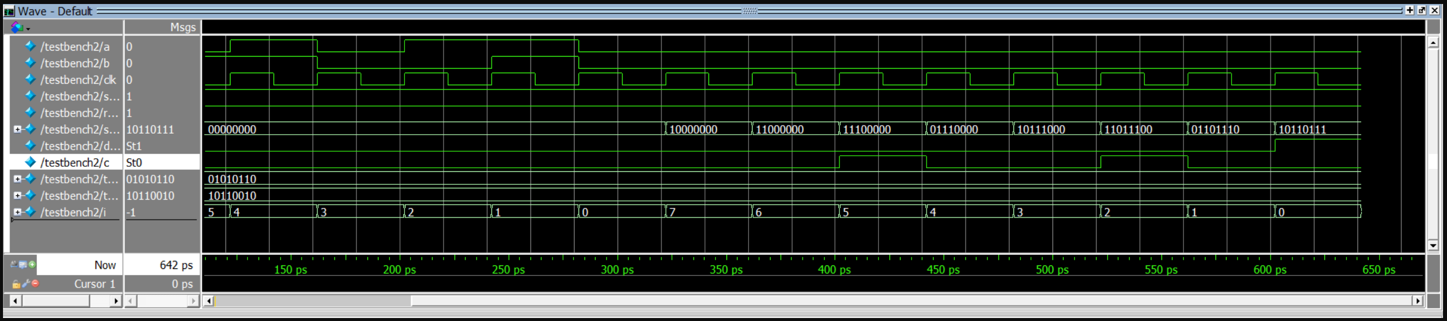
```
module testbench2();
reg a, b, clk, start, resetn;
wire [7:0]sum;
wire done, c;
reg [7:0] testinputx, testinputy;
integer i;

serial_adder ut(.ain(a), .bin(b), .clk(clk), .start(start), .resetn(resetn),.sum(sum),
.done(done), .c(c));
initial
begin
//giving starting default values
    a=0; b=0; start=1; clk=0;
//setting teh module at reset.
    resetn=1; #1; resetn=0; #1; resetn=1;
//input values for x and y, from left to right.
```

```
    testinputx = 8'b01010110;
    testinputy = 8'b10110010;
//cyclic inputs x and y transversing the above values.
    for( i=7; i>=0; i=i-1)begin
        a = testinputx[i];
        b = testinputy[i];
        clk =1; #20;
        clk =0; #20;
    end
//additional cycles for computing the adder result.
    for( i=7; i>=0; i=i-1)begin
        clk =1; #20;
        clk =0; #20;
    end

end
endmodule
```

