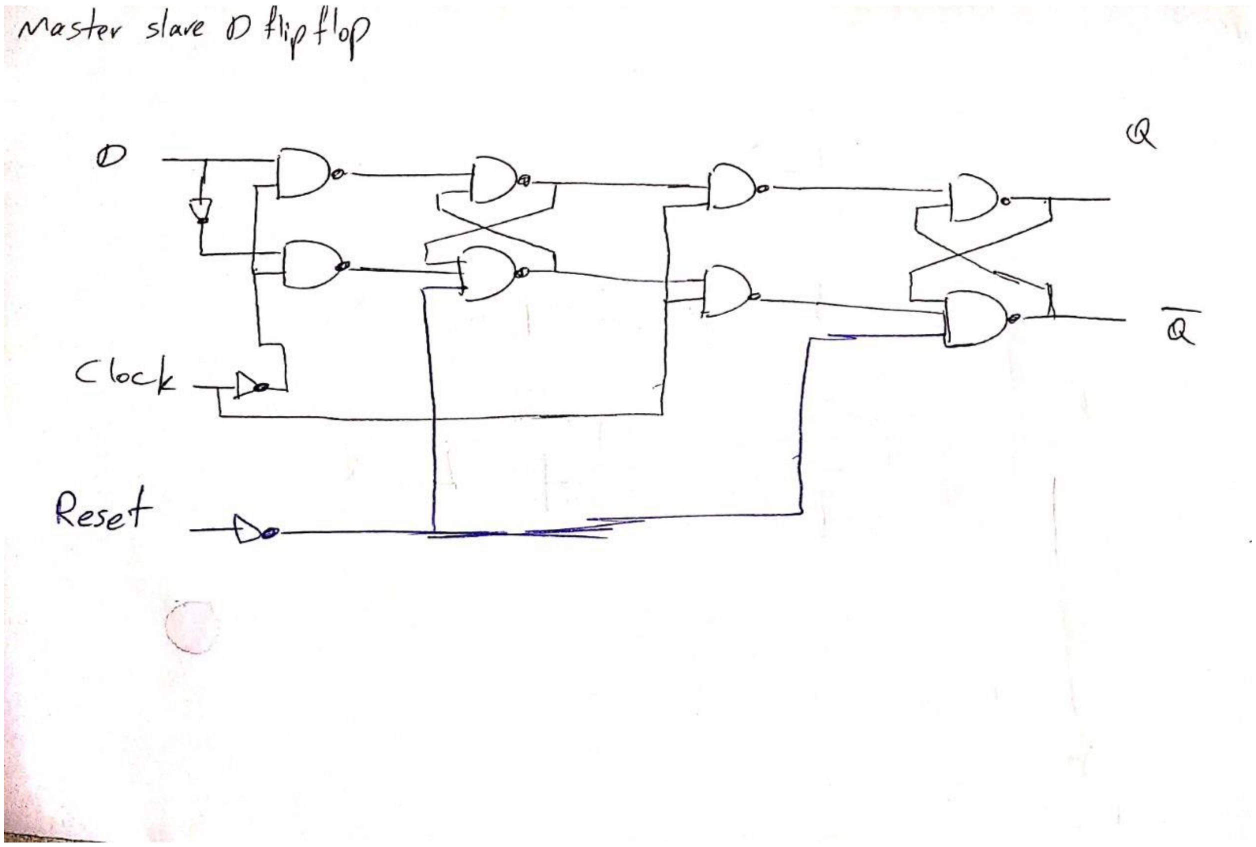
۵ روز گریس

محمد حسین جوادی

۸۱۰۱۰۳۵۶۷

Master slave d flip flop

Schematic:



Code:

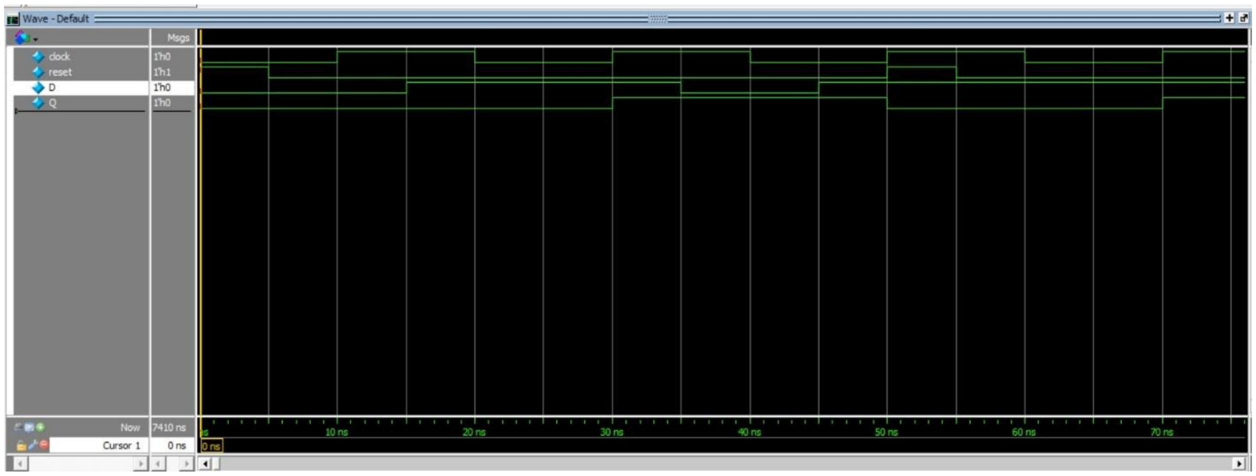
```
`timescale 1ns/1ns

module msdff (
    input D,Reset,clock,output  Q
);

    wire D_bar;
    wire C_bar;
    wire N_r;
    wire upper1;
    wire upper2;
    wire upper3;
    wire upper4;
    wire down1;
    wire down2;
    wire down3;
    wire Q_bar;
```

```verilog

    not d_comp(D_bar,D);
    not C_comp(C_bar,clock);
    not R_comp(N_r,Reset);
    //master
    nand first_upper(upper1,D,C_bar);
    nand first_down(down1,D_bar,C_bar);
    nand second_upper(upper2,upper1,down2);
    nand second_down(down2,down1,upper2,N_r);

    //slave
    nand s_first_upper(upper3,upper2,clock);
    nand s_first_down(down3,down2,clock);
    nand s_second_upper(Q,upper3,Q_bar);
    nand s_second_down(Q_bar,Q,down3,N_r);

endmodule

```

Testbench:

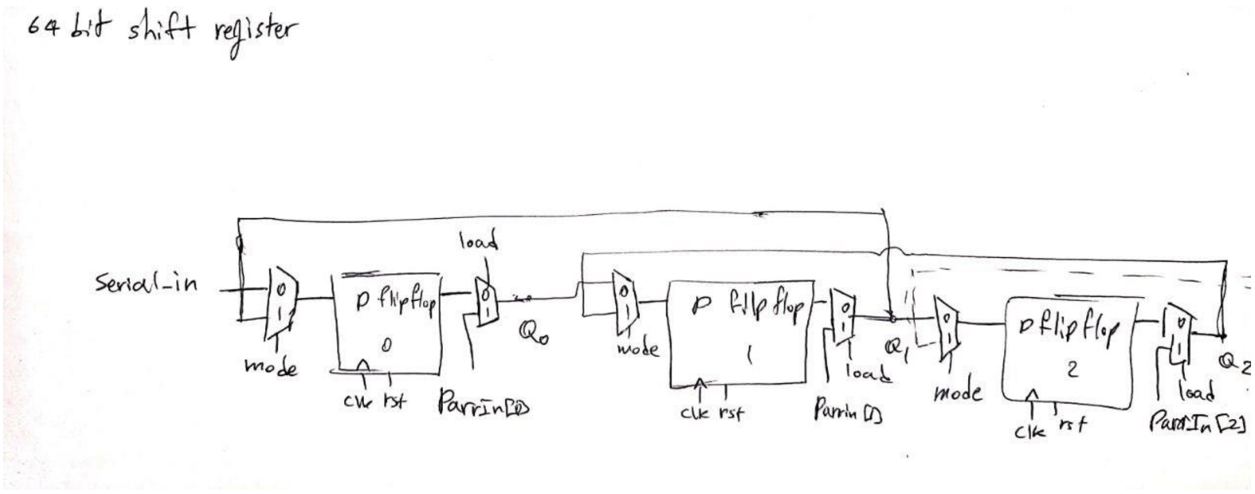
```verilog
`timescale 1ns/1ns

module dfftb ();
reg D;
reg clock;
reg reset;
wire Q;

msdff name(D,reset,clock,Q);

initial begin
    clock = 0;
    forever #10 clock = ~clock;
end

initial begin
    D = 0;
    reset = 1;

    #5
    reset = 0;

    #10
    D = 1;
    #10
    D = 1;
    #10
    D = 0;
    #10
    D = 1;
    #5
    reset = 1;

    #5
    reset = 0;
end
endmodule

```

Simulation:



64 bit shift register:

Schematic:



Code:

