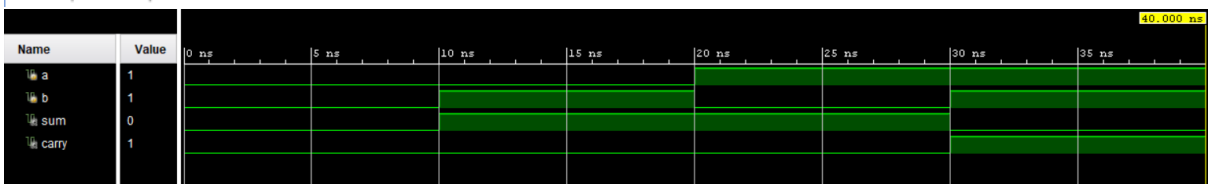
HALF ADDER:

```
module Half_adder (
    input a,b,
    output sum,carry
);

assign sum = a ^ b;
assign carry = a & b;

endmodule
```

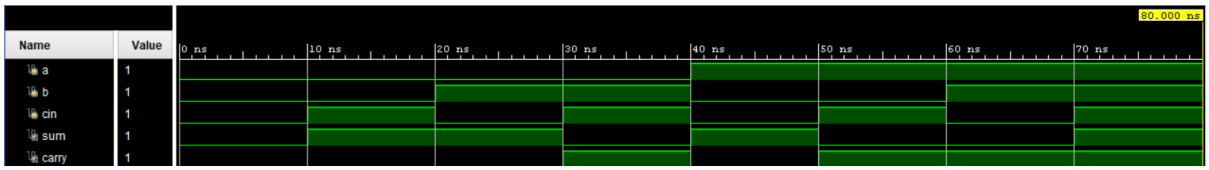
```
1   module half_adder_tb;
2   reg a,b;
3   wire sum,carry;
4
5   Half_adder uut(a,b,sum,carry);
6
7   initial begin
8   a = 0; b = 0;
9   #10
10  b = 0; b = 1;
11  #10
12  a = 1; b = 0;
13  #10
14  b = 1; b = 1;
15  #10
16  $finish();
17  end
18
19  endmodule
```



FULL ADDER:

```
module full_adder (
    input a,b,cin,
    output sum,carry
);
    assign sum = a^b^cin ;
    assign carry = (a&b)|(b&cin)|(cin&a);
endmodule
```

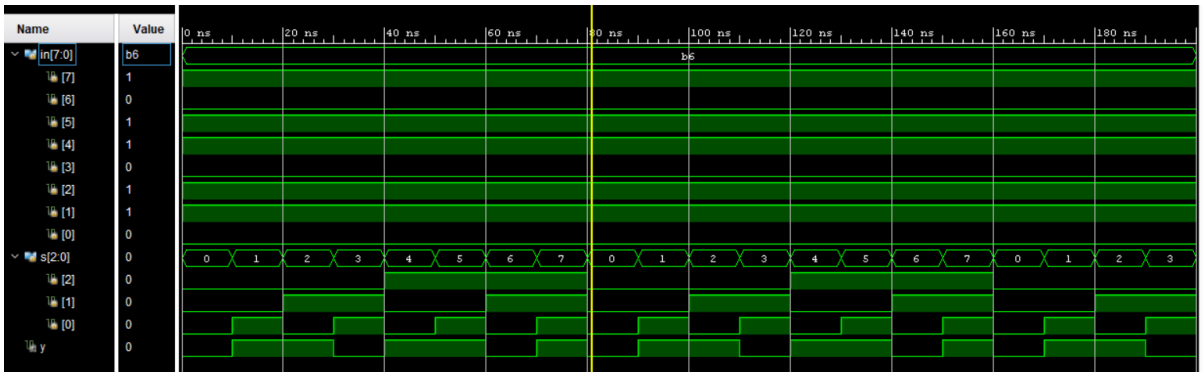
```
module full_adder_tb;
reg a,b,cin;
wire sum,carry;

full_adder uut(a,b,cin,sum,carry);

initial begin
a = 0; b = 0; cin = 0;
#10
a = 0; b = 0; cin = 1;
#10
a = 0; b = 1; cin = 0;
#10
a =0; b = 1; cin = 1;
#10
a = 1; b = 0; cin = 0;
#10
a = 1; b = 0; cin = 1;
#10
a = 1; b = 1; cin = 0;
#10
a =1; b = 1; cin = 1;
#10
$finish();
end

endmodule
```