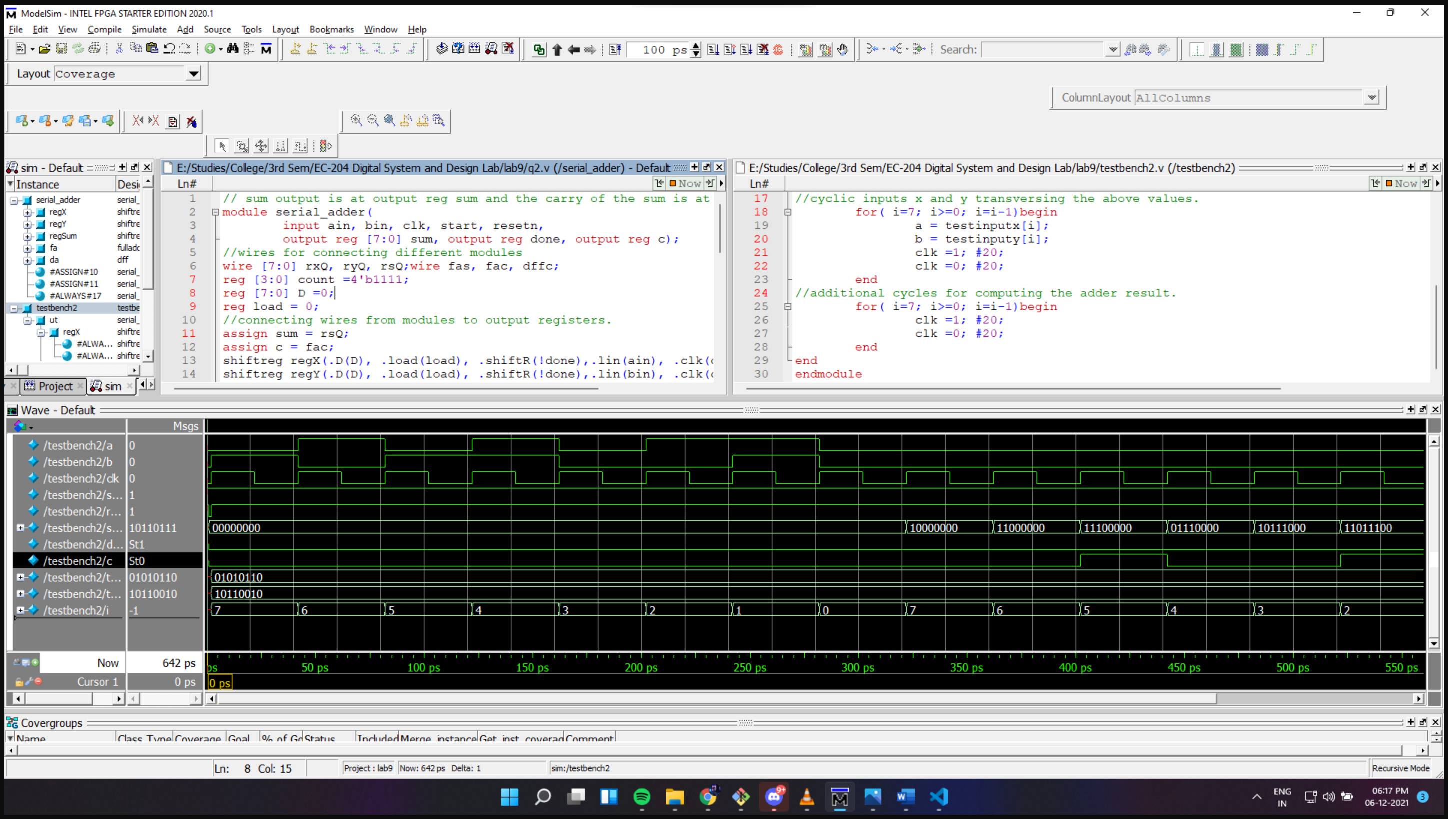
For the input given in the example, we can see the following output.

Output wave:





Full Screenshot:



E:/Studies/College/3rd Sem/EC-204 Digital System and Design Lab/lab9/q2.v (/serial_adder) - Default

```verilog
// sum output is at output reg sum and the carry of the sum is at
module serial_adder(
        input ain, bin, clk, start, resetn,
        output reg [7:0] sum, output reg done, output reg c);
//wires for connecting different modules
wire [7:0] rxQ, ryQ, rsQ;wire fas, fac, dffc;
reg [3:0] count =4'b1111;
reg [7:0] D =0;
reg load = 0;
//connecting wires from modules to output registers.
assign sum = rsQ;
assign c = fac;
shiftreg regX(.D(D), .load(load), .shiftR(!done),.lin(ain), .clk(
shiftreg regY(.D(D), .load(load), .shiftR(!done),.lin(bin), .clk(
```