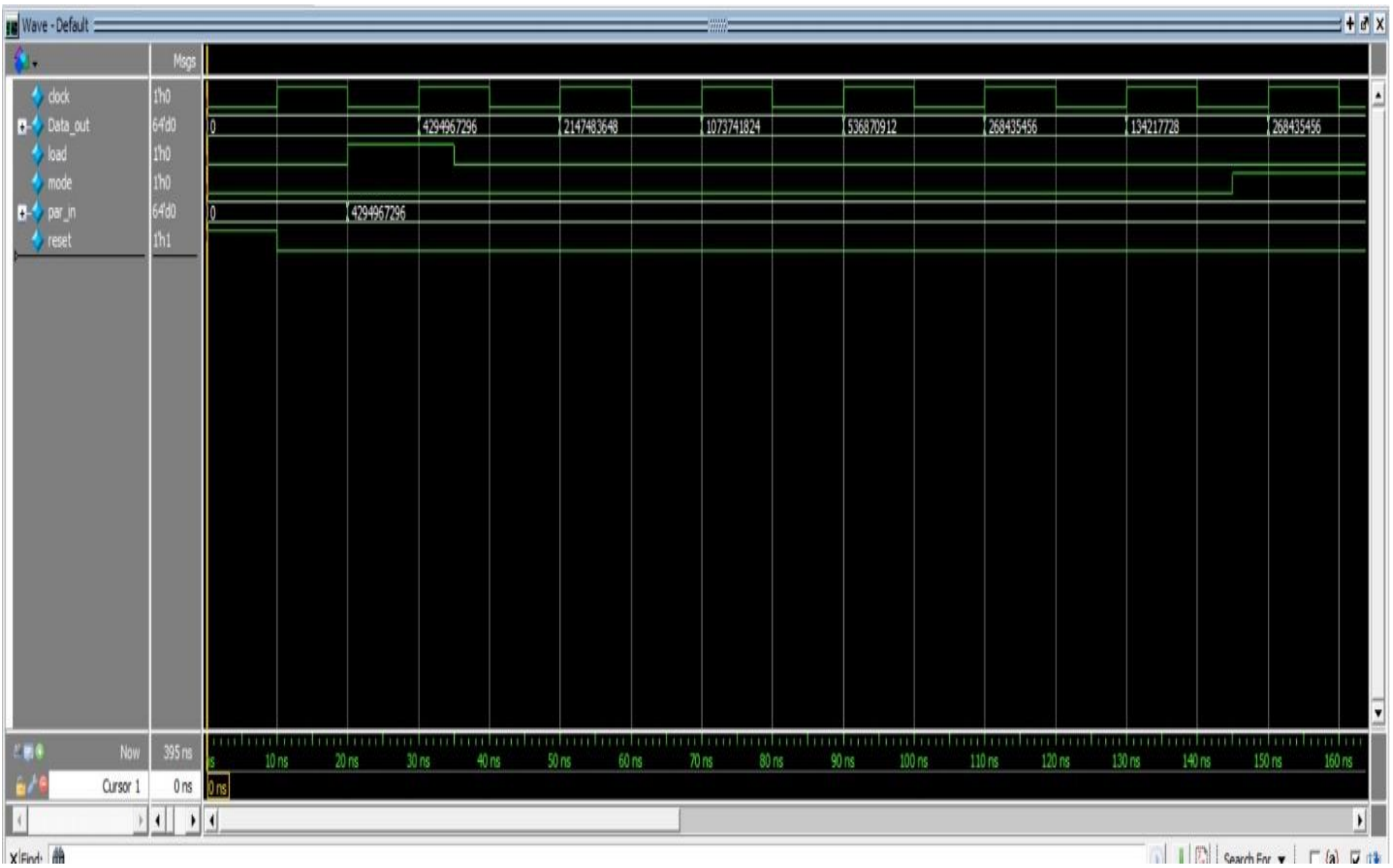
```
`timescale 1ns/1ns

module sr64(par_in,Data_out,mode,load,serin,clock,reset);
parameter n = 64;
input [n-1:0]par_in;
input mode;
input load;
input serin;
input clock;
input reset;
output reg [n-1:0]Data_out;
integer k;

always @(posedge clock or posedge reset) begin
    if (reset)
```

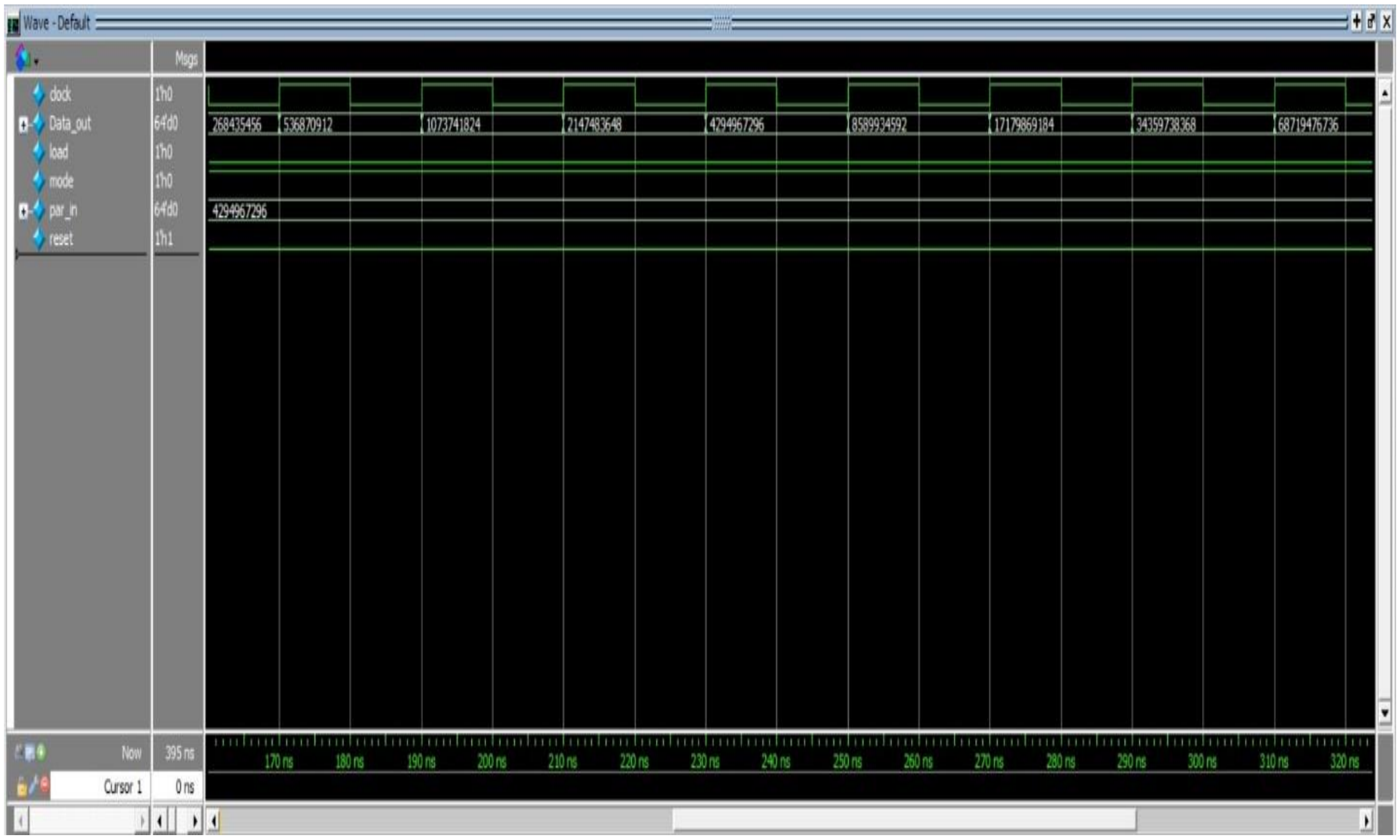
```
        Data_out <= {n{1'b0}};
    else if(load)
        Data_out <= par_in;
    else
        begin
            if(!mode) begin
                for (k = 0;k < n-1 ;k = k + 1 ) begin
                    Data_out[k] <= Data_out[k + 1];
                end
                Data_out[n - 1] <= serin;
            end
            else begin
                for (k = n - 1;k > 0 ;k = k - 1 ) begin
                    Data_out[k] <= Data_out[k - 1];
                end
                Data_out[0] <= serin;
        end
        end
end
endmodule

```

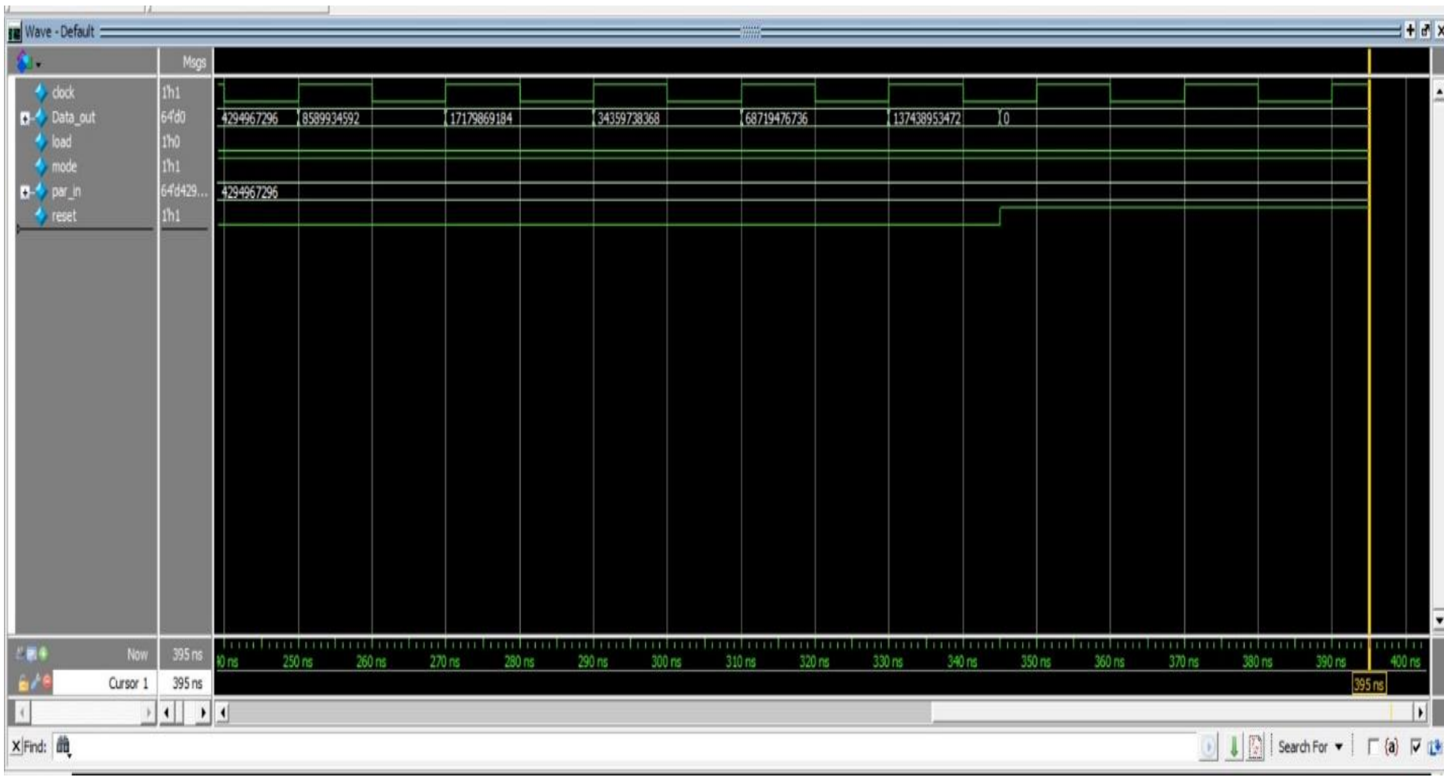
Testbench:

```
`timescale 1ns/1ns

module sr64tb();
    parameter n = 64;
    reg [n-1:0] par_in;
    reg mode;
    reg load;
    reg serin;
    reg clock;
    reg reset;

    wire [n-1:0]Data_out;

    sr64 name(par_in,Data_out,mode,load,serin,clock,reset);

    initial begin
        clock = 0;
        forever #10 clock = ~clock;
    end

    initial begin
        par_in = 64'd0;
        mode = 0;
        load = 0;
        serin = 0;
        reset = 1;

        #10
        reset = 0;

        #10
        par_in = 64'd4294967296;
        load = 1;

        #15
        load = 0;

        #10
        mode = 0;
        serin = 0;

        #100
        mode = 1;
        serin = 0;

        #200
        reset = 1;

        #50
        $stop;
    end
endmodule