MULTIPLEXER:

```
module mux(in,s,y);
output  y;
input [7:0] in;
input [2:0] s;
reg y;
always @(s)
begin
case(s)
3'b000 : y=in[0];
3'b001 : y=in[1];
3'b010 : y=in[2];
3'b011 : y=in[3];
3'b100 : y=in[4];
3'b101 : y=in[5];
3'b110 : y=in[6];
3'b111 : y=in[7];
endcase
end
endmodule
```

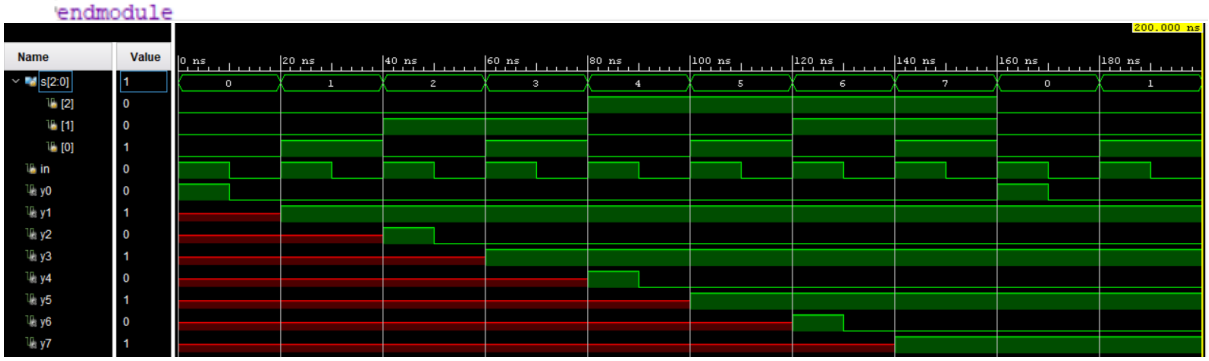
```
module mux_tb();
reg [7:0]in;
reg [2:0]s;
wire y;
mux m1(in,s,y);
initial
begin
in=8'b10110110;
s[0]=1'b0;
s[1]=1'b0;
s[2]=1'b0;
end
always #40  s[2]=~s[2];
always #20  s[1]=~s[1];
always #10  s[0]=~s[0];
initial
begin
#200 $finish;
end
endmodule
```



DEMULTIPLEXER:

```
module demux(y0,y1,y2,y3,y4,y5,y6,y7,s,in);
output y0,y1,y2,y3,y4,y5,y6,y7;
input [2:0]s;
input in;
reg y0,y1,y2,y3,y4,y5,y6,y7;
always @(s|in)
begin
case(s)
3'b000 : y0=in;
3'b001 : y1=in;
3'b010 : y2=in;
3'b011 : y3=in;
3'b100 : y4=in;
3'b101 : y5=in;
3'b110 : y6=in;
3'b111 : y7=in;
endcase
end
endmodule
```