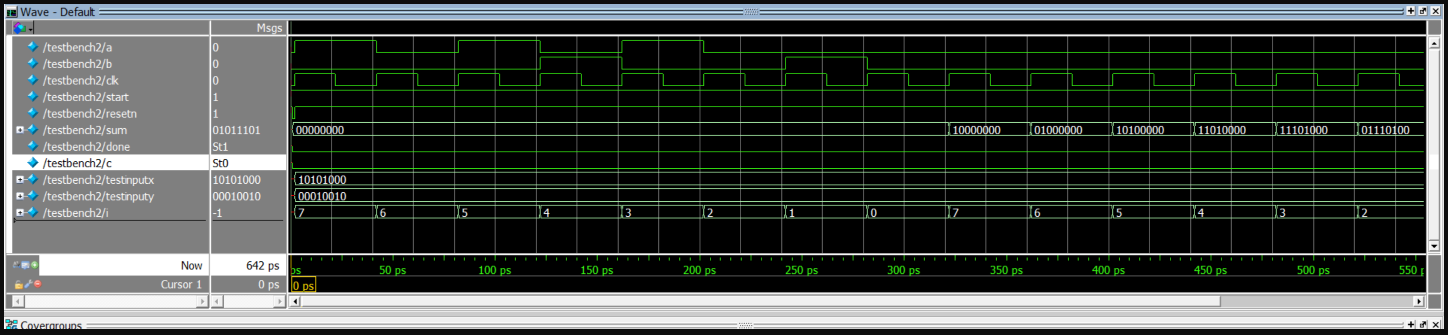
E:/Studies/College/3rd Sem/EC-204 Digital System and Design Lab/lab9/testbench2.v (/testbench2)

```verilog
//cyclic inputs x and y transversing the above values.
        for( i=7; i>=0; i=i-1)begin
                a = testinputx[i];
                b = testinputy[i];
                clk =1; #20;
                clk =0; #20;
        end
//additional cycles for computing the adder result.
        for( i=7; i>=0; i=i-1)begin
                clk =1; #20;
                clk =0; #20;
        end
end
endmodule
```

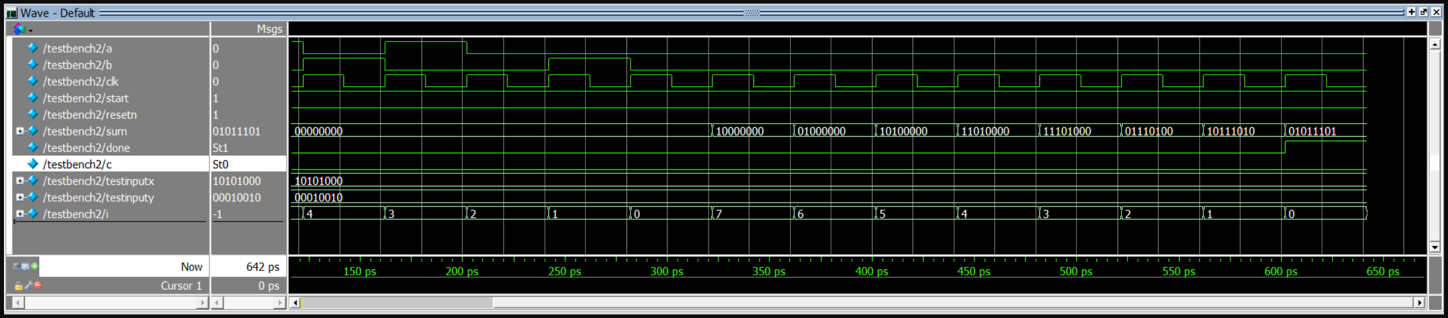
```
/testbench2/a         0
/testbench2/b         0
/testbench2/clk       0
/testbench2/s...      1
/testbench2/r...      1
/testbench2/s...      10110111
/testbench2/d...      St1
/testbench2/c         St0
/testbench2/t...      01010110
/testbench2/t...      10110010
/testbench2/i         -1
Now                   642 ps
Cursor 1              0 ps
```

For input: ain=10101000 , bin= 00010010

Output wave:



```
/testbench2/a            0
/testbench2/b            0
/testbench2/clk          0
/testbench2/start        1
/testbench2/resetn       1
/testbench2/sum          01011101
/testbench2/done         St1
/testbench2/c            St0
/testbench2/testinputx   10101000
/testbench2/testinputy   00010010
/testbench2/i            -1
Now                      642 ps
Cursor 1                 0 ps
```



```
/testbench2/a            0
/testbench2/b            0
/testbench2/clk          0
/testbench2/start        1
/testbench2/resetn       1
/testbench2/sum          01011101
/testbench2/done         St1
/testbench2/c            St0
/testbench2/testinputx   10101000
/testbench2/testinputy   00010010
/testbench2/i            -1
Now                      642 ps
Cursor 1                 0 ps
```