```

Simulation:



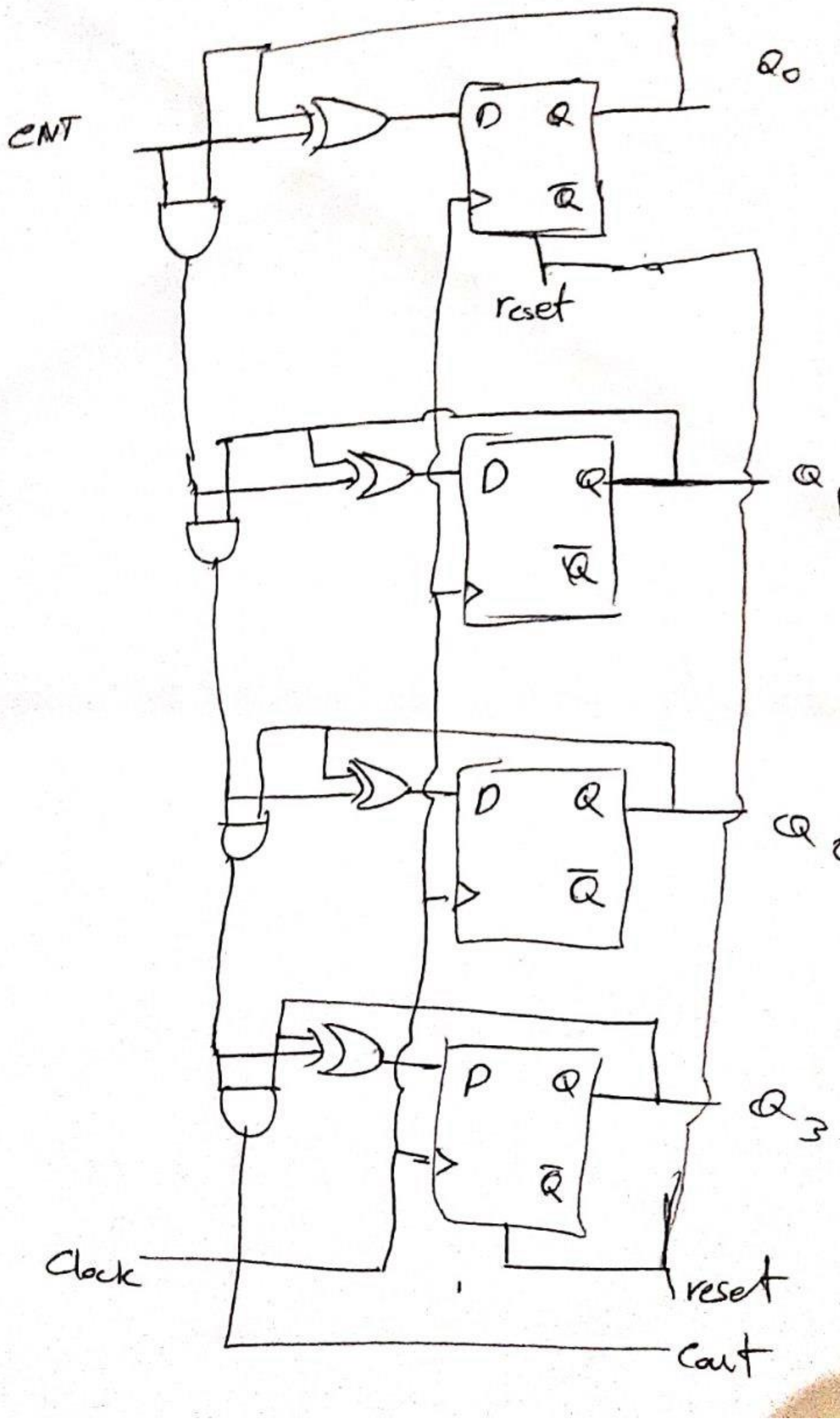




4 bit up counter

Shematic:

4 bit counter:



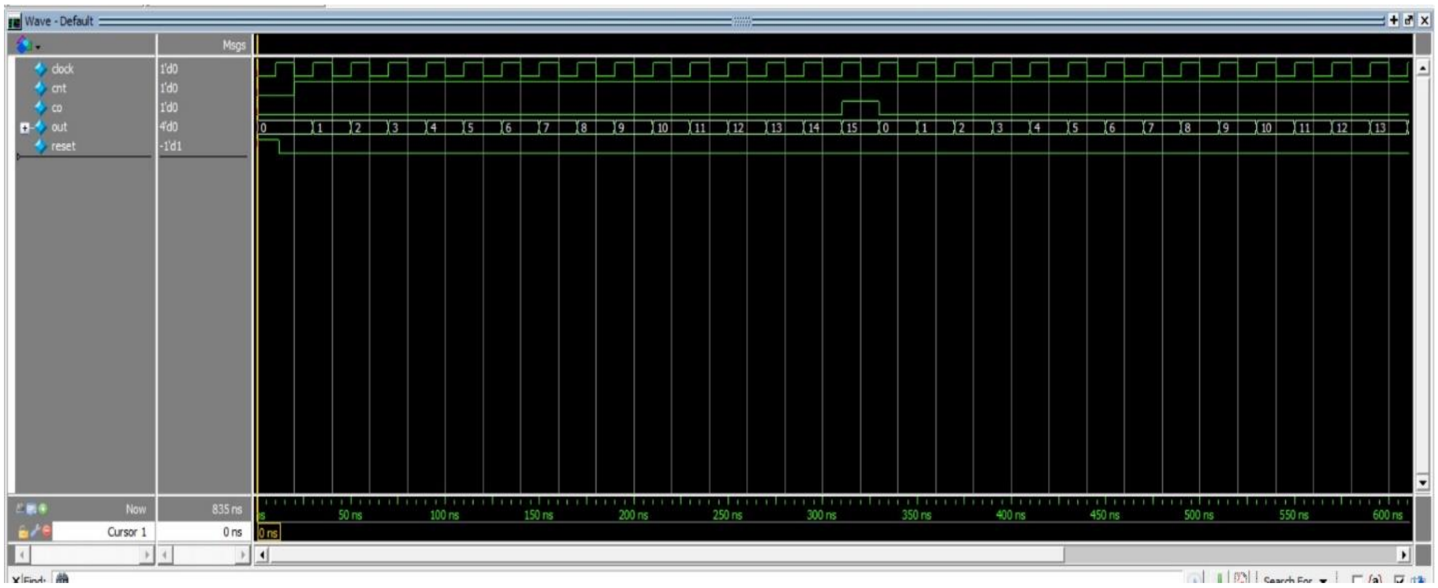
Code:

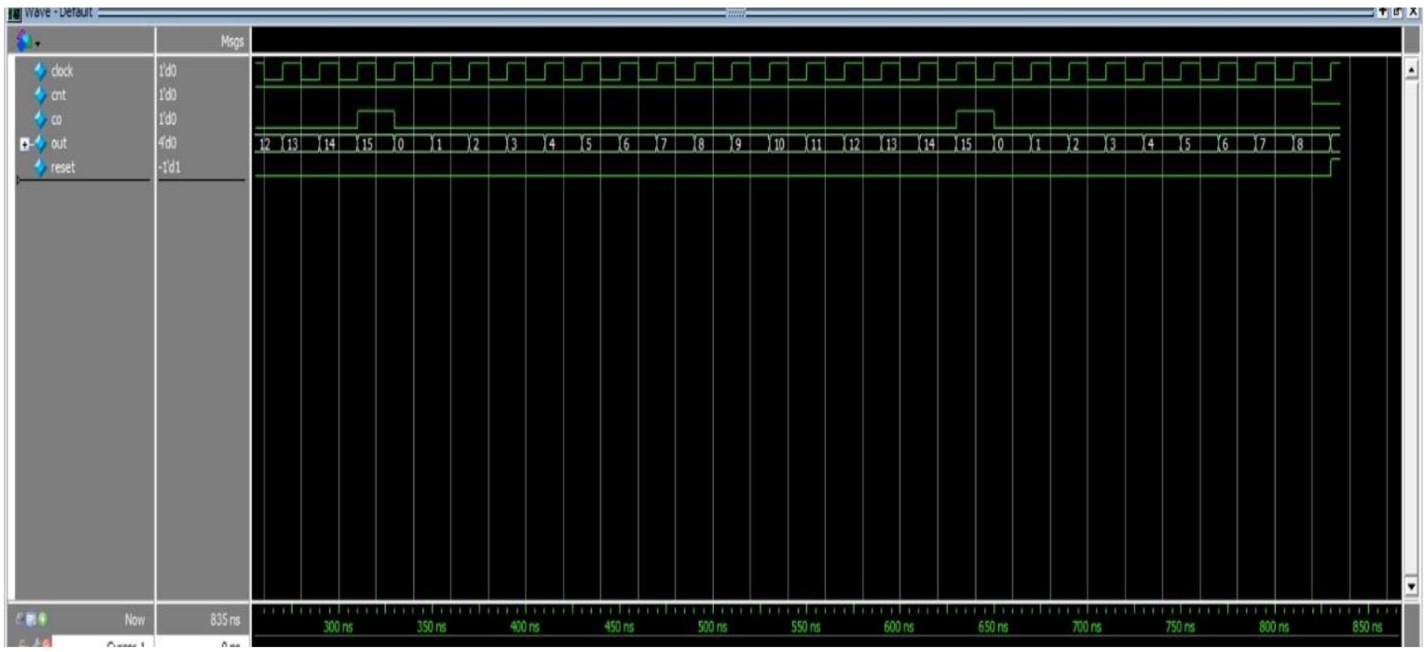
```
`timescale 1ns/1ns

module counter #(parameter n = 4)(input clock,input reset,input cnt,output co,output [n-1:0]out);
    wire [n-1:0]D;
    wire[n-1:0]toggle;
    genvar  k;
    generate
    for (k = 0;k < n;k = k + 1)begin
        if (k == 0)
            assign toggle[k] = cnt;
        else
            assign toggle[k] = cnt & &out[k-1:0];

        assign D[k] = out[k] ^ toggle[k];

        msdff ins(D[k],reset,clock,out[k]);
    end
    endgenerate
    assign co = cnt & &out;
endmodule