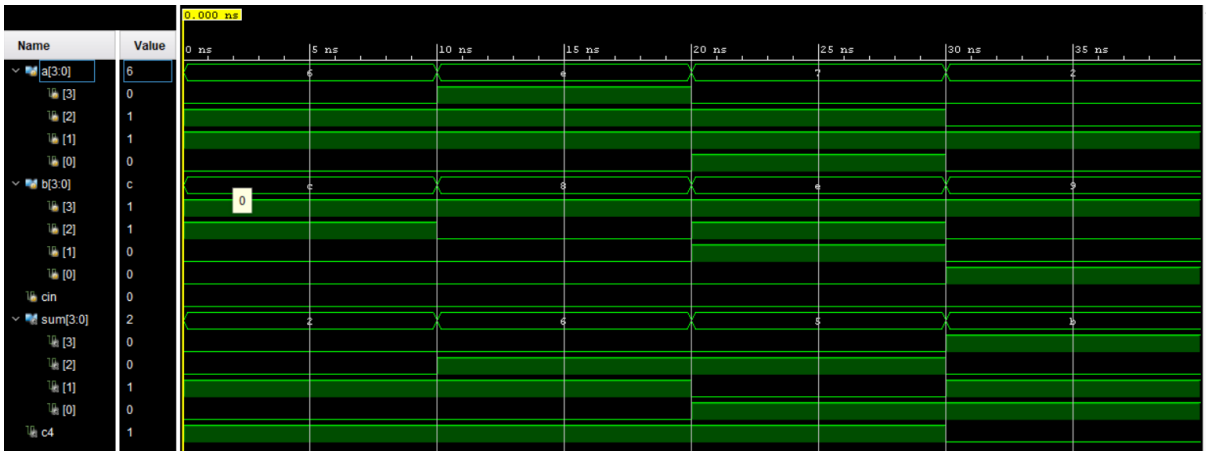
```
module demux_tb();
reg [2:0]s;
reg in;
wire y0,y1,y2,y3,y4,y5,y6,y7;
demux d1(y0,y1,y2,y3,y4,y5,y6,y7,s,in);
initial
begin
in=1;
s=3'b000;
end
always #10 in=~in;
always #80 s[2]=~s[2];
always #40 s[1]=~s[1];
always #20 s[0]=~s[0];
initial
begin
#200 $finish;
end
endmodule
```



FOUR BIT ADDER:

```
module four_bit_adder(
    input [3:0]a,b,
    input cin,
    output [3:0]sum,
    output c4);

wire c1,c2,c3;      //Carry out of each full adder

full_adder fa0(a[0],b[0],cin,sum[0],c1);
full_adder fa1(a[1],b[1],c1,sum[1],c2);
full_adder fa2(a[2],b[2],c2,sum[2],c3);
full_adder fa3(a[3],b[3],c3,sum[3],c4);

endmodule

module full_adder (
    input a,b,c,
    output sum,carry
);
    assign sum = a^b^c ;
    assign carry = (a&b)|(b&c)|(c&a);
endmodule
```

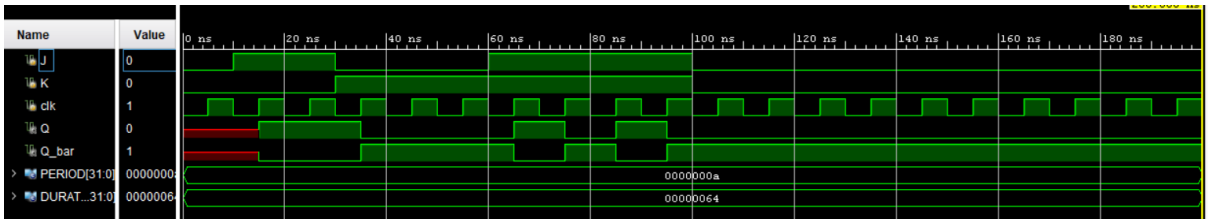
```
module four_bit_adder_tb;
reg [3:0]a,b;
reg cin;
wire [3:0]sum;
wire c4;

four_bit_adder uut(a,b,cin,sum,c4);

initial begin
cin = 0;
a = 4'b0110;
b = 4'b1100;
#10
a = 4'b1110;
b = 4'b1000;
#10
a = 4'b0111;
b = 4'b1110;
#10
a = 4'b0010;
b = 4'b1001;
#10
$finish();
end

endmodule
```



JK FLIFLOPS:

```
module jk_flipflop (
    input J,
    input K,
    input clk,
    output  Q,
    output  Q_bar
);

    // Internal state variable
    reg state;

    // Behavioral model for JK flip-flop
    always @(posedge clk) begin
        case ({J, K})
            2'b00: state <= state; // No change
            2'b01: state <= 1'b0;   // Reset
            2'b10: state <= 1'b1;   // Set
            2'b11: state <= ~state; // Toggle
        endcase
    end

    // Output assignments
    assign Q = state;
    assign Q_bar = ~state;

endmodule
```

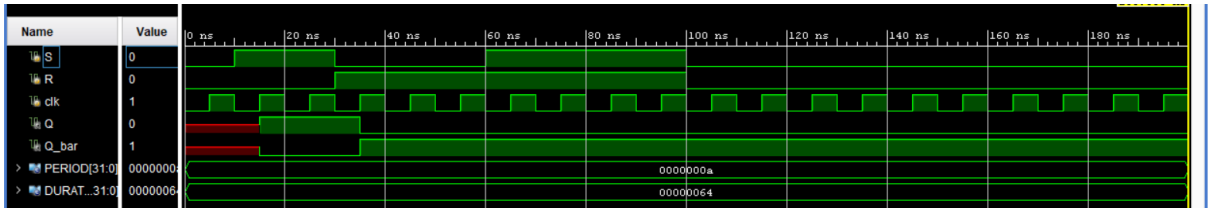
```
module jk_flipflop_tb;
    parameter PERIOD = 10; // Clock period
    parameter DURATION = 100; // Simulation duration
    reg J, K, clk;
    wire Q, Q_bar;
    jk_flipflop jk_ff (
        .J(J),
        .K(K),
        .clk(clk),
        .Q(Q),
        .Q_bar(Q_bar)
    );
    // Clock generation
    always #((PERIOD/2)) clk = ~clk;
    // Stimulus
    initial begin
        // Initial values
        J = 0; K = 0; clk = 0;

        // Apply stimulus
        #10 J = 1; K = 0; // Set
        #20 J = 0; K = 1; // Reset
        #30 J = 1; K = 1; // Toggle
        #40 J = 0; K = 0; // No change
        // Finish simulation
        #DURATION $finish;
    end
    // Display results
    initial begin
        $monitor("Time = %0t, J = %b, K = %b, Q = %b, Q_bar = %b", $time, J, K, Q, Q_bar);
    end

endmodule
```



SR FLIPFLOPS:

```verilog
module sr_flipflop (
    input S,
    input R,
    input clk,
    output  Q,
    output  Q_bar
);

    // Internal state variables
    reg state_Q, state_Q_bar;

    // Behavioral model for SR flip-flop
    always @(posedge clk) begin
        if (R && !S) begin
            state_Q <= 0;     // Reset
            state_Q_bar <= 1; // Q_bar is the complement of Q
        end else if (!R && S) begin
            state_Q <= 1;     // Set
            state_Q_bar <= 0; // Q_bar is the complement of Q
        end else if (!R && !S) begin
            state_Q <= state_Q;     // No change
            state_Q_bar <= state_Q_bar; // No change
        end else begin
            state_Q <= 0;     // Undefined state, arbitrarily set to reset
            state_Q_bar <= 1; // Undefined state, arbitrarily set to reset
        end
    end

    // Output assignments
    assign Q = state_Q;
    assign Q_bar = state_Q_bar;

endmodule
```

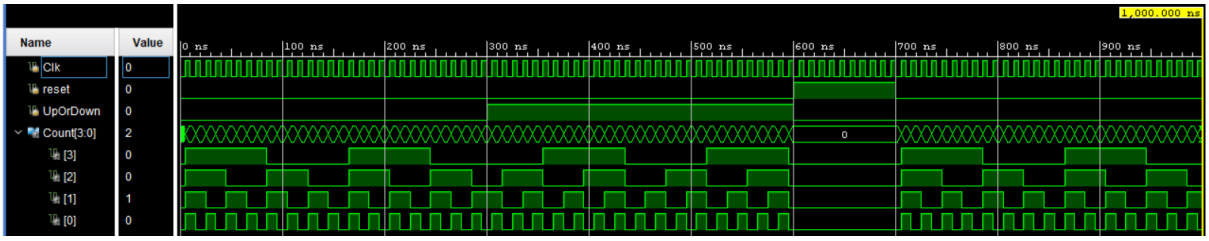
```
module sr_flipflop_tb;

    // Parameters
    parameter PERIOD = 10; // Clock period
    parameter DURATION = 100; // Simulation duration

    // Signals
    reg S, R, clk;
    wire Q, Q_bar;

    // Instantiate the SR flip-flop
    sr_flipflop sr_ff (
        .S(S),
        .R(R),
        .clk(clk),
        .Q(Q),
        .Q_bar(Q_bar)
    );

    // Clock generation
    always #((PERIOD/2)) clk = ~clk;

    // Stimulus
    initial begin
        // Initial values
        S = 0; R = 0; clk = 0;

        // Apply stimulus
        #10 S = 1; R = 0; // Set
        #20 S = 0; R = 1; // Reset
        #30 S = 1; R = 1; // Invalid input, arbitrary behavior
        #40 S = 0; R = 0; // No change

        // Finish simulation
        #DURATION $finish;
    end

    // Display results
    initial begin
        $monitor("Time = %0t, S = %b, R = %b, Q = %b, Q_bar = %b", $time, S, R, Q, Q_bar);
    end

endmodule
```



UP DOWN COUNTER:

```
module counter(
    input clk10,input rst,input ud,output [3:0]count );
    reg [3:0]count;
    reg clk2;
    reg [27:0]n,period;
    reg [26:0]duty;
    initial begin
    count=0;
    period<=28'd200000000;
    n<=28'd0;
    duty<=27'd100000000;
    end
    always @ (posedge clk10)
    begin
    n=n+1;
    if(n<=duty)
    clk2<=1;
    else
    clk2<=0;
    if(n==period)
    n<=0;
    end
    always @ (posedge(clk2)) begin
    if(rst)
    count<=0;
    else if (ud==0)
    count<=count+1;
    else
    count<=count-1;
    end

endmodule
```

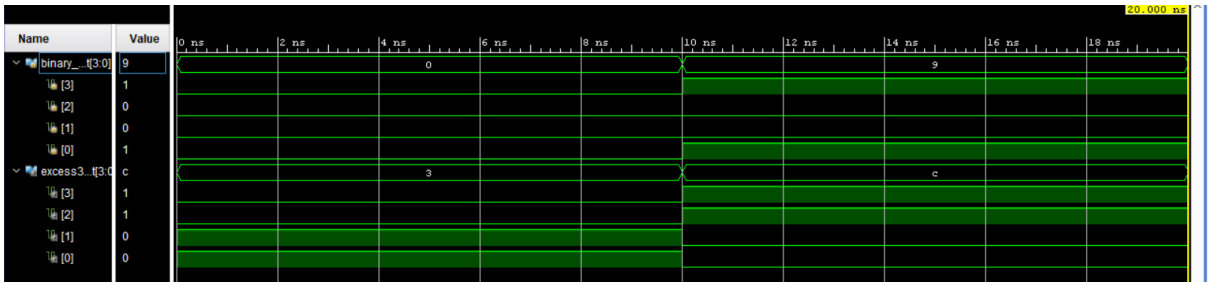
```
module tb_counter;
    // Inputs
    reg Clk;
    reg reset;
    reg UpOrDown;

    // Outputs
    wire [3:0] Count;

    // Instantiate the Unit Under Test (UUT)
    upordown_counter uut (
        .Clk(Clk),
        .reset(reset),
        .UpOrDown(UpOrDown),
        .Count(Count)
    );
//Generate clock with 10 ns clk period.
    initial Clk = 0;
    always #5 Clk = ~Clk;

    initial begin
        // Apply Inputs
        reset = 0;
        UpOrDown = 0;
        #300;
        UpOrDown = 1;
      #300;
        reset = 1;
        UpOrDown = 0;
        #100;
        reset = 0;
    end
endmodule
```



BINARY TO EXCESS 3:

```
module BinaryToExcess3 (
    input [3:0] binary_input,
    output [3:0] excess3_output
);

assign excess3_output = binary_input + 4'b0011;

endmodule
```

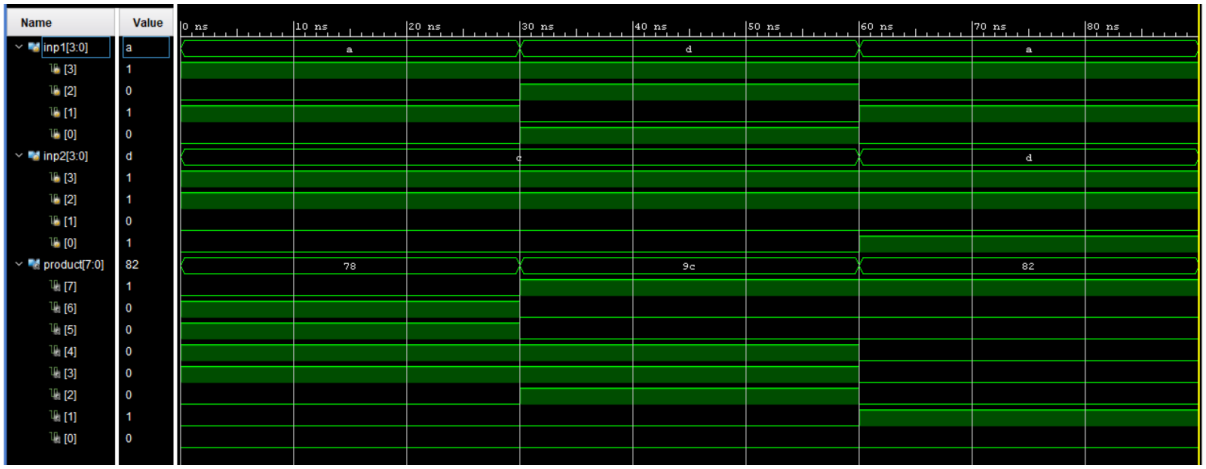
```
module BinaryToExcess3_tb;
reg [3:0] binary_input;
wire [3:0] excess3_output;
BinaryToExcess3 uut (
    .binary_input(binary_input),
    .excess3_output(excess3_output)
);
initial begin
    binary_input = 4'b0000;
    #10;
    binary_input = 4'b1001;
    #10;
    $finish;
end
endmodule
```



4X4 ARRAY MULTIPLIER:

```
`timescale 1ns / 1ps
module multiplier_4_x_4(product,inp1,inp2);
  output [7:0]product;
  input [3:0]inp1;
  input [3:0]inp2;
  assign product[0]=(inp1[0]&inp2[0]);
  wire x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17;
  HA HA1(product[1],x1,(inp1[1]&inp2[0]),(inp1[0]&inp2[1]));
  FA FA1(x2,x3,inp1[1]&inp2[1],(inp1[0]&inp2[2]),x1);
  FA FA2(x4,x5,(inp1[1]&inp2[2]),(inp1[0]&inp2[3]),x3);
  HA HA2(x6,x7,(inp1[1]&inp2[3]),x5);

  HA HA3(product[2],x15,x2,(inp1[2]&inp2[0]));
  FA FA5(x14,x16,x4,(inp1[2]&inp2[1]),x15);
  FA FA4(x13,x17,x6,(inp1[2]&inp2[2]),x16);
  FA FA3(x9,x8,x7,(inp1[2]&inp2[3]),x17);

  HA HA4(product[3],x12,x14,(inp1[3]&inp2[0]));
  FA FA8(product[4],x11,x13,(inp1[3]&inp2[1]),x12);
  FA FA7(product[5],x10,x9,(inp1[3]&inp2[2]),x11);
  FA FA6(product[6],product[7],x8,(inp1[3]&inp2[3]),x10);
endmodule

module HA(sout,cout,a,b);
  output sout,cout;
  input a,b;
  assign sout=a^b;
  assign cout=(a&b);
endmodule

module FA(sout,cout,a,b,cin);
  output sout,cout;
  input a,b,cin;
  assign sout=(a^b^cin);
  assign cout=((a&b)|(a&cin)|(b&cin));
endmodule
```

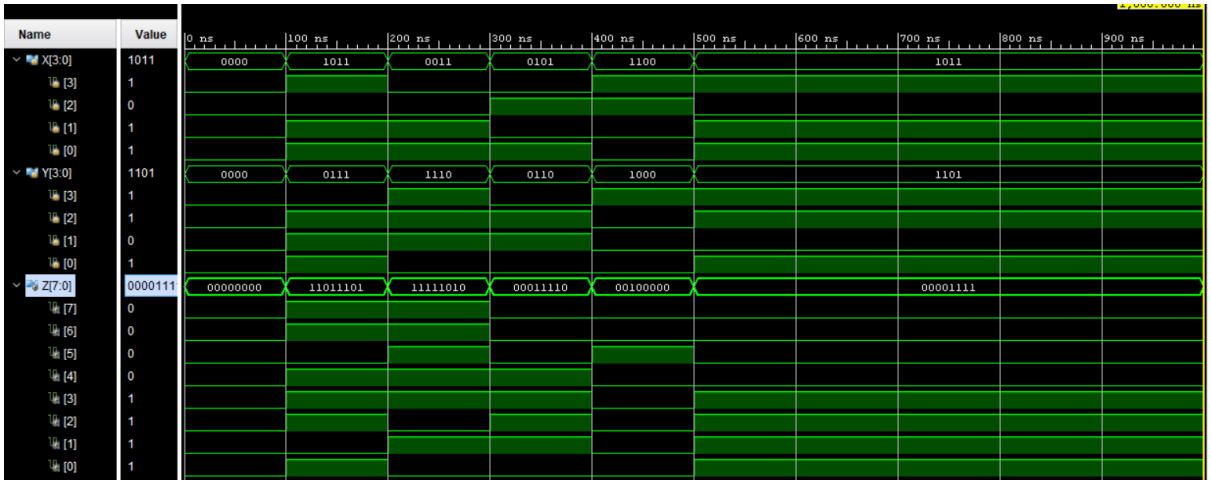
```
`timescale 1ns / 1ps
 module tb;

  reg [3:0]inp1;
  reg [3:0]inp2;
  wire [7:0]product;

  multiplier_4_x_4 uut(.inp1(inp1),.inp2(inp2),.product(product));

  initial
  begin
    inp1=10;
    inp2=12;
    #30 ;

    inp1=13;
    inp2=12;
    #30 ;

    inp1=10;
    inp2=13;
    #30 ;

    $finish;
  end
endmodule
```



BOOTH MULTIPLIER:

```
module booth_multi(X, Y, Z);
      input signed [3:0] X, Y;
       output signed [7:0] Z;
       reg signed [7:0] Z;
       reg [1:0] temp;
       integer i;
       reg E1;
       reg [3:0] Y1;
       always @ (X, Y)
       begin
       Z = 8'd0;
       E1 = 1'd0;
       for (i = 0; i < 4; i = i + 1)
       begin
       temp = {X[i], E1};
       Y1 = - Y;
  case (temp)
       2'd2 : Z [7 : 4] = Z [7 : 4] + Y1;
       2'd1 : Z [7 : 4] = Z [7 : 4] + Y;
       default : begin end
       endcase
       Z = Z >> 1;

       Z[7] = Z[6];


       E1 = X[i];
           end
       if (Y == 4'd8)


           begin
               Z = - Z;
           end

       end


endmodule
```

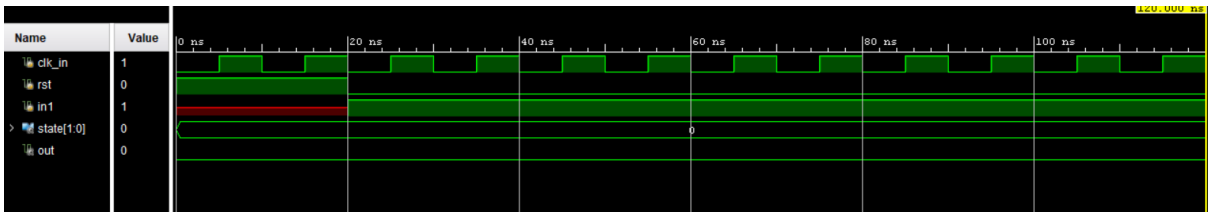
```
1  module booth_multi_tb;
2      reg [3:0] X;
3      reg [3:0] Y;
4      wire [7:0] Z;
5      booth_multi uut (
6          .X(X),
7          .Y(Y),
8          .Z(Z)
9      );
10
11     initial begin
12         // Initialize Inputs
13         X = 0;
14         Y = 0;
15
16         // Wait 100 ns for global reset to finish
17         #100; X=-5; Y=7;
18         #100; X=3; Y=-2;
19         #100; X=5; Y=6;
20         #100; X=-4; Y=8;
21         #100; X=11; Y=13;
22
23         end
24
25 endmodule
```



FINITE STATE MACHINE:

```
23  module fsm(clk_in, rst, in1,state,out);
24   input clk_in, rst, in1;
25   output reg out;
26   wire clk_out;
27   output reg [1:0] state;
28   frequency_div fd1(.clk_in(clk_in),.clk_out(clk_out));
29   parameter s1 = 2'b00;
30   parameter s2 = 2'b01;
31   parameter s3 = 2'b10;
32   parameter s4 = 2'b11;
33   always @(posedge clk_out or posedge rst)
34       begin
35        if(rst) begin
36           state <= s1;
37           out <= 1'b0;
38         end
39        else begin
40        case (state)
41         s1: begin
42             if(in1 == 1'b1) begin
43              state <= s2;
44              out <= 1'b0;
45              end
46             else begin
47                state <= s1;
48                out <= 1'b0;
49              end
50              end
51          s2:begin
52              if(in1 == 1'b1) begin
53                  state <= s3;
54                  out <= 1'b0;
55               end
56                 else begin
57                     state <= s1;
58                     out <= 1'b0;
59                 end
60                 end
61          s3: begin
62                if(in1 == 1'b1) begin
63                      state <= s4;
64                      out <= 1'b0;
```

```
65                 end
66                 else begin
67                        state <= s1;
68                        out <= 1'b0;
69                 end
70                 end
71           s4: begin
72               if(in1 == 1'b1) begin
73                    state <= s1;
74                    out <= 1'b1;
75                    end
76                 else begin
77                    state <= s1;
78                    out <= 1'b0;
79                  end
80                  end
81         endcase
82       end
83      end
84  endmodule
85
86  module frequency_div(input clk_in,input rst,input ud, output [3:0]count, output reg clk_out);
87      reg [3:0]count;
88      reg clk2;
89      reg [27:0]n,period;
90      reg [26:0]duty;
91      initial begin
92      count=0;
93      period<=28'd200000000;
94      n<=28'd0;
95      duty<=27'd100000000;
96      end
97      always @ (posedge clk_in)
98      begin
99      n=n+1;
100     if(n<=duty)
101     clk2<=1;
102     else
103     clk2<=0;
104     if(n==period)
105     n<=0;
106     end
107 endmodule
```



```
module fsm_tb();
reg clk_in,rst,inl;
wire [1:0]state;
wire out;

fsm dut(.clk_in(clk_in),.rst(rst),.inl(inl),.state(state),.out(out));

initial begin clk_in=1'b0;
rst=1'b1;
end

always #5 clk_in = ~clk_in;
initial begin
    #20
   inl=1'b1;
   rst=1'b0;
   #20
   inl=1'b1;
   rst=1'b0;
   #20
   inl=1'b1;
   rst=1'b0;
   #20
   inl=1'b1;
   rst=1'b0;
   #20
   inl=1'b1;
   rst=1'b0;

end
initial #120 $finish;
endmodule
```