```

Testbench

```
`timescale 1ns/1ns

module countertb ();

    parameter n = 4;
    reg cnt;
    wire co;
    reg clock;
    reg reset;
    wire [n-1:0]out;

    counter #(n) name(clock,reset,cnt,co,out);

    initial begin
        clock = 0;
        forever #10 clock = ~clock;
    end

    initial begin
        reset = 1;
        cnt = 0;

        #12
        reset = 0;

        #8
        cnt = 1;

        #800
        cnt = 0;

        #10
        reset = 1;

        #5
        $stop;
    end

endmodule


```

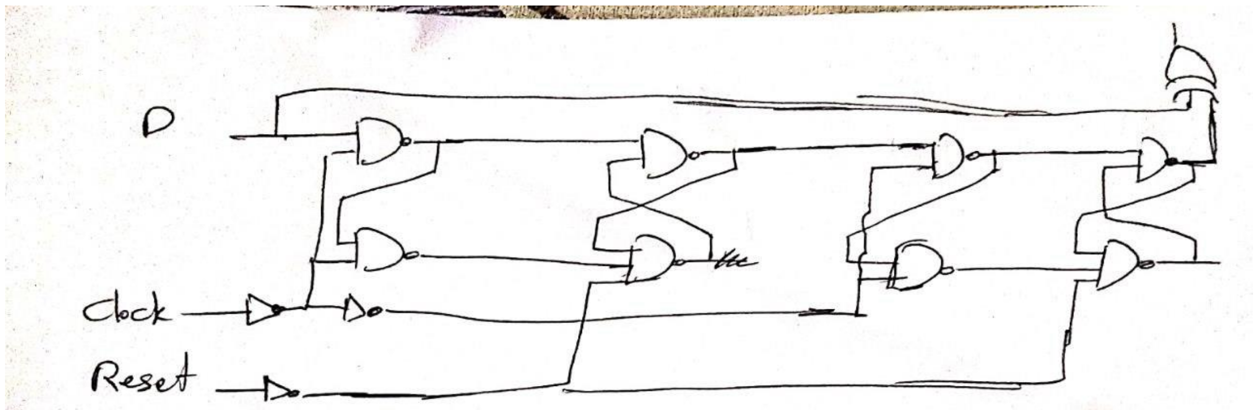
Simulation:





Toggle detector:

Schematic:



تفاوت اصلی latch با flipflop، در level sens بودن آن است که همین باعث می‌شود که در latch بلافاصله با تغییر input (در high level) ورودی و خروجی برابر شوند ولی flipflop تا risingedge مقدار را نگه می‌دارد و امکان مقایسه با XOR را می‌دهد.

Code:

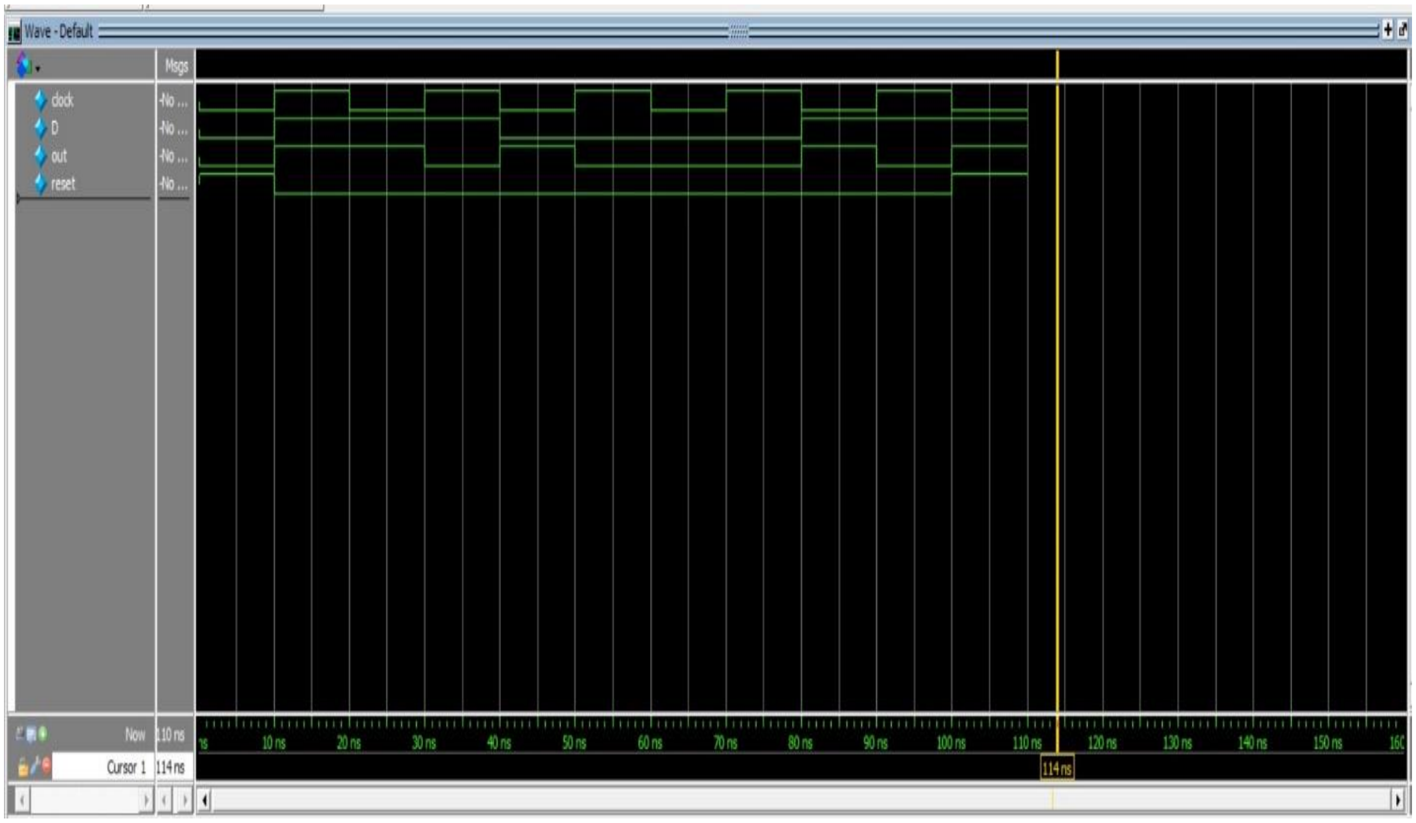
```
`timescale 1ns/1ns

module td (input clock,input D,input reset,output Q);
    wire temp;
    msdff new(D,reset,clock,temp);
    assign Q = D ^ temp;

endmodule

```

Testbench:

```
`timescale 1ns/1ns
module toggletb ();

    reg reset;
    reg D;
    reg clock;
    wire out;

    td name(clock,D,reset,out);

    initial begin
```

```
        clock = 0;
        forever #10 clock = ~clock;
    end

    initial begin
        D = 0;
        reset = 1;

        #10
        reset = 0;
        D = 1;

        #30
        D = 0;

        #40
        D = 1;

        #20
        reset = 1;

        #10
        $stop;
    end

endmodule

```

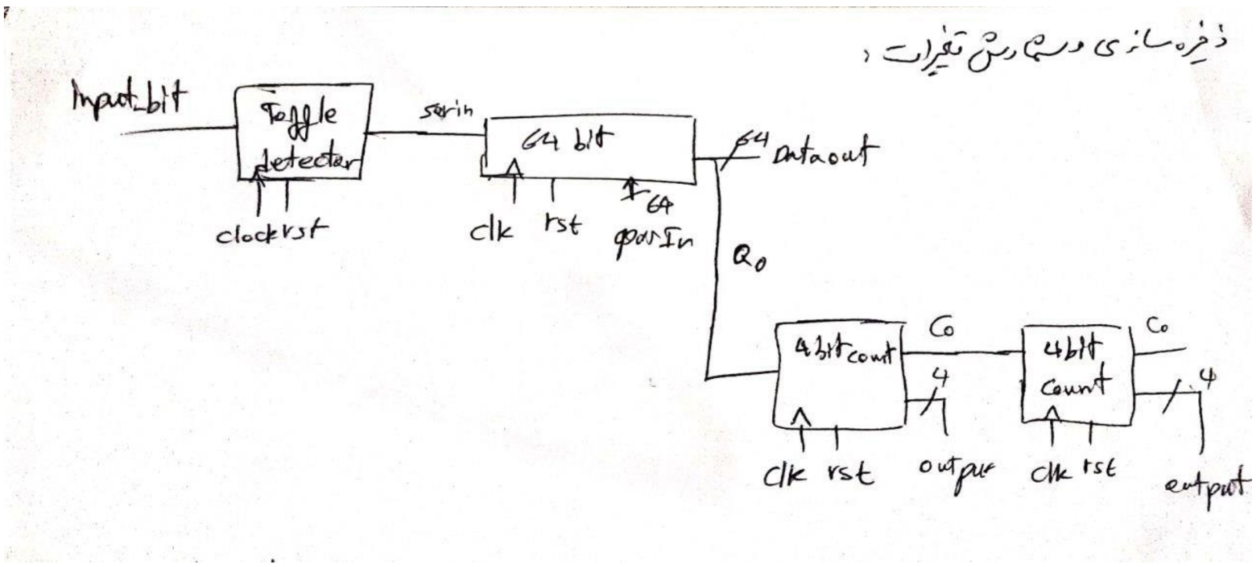
Simulation:



Toggle counter

حداکثر ۲۵۵ تغییر را میتواند بشمارد.

Schematic:



Code:

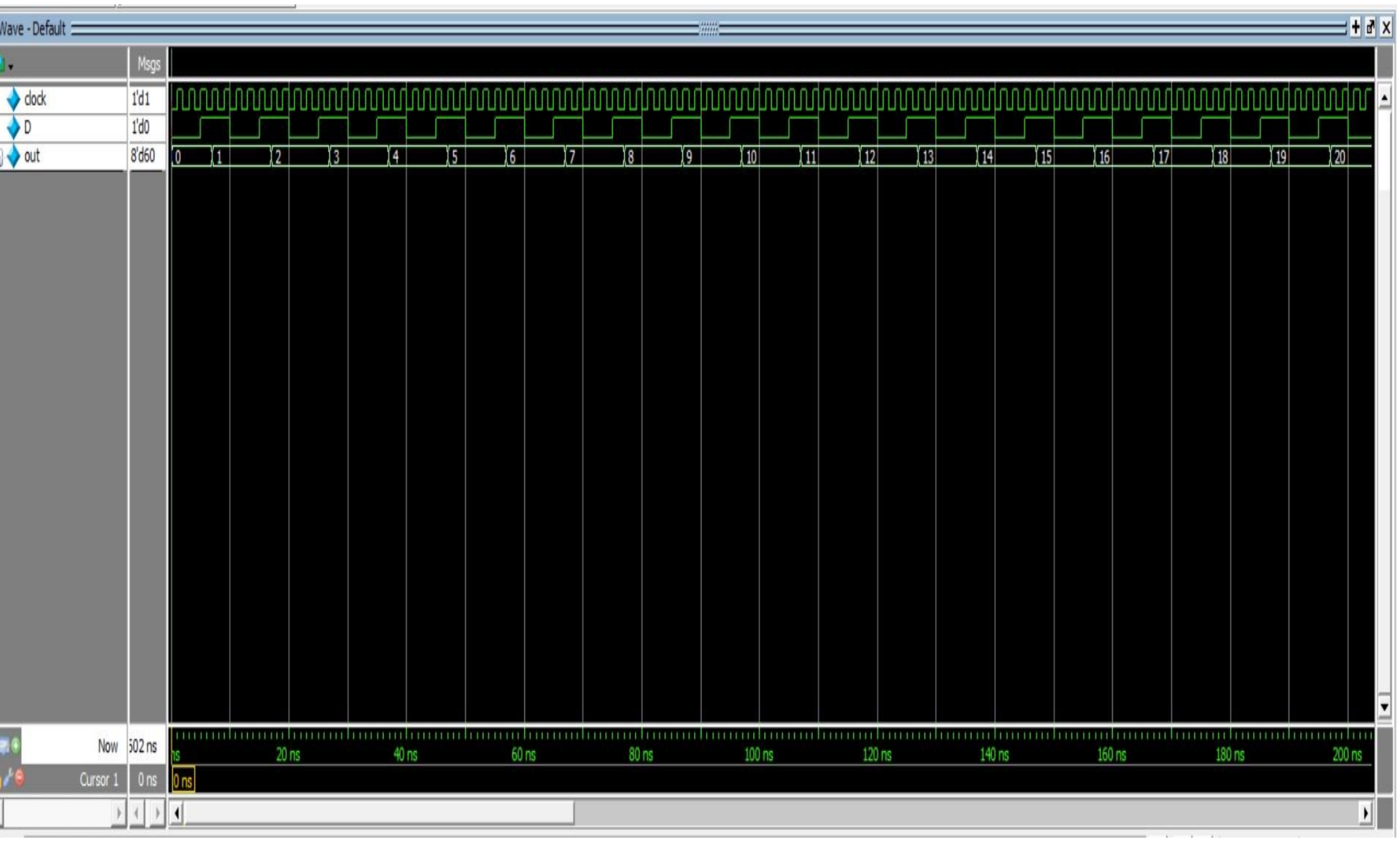
```
`timescale 1ns/1ns

module Toggle_Counting #(parameter n = 64,parameter h = 8)(par_in,clock,reset,mode,load,D,out);
    input clock;
    input reset;
    input load;
    input mode;
    input [n-1:0]par_in;
    input D;
    wire Q;
    wire [n-1:0]Data_out;
    wire co;
    output [h-1:0]out;

    td toggle(~clock,D,reset,Q);
    sr64 shift(par_in,Data_out,mode,load,Q,clock,reset);
    counter #(h) count(clock,reset,Data_out[n-1],co,out);

endmodule

```

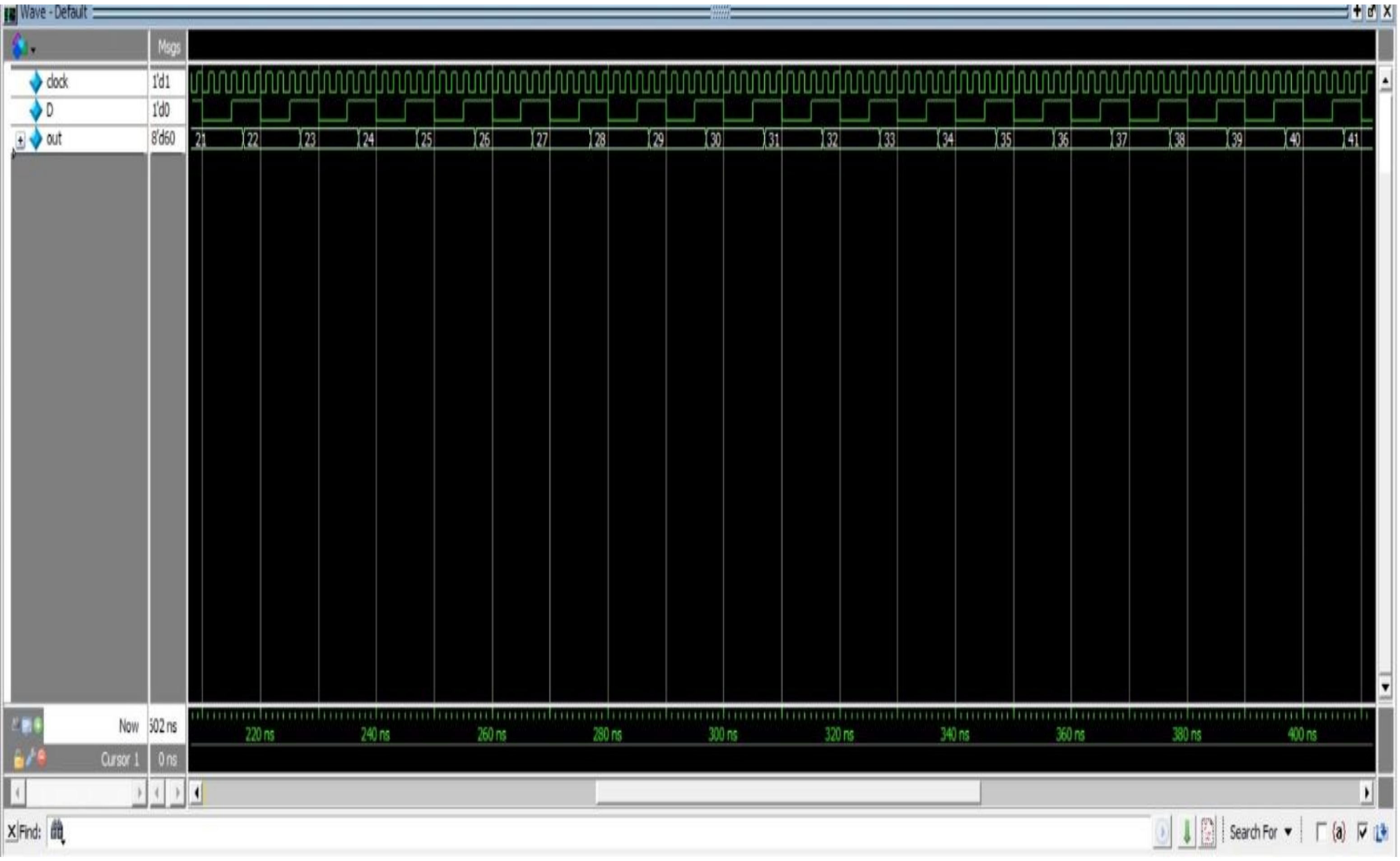
Testbench:

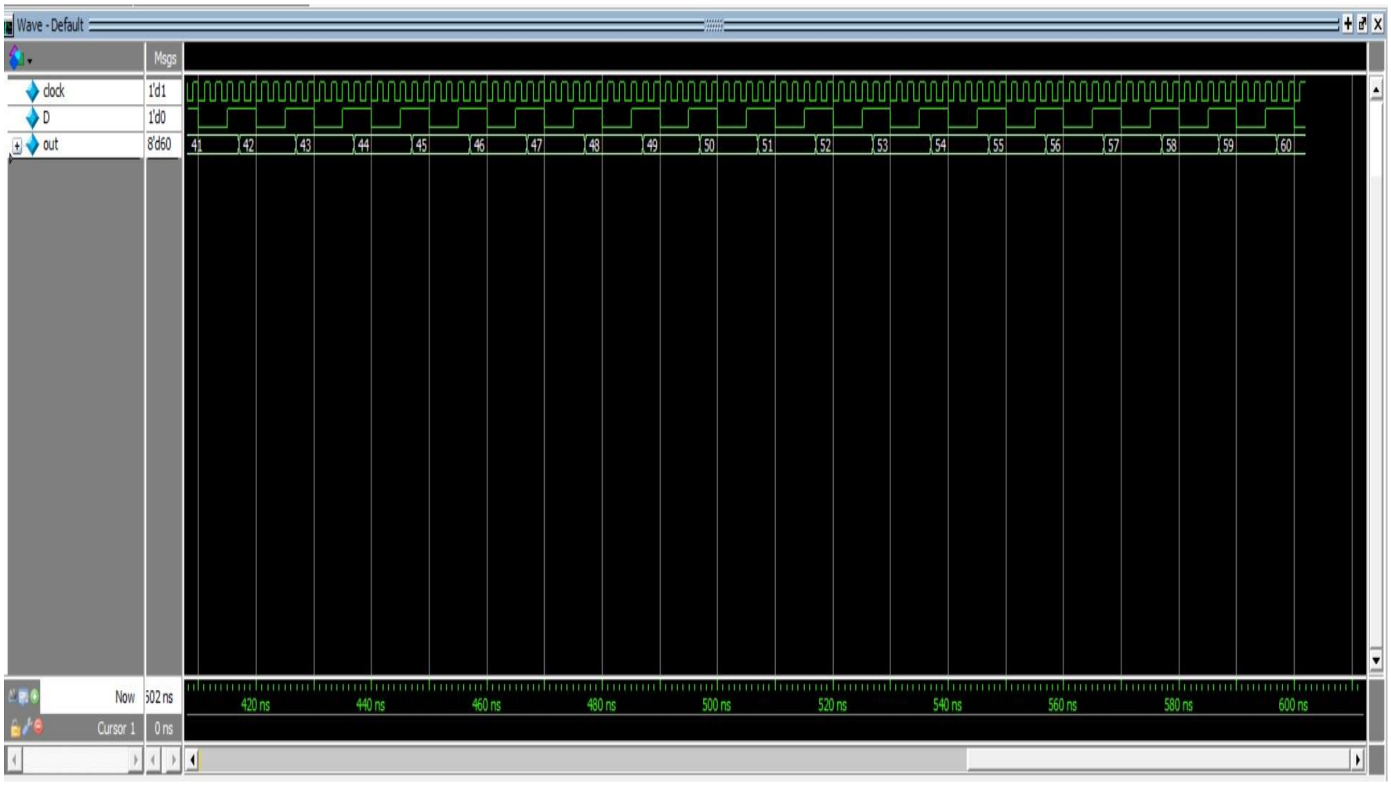
```
`timescale 1ns/1ns

```

```
module tctb ();
    parameter n = 64;
    parameter h = 8;
    reg clock;
    reg reset;
    reg load;
    reg mode;
    reg [n-1:0]par_in;
    reg D;
    wire [h-1:0]out;

    Toggle_Counting #(n,h)name  (par_in,clock,reset,mode,load,D,out);

    initial begin
        clock = 0;
        forever #1 clock = ~clock;
    end

    initial begin
        D = 0;
        forever #5 D = ~D;
    end

    initial begin
        mode = 0;
        reset = 1;
        load = 0;
        par_in = 64'd0;

        #2
        reset = 0;

        #600
        $stop;

    end

endmodule

```

Simulations:


Wave - Default
Msgs
clock 1'd1
D 1'd0
out 8'd60
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Now 502 ns
Cursor 1 0 ns
ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns
0 ns

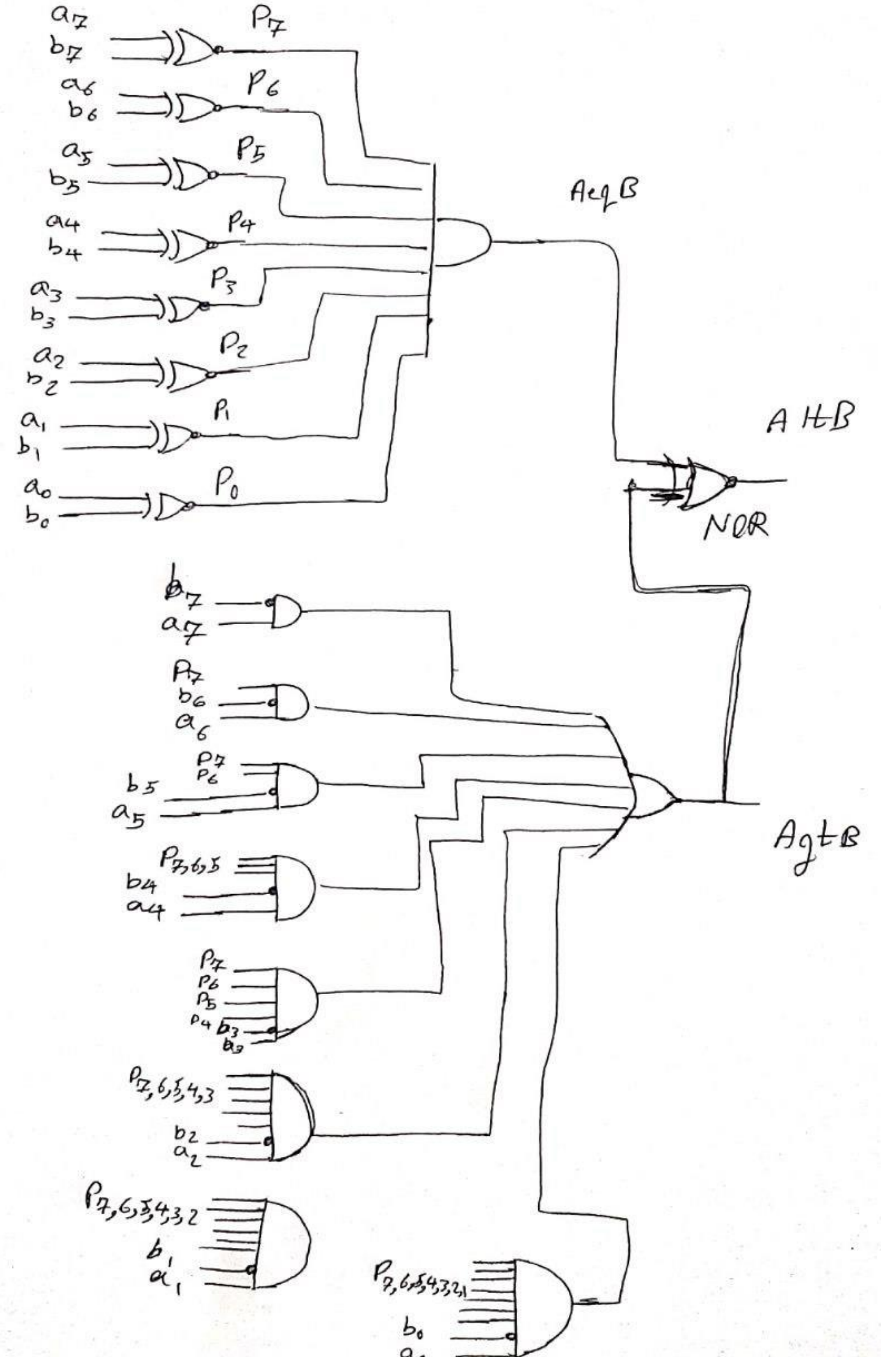

Wave - Default
Msgs
clock 1'd1
D 1'd0
out 8'd60
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
Now 502 ns
Cursor 1 0 ns
220 ns 240 ns 260 ns 280 ns 300 ns 320 ns 340 ns 360 ns 380 ns 400 ns
Find:
Search For



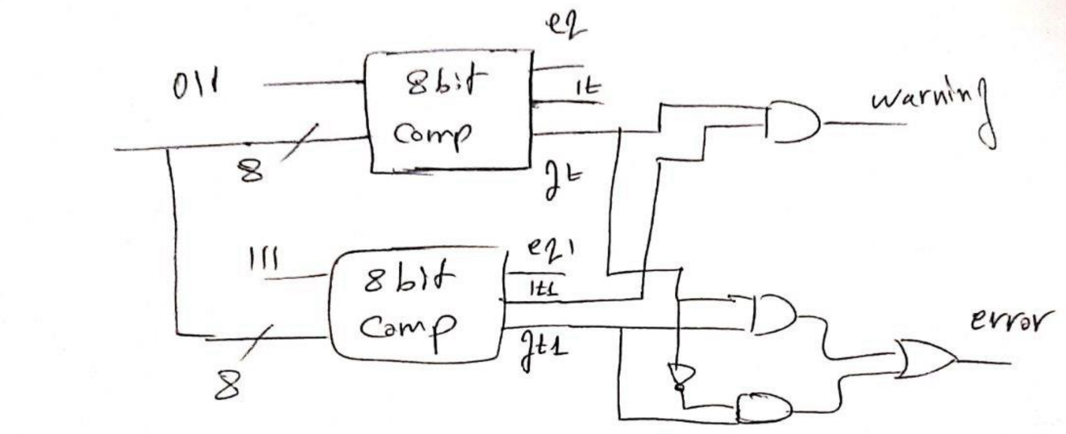
Comparator:

Schematic:

8 bit comparator



Comparator:



Code:

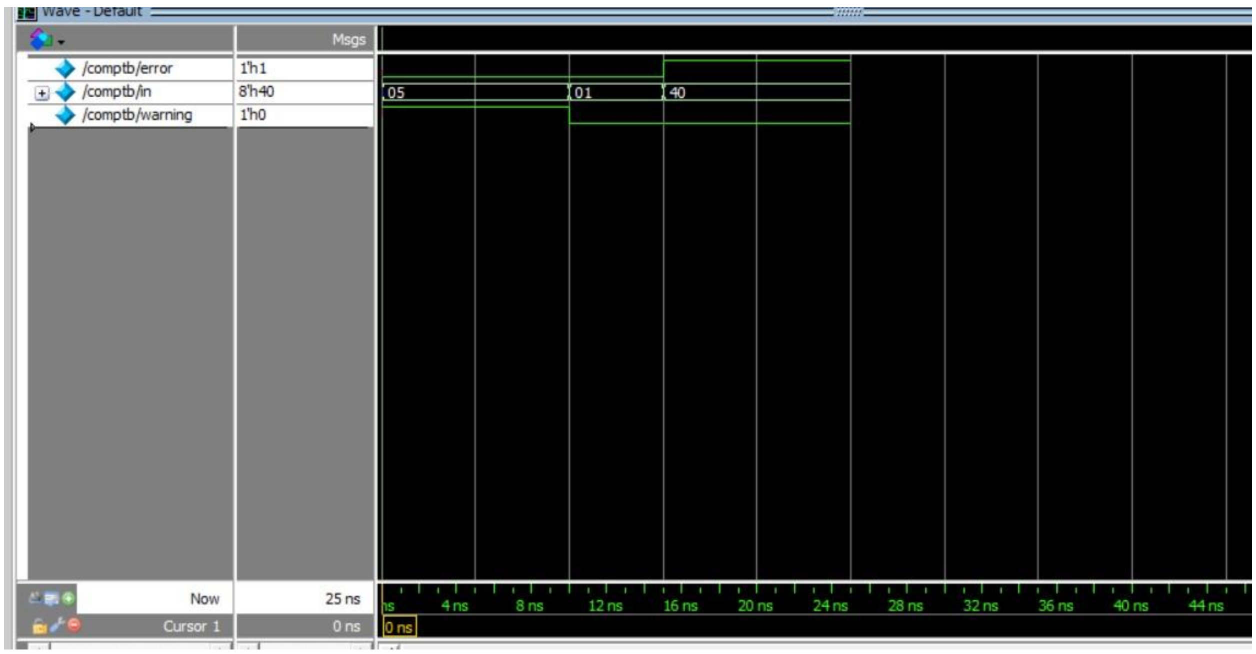
```
`timescale 1ns/1ns

module comparator8 #(parameter n = 8)(input [n-1:0]main_number,input [n-1:0]given_number,output
eq,output gt,output lt);
    wire [n-1:0]p;
    wire [n-1:0]q;

    genvar k;
    genvar i;
    generate
        for (k = 0;k < n;k = k + 1)begin
            assign p[k] = ~(main_number[k] ^ given_number[k]);
        end
        for (i = 0;i < n ;i = i+1)
        begin
            if (i == n-1)
                assign q[i] = main_number[i] & ~given_number[i];
            else
                assign q[i] = main_number[i] & ~given_number[i] & &p[n-1:i+1];
        end
    endgenerate

    assign eq = &p;
    assign gt = |q;
    assign lt = ~(eq | gt);
endmodule

```

```
`timescale 1ns/1ns

module comparator#(parameter n = 8)(input [n-1:0]in,output warning,output error);
wire eq;
wire lt;
wire gt;
wire eq1;
wire lt1;
wire gt1;

comparator8 #(n) first(in,8'b00000011,eq,gt,lt);
comparator8 #(n) second(in,8'b00000111,eq1,gt1,lt1);

assign warning = gt & lt1;

assign error = (gt1 & gt) | (gt1 & ~gt);

endmodule

```

Testbench:

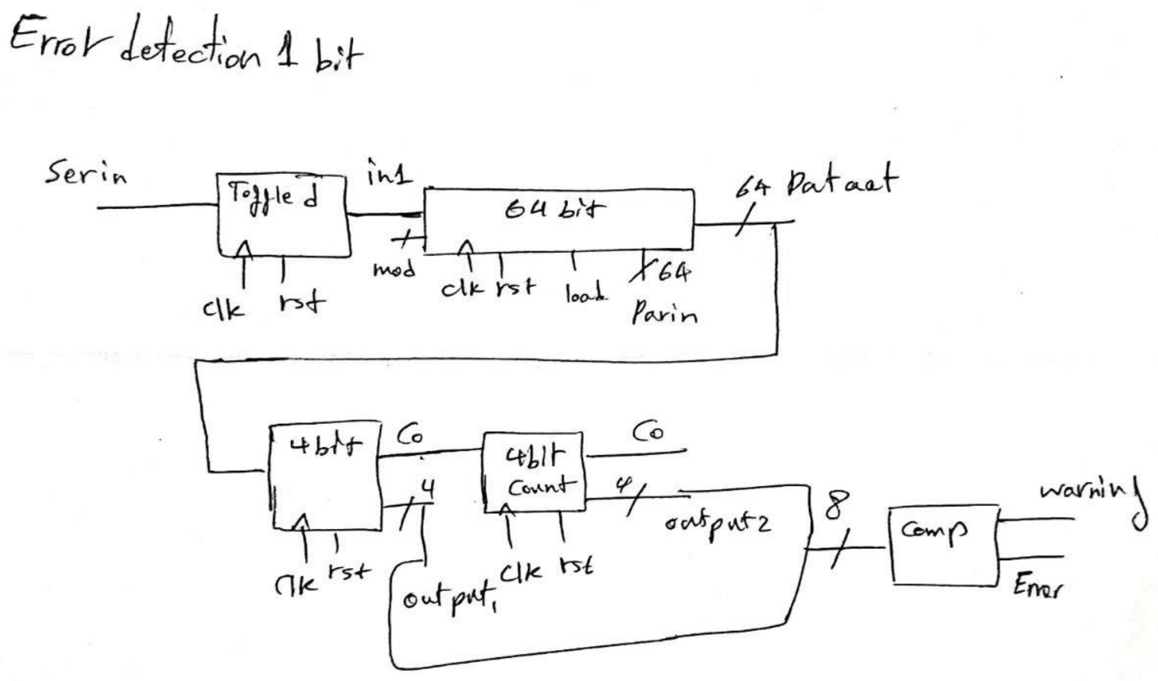
```
`timescale 1ns/1ns

module comptb ();
parameter n = 8;
reg [n-1:0]in;
wire warning;
wire error;

comparator #(n) inst(in,warning,error);
initial begin
    in = 8'd5;

    #10
    in = 8'd1;

    #5
    in = 8'd64;

    #10
    $stop;
end

endmodule

```

Simulations:



Error Detector:

Shematic:



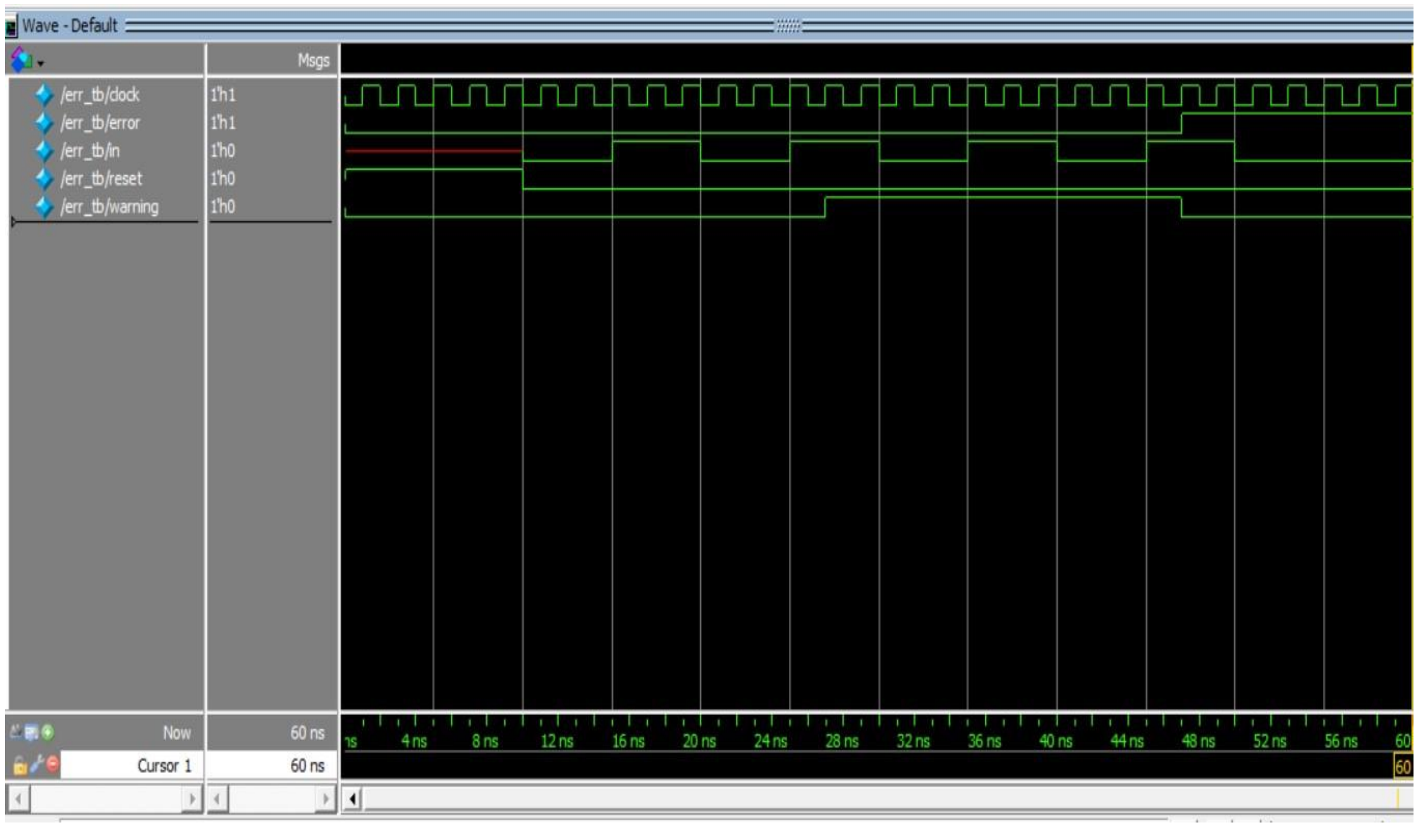
Code:

```
`timescale 1ns/1ns

module error_d #(parameter n = 64,parameter h = 8) (input in,input clock,input reset,output warning,output error );
wire mode;
wire [n-1:0] par_in;
wire load;

assign mode = 0;
assign par_in = {n{1'b0}};
assign load = 0;

wire [h-1:0] tempout;

Toggle_Countingd #(n,h) inst(par_in,clock,reset,mode,load,in,tempout);
comparator #(h) temp(tempout,warning,error);


endmodule

```

Testbench:

```verilog
`timescale 1ns/1ns

module err_tb ();
parameter n =64;
parameter h = 8;

reg in;
reg clock;
reg reset;

wire warning;
wire error;

error_d #(n,h) name(in,clock,reset,warning,error);

initial begin
    clock = 0;
    forever #1 clock = ~clock;
end

initial begin
    reset = 1;

    #10
    reset = 0;
    in = 0;

    repeat (8) begin
        #5 in = ~in;
    end

    #10
    $stop;
end

endmodule

```

Simulations:



8 bit error detector:

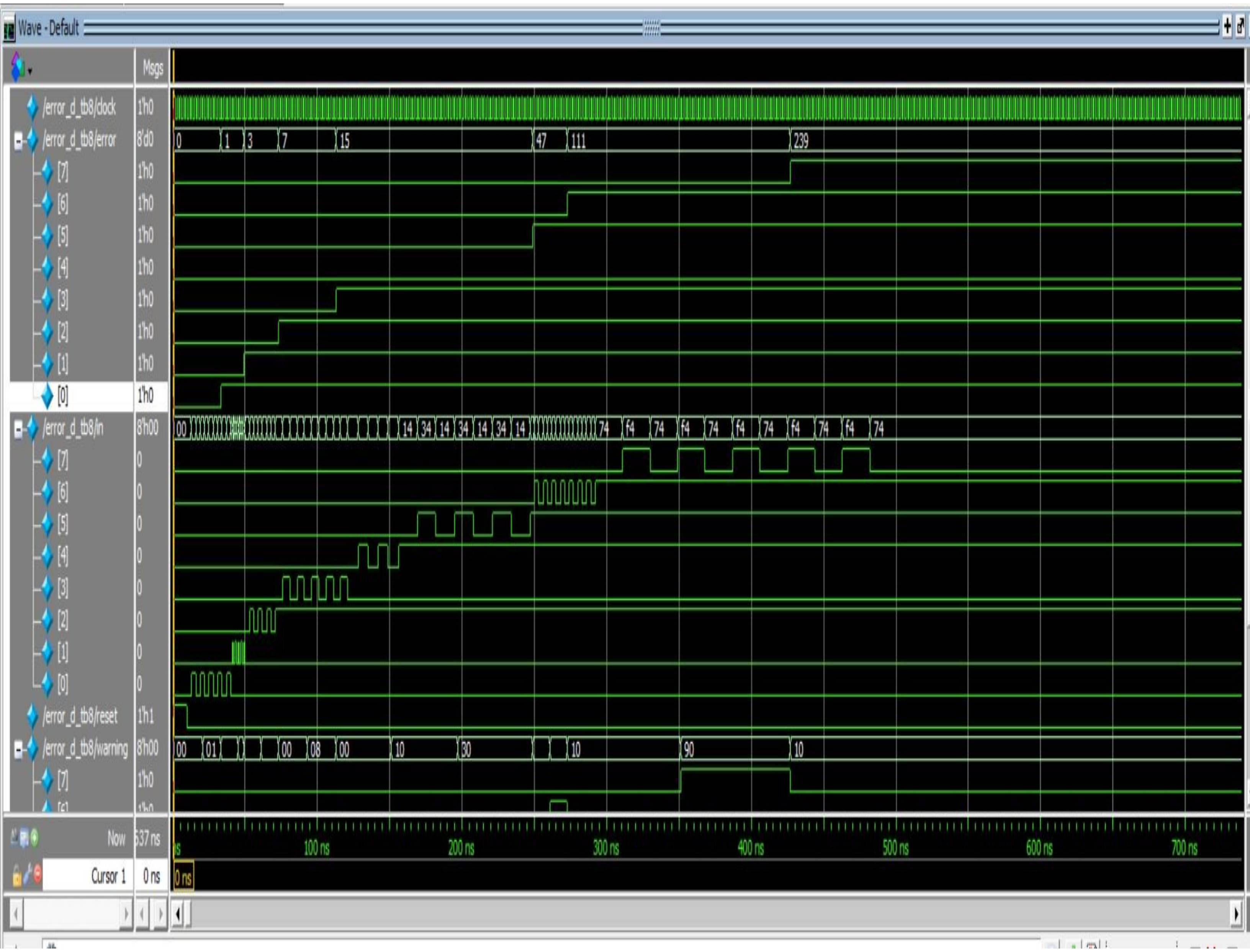
Code:

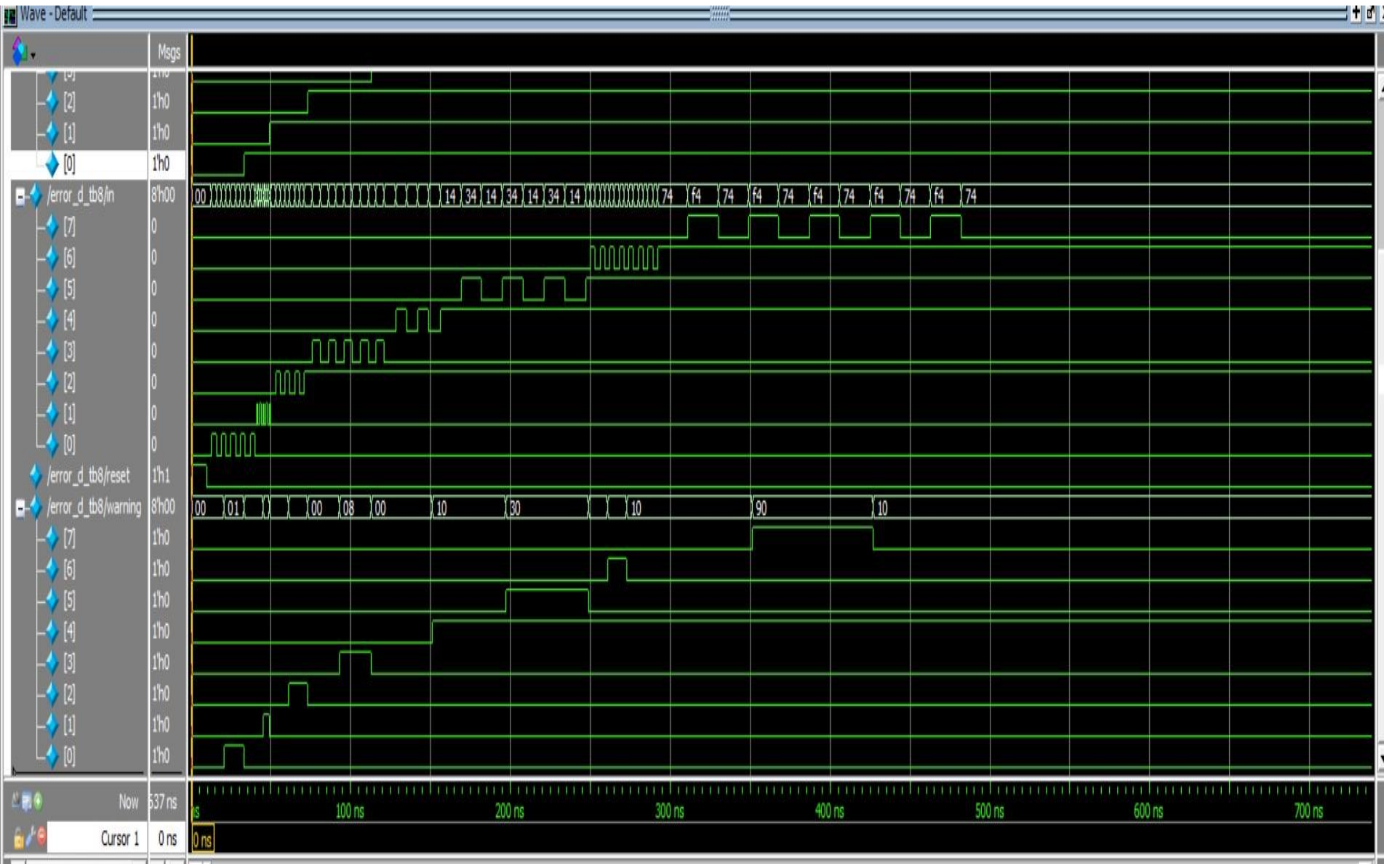
```
`timescale 1ns/1ns

module  error_d8 #(parameter n = 64,parameter h = 8)(input [h-1:0]in,input clock,input
reset,output [h-1:0] warning,output [h-1:0] error);
    genvar k;
    generate
    for (k = 0;k < h ;k = k + 1) begin
        error_d #(n,h) temp(in[k],clock,reset,warning[k],error[k]);
    end
    endgenerate

endmodule

```

Testbench:

```
`timescale 1ns/1ns

module error_d_tb8 ();

parameter n = 64;
parameter h = 8;

reg clock;
reg reset;
reg [h-1:0] in;
wire [h-1:0] warning;
wire [h-1:0] error;

error_d8 #(n,h) name(in,clock,reset,warning,error);

initial begin
    clock = 0;
    forever #1 clock = ~clock;
end

initial begin
    reset = 1;
    in = 0;

    #10
    reset = 0;

    repeat (10) begin
        #3 in[0] = ~in[0];
    end
    repeat (10) begin
        #1 in[1] = ~in[1];
    end
    repeat (7) begin
        #3 in[2] = ~in[2];
    end
    repeat (10) begin
        #5 in[3] = ~in[3];
    end
    repeat (5) begin
        #7 in[4] = ~in[4];
    end
    repeat (7) begin
        #13 in[5] = ~in[5];
    end
    repeat (15) begin
        #3 in[6] = ~in[6];
    end
    repeat (10) begin
        #19 in[7] = ~in[7];
    end
end

endmodule

```

Simulations


Wave - Default
Msgs
/error_d_tb8/clock 1'h0
/error_d_tb8/error 8'd0
0 1 3 7 15 47 111 239
[7] 1'h0
[6] 1'h0
[5] 1'h0
[4] 1'h0
[3] 1'h0
[2] 1'h0
[1] 1'h0
[0] 1'h0
/error_d_tb8/in 8'h00
00 14 34 14 34 14 34 14 74 f4 74 f4 74 f4 74 f4 74 f4 74
[7] 0
[6] 0
[5] 0
[4] 0
[3] 0
[2] 0
[1] 0
[0] 0
/error_d_tb8/reset 1'h1
/error_d_tb8/warning 8'h00
00 01 00 08 00 10 30 10 90 10
[7] 1'h0
Now 537 ns
Cursor 1 0 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns


Wave - Default
Msgs
[2] 1'h0
[1] 1'h0
[0] 1'h0
/error_d_tb8/in 8'h00
[7] 0
[6] 0
[5] 0
[4] 0
[3] 0
[2] 0
[1] 0
[0] 0
/error_d_tb8/reset 1'h1
/error_d_tb8/warning 8'h00
[7] 1'h0
[6] 1'h0
[5] 1'h0
[4] 1'h0
[3] 1'h0
[2] 1'h0
[1] 1'h0
[0] 1'h0
Now 537 ns
Cursor 1 0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns