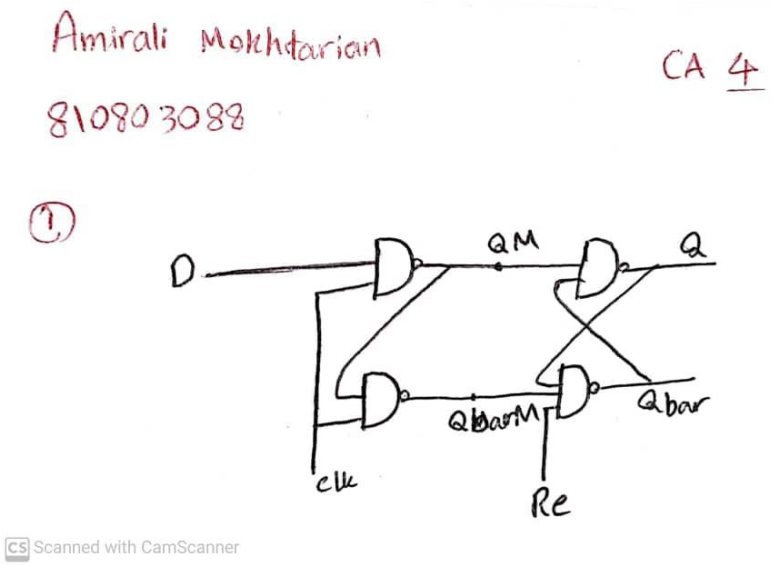
Amirali Mokhtarian CA4 FALL1404

810803088

P1

```verilog
module DLR(input D,clk,Re , output Q);
    wire QM, QbarM , Qbar;
    nand(QM , D , clk); //if we want to reset while clock is on we can add a Re here which is overwrite
    nand(QbarM , QM , clk);
    nand(Qbar , Q , QbarM , Re);
    nand(Q,QM,Qbar);
endmodule
```

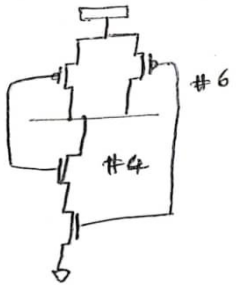


P2

```
1  `timescale 1ns/1ns
2  module DLR2(input D,clk,Re , output Q);
3      wire QM, QbarM , Qbar;
4      nand #8 (QM , D , clk);
5      nand #8 (QbarM , QM , clk);
6      nand #12 (Qbar , Q , QbarM , Re);
7      nand #8 (Q,QM,Qbar);
8  endmodule
```

2. gates delay:

Nand:



to 0 delay:

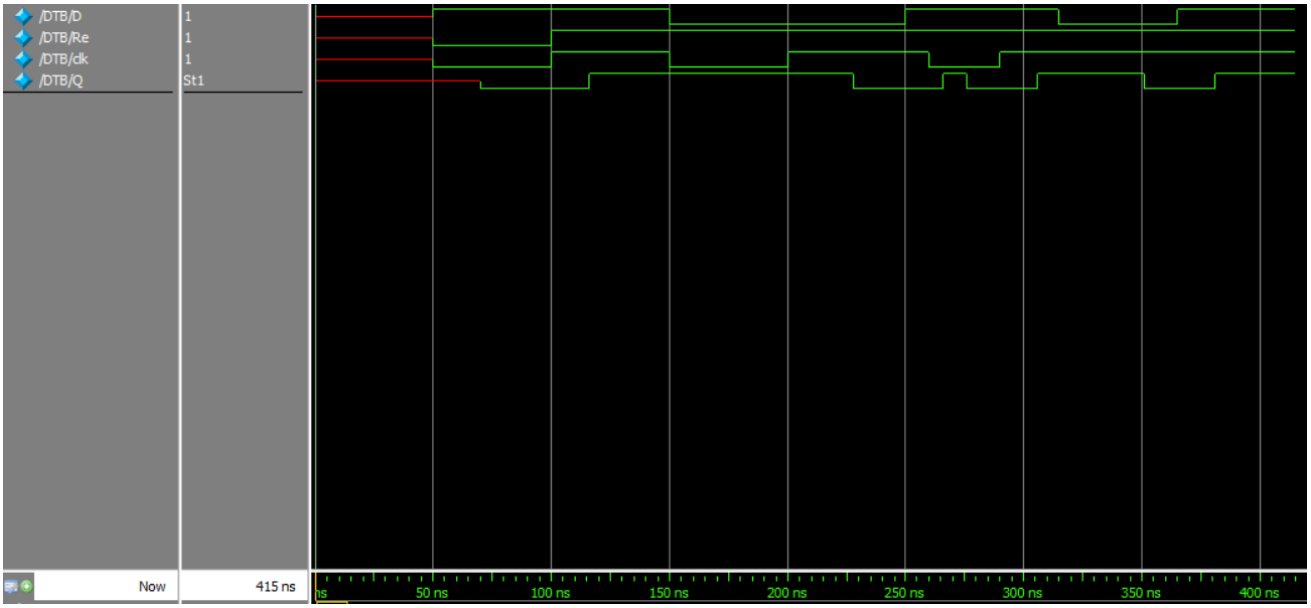
4 + 4 = 8 = to 1 & to 0 delay

3-input Nand:

4 + 4 + 4 = 12ns to 1, to 0 delay ✓

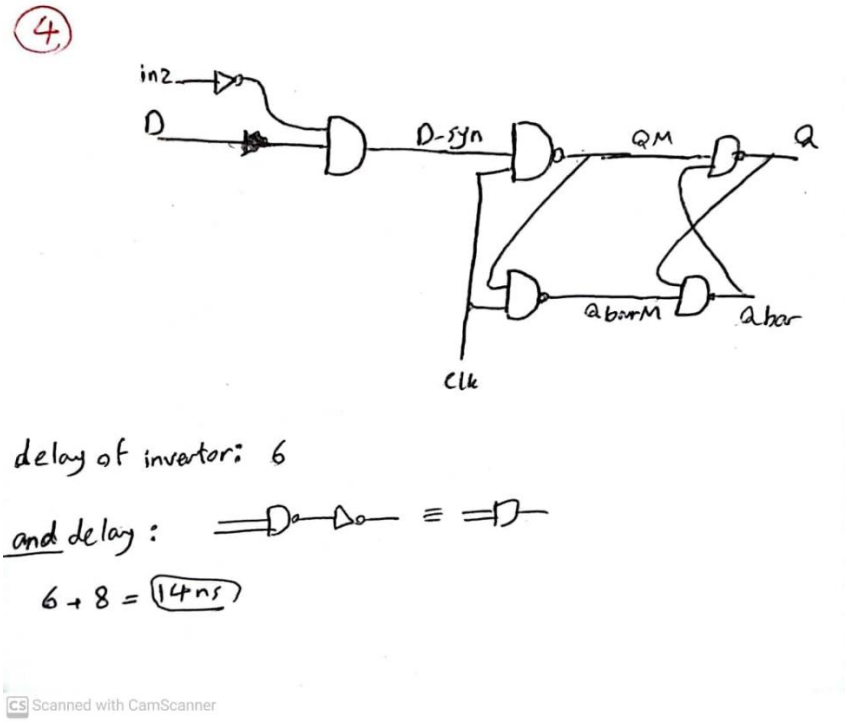
P3

```
`timescale 1ns/1ns
module DTB();
    reg D,Re,clk;
    wire Q;
    DLR2 DT(.D(D), .Re(Re), .clk(clk), .Q(Q));

    initial begin
        #50;
        D=1'b1; clk=1'b0;
        Re=1'b0; #50;

        Re=1'b1; clk=1'b1; #50;

        clk=1'b0; D=1'b0; #50;
        clk=1'b1; #50;

        //glitch
        D=1'b1; #10;
        clk=1'b0; #30;
        clk=1'b1; #25;

        //trancparency
        D=1'b0; #50;
        D=1'b1; #50;

        $finish;
    end
endmodule
```



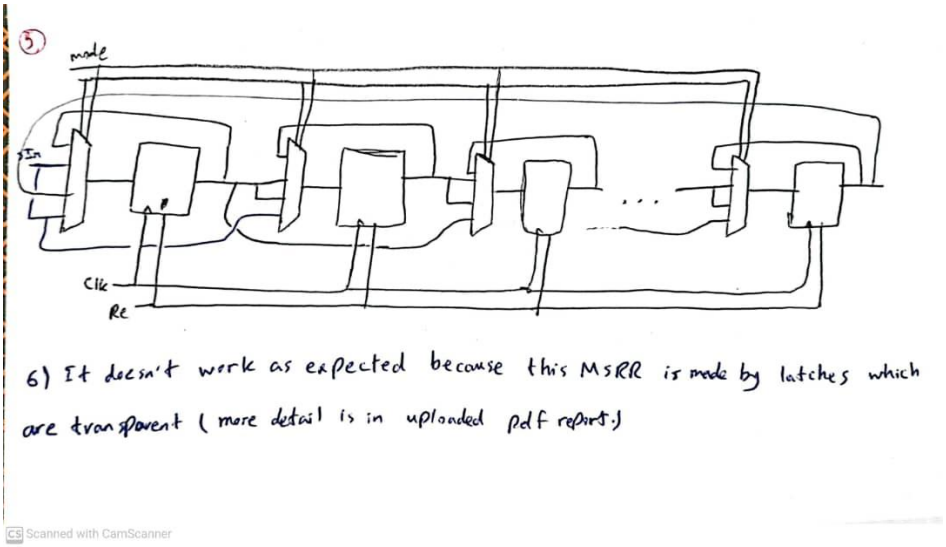
P4

```verilog
module DLR_inz(input D,clk,Re,inz , output Q);
    wire QM, QbarM , Qbar;
    wire D_syn, inzBAR;

    not #6 (inzBAR , inz);

    and #14 (D_syn , inzBAR , D);
    nand #8 (QM , D_syn , clk);
    nand #8 (QbarM , QM , clk);
    nand #12(Qbar , Q , QbarM , Re);
    nand #8 (Q,QM,Qbar);
endmodule
```



P5

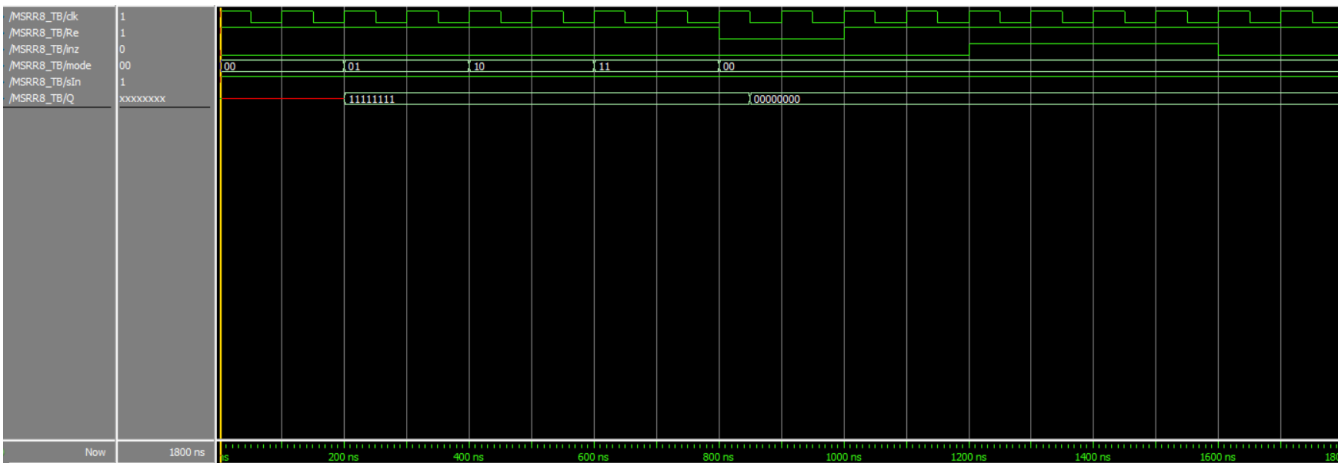
```
module MSRR8(input clk, Re, inz, sIn , input [1:0]mode , output [7:0]Q);
    reg [7:0] D_in;
    generate
        genvar i;
        for(i=0 ; i<8 ; i=i+1) begin
            always @(*) begin
                case (mode)

                    2'b00: D_in[i]=Q[i];

                    2'b01: begin
                    if(i==7)
                        D_in[i]=sIn;
                    else
                        D_in[i]=Q[i+1];
                    end

                    2'b10: begin
                    if(i==7)
                        D_in[i]=Q[0];
                    else
                        D_in[i]=Q[i+1];
                    end

                    2'b11: begin
                        if(i==7 || i==6)
                            D_in[i]=sIn;
                        else
                            D_in[i]=Q[i+2];
                    end

                endcase
            end

            DLR_inz latch_i(.D(D_in[i]), .clk(clk) , .Re(Re) , .inz(inz), .Q(Q[i]));
        end
    endgenerate
endmodule
```



6) It doesn't work as expected because this MSRR is made by latches which are transparent (more detail is in uploaded pdf report.)

P6

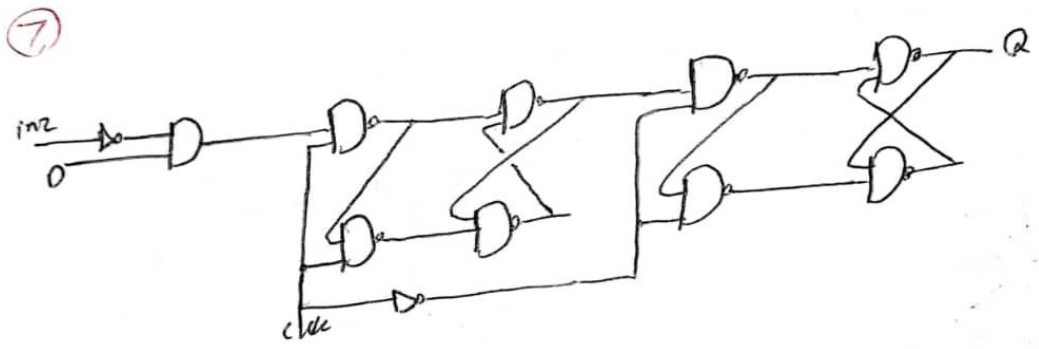
```
1  `timescale 1ns/1ns
2  module MSRR8_TB();
3      reg clk, Re, inz;
4      reg [1:0] mode;
5      reg sIn;
6      wire [7:0] Q;
7
8      MSRR8 dut(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Q));
9
10     initial begin
11         clk = 1'b0;
12         forever begin
13            clk = ~clk; #50;
14         end
15     end
16
17     initial begin
18         sIn  = 1'b1;
19         mode = 2'b00;
20         Re=1'b1;
21         inz=1'b0;
22
23         mode = 2'b00; #200;
24         mode = 2'b01; #200;
25         mode = 2'b10; #200;
26         mode = 2'b11; #200;
27
28         mode = 2'b00;
29         Re = 1'b0; #200;
30         Re = 1'b1; #200;
31         inz = 1'b1; #400;
32         inz = 1'b0; #200;
33
34         $finish;
35     end
36 endmodule
37
```



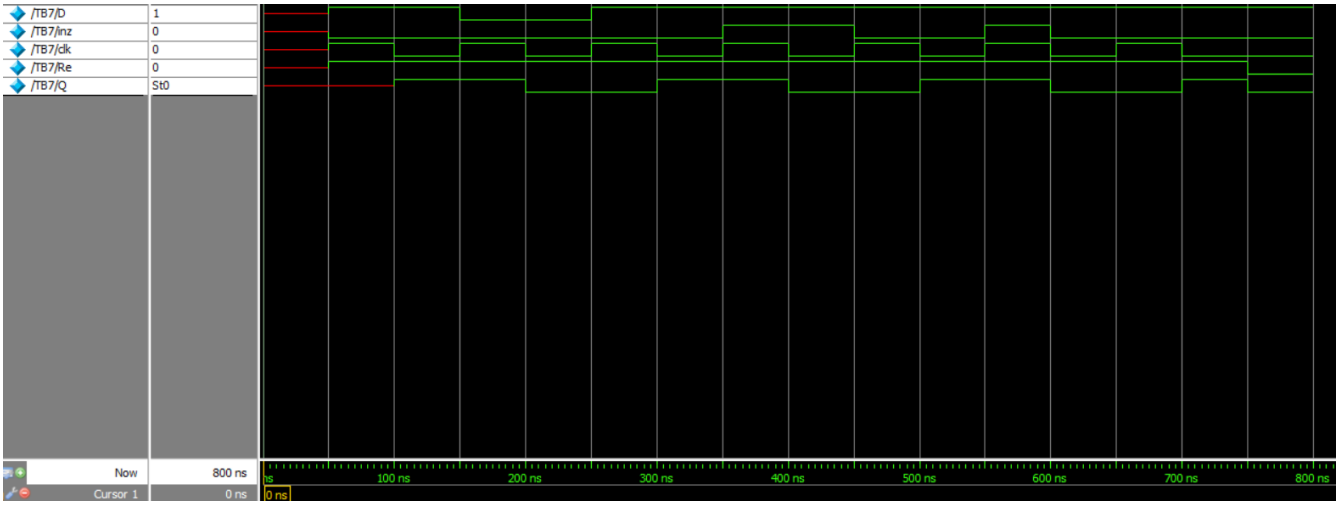
Our MSRR8 is built using master–slave D latches, not edge-triggered D flip-flops. Because latches are transparent when their clock input is active, the internal value can propagate through multiple stages during the same clock level. This transparency is the main cause of unexpected behavior.

P7

```verilog
module DLR(input D,clk,Re , output Q);
    wire QM, QbarM , Qbar;
    nand(QM , D , clk);
    nand(QbarM , QM , clk);
    nand(Qbar , Q , QbarM , Re);
    nand(Q,QM,Qbar);
endmodule

module DFF(input D , clk , inz , Re , output Q);
    wire D_syn , inzBAR , clkBAR , Q1;
    not(inzBAR , inz);
    and(D_syn , D , inzBAR);

    DLR dlr1(D_syn , clk , Re , Q1);
    not(clkBAR , clk);
    DLR dlr2(Q1 , clkBAR , Re , Q);
endmodule


```



```
`timescale 1ns/1ns
module TB7();
    reg D,inz,clk,Re;
    wire Q;
    DFF dfftb(.D(D) , .inz(inz) , .Re(Re) , .clk(clk) , .Q(Q));
    initial begin
        #50;
        {D,inz,clk,Re}=4'b1011; #50;
        {D,inz,clk,Re}=4'b1001; #50;
        {D,inz,clk,Re}=4'b0011; #50;
        {D,inz,clk,Re}=4'b0001; #50;
        {D,inz,clk,Re}=4'b1011; #50;
        {D,inz,clk,Re}=4'b1001; #50;

        {D,inz,clk,Re}=4'b1111; #50;
        {D,inz,clk,Re}=4'b1101; #50;

        {D,inz,clk,Re}=4'b1011; #50;
        {D,inz,clk,Re}=4'b1001; #50;

        {D,inz,clk,Re}=4'b1111; #50;
        {D,inz,clk,Re}=4'b1001; #50; //inz=0 so it looks like it shouldn't reset
        //but the value 1 doesn't propogate yet because of edge triggering

        //Re checking:
        {D,inz,clk,Re}=4'b1011; #50;
        {D,inz,clk,Re}=4'b1001; #50;
        {D,inz,clk,Re}=4'b1000; #50;
    end

endmodule
```

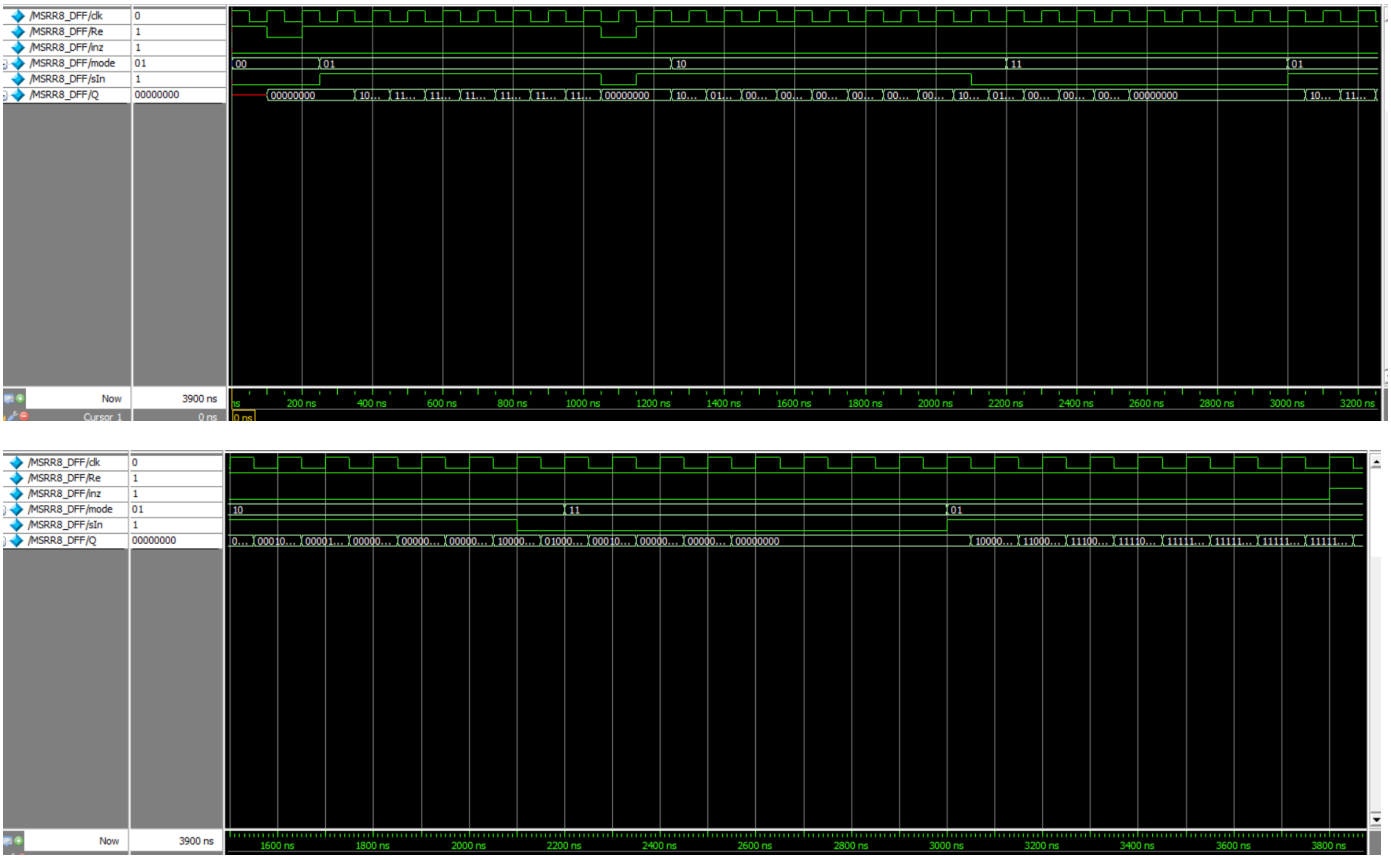


P8

```
module MSRR8_FF (input clk, Re, inz, sIn, input [1:0]mode , output [7:0]Q);
    reg [7:0]D_in;
    generate
        genvar i;
        for(i=0 ; i<8 ; i=i+1) begin
            always @(sIn, clk, Q) begin
                case (mode)

                    2'b00: D_in[i]=Q[i];

                    2'b01: begin
                        if(i==7)
                            D_in[i]=sIn;
                        else
                            D_in[i]=Q[i+1];
                    end

                    2'b10: begin
                        if(i==7)
                            D_in[i]=Q[0];
                        else
                            D_in[i]=Q[i+1];
                    end

                    2'b11: begin
                        if(i==7 || i==6)
                            D_in[i]=sIn;
                        else
                            D_in[i]=Q[i+2];
                    end
                endcase
            end
            DFF DD(.D(D_in[i]), .clk(clk) , .Re(Re) , .inz(inz), .Q(Q[i]));
        end
    endgenerate
endmodule
```

```
`timescale 1ns/1ns
module MSRR8_DFF();
    reg clk, Re, inz;
    reg [1:0] mode;
    reg sIn;
    wire [7:0] Q;

    MSRR8_FF MSTB(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Q));

    initial begin
        clk = 1'b0;
        forever begin
            clk = ~clk; #50;
            end
        end

    initial begin
        mode = 2'b00;
        sIn  = 1'b0;
        inz  = 1'b0;
        Re   = 1'b1;
        #100;

        Re = 1'b0;
        #100;

        Re=1'b1; #50;
        sIn=1'b1; mode=2'b01; #800;
        sIn=1'b0;
        Re=1'b0; #100;
        Re=1'b1;
        sIn=1; #100;

        mode=2'b10; #850;

        sIn=1'b0; #100;
        mode=2'b11; #800;

        sIn=1'b1;
        mode = 2'b01; #800;
        inz=1'b1; #100;
        $finish;
    end
endmodule

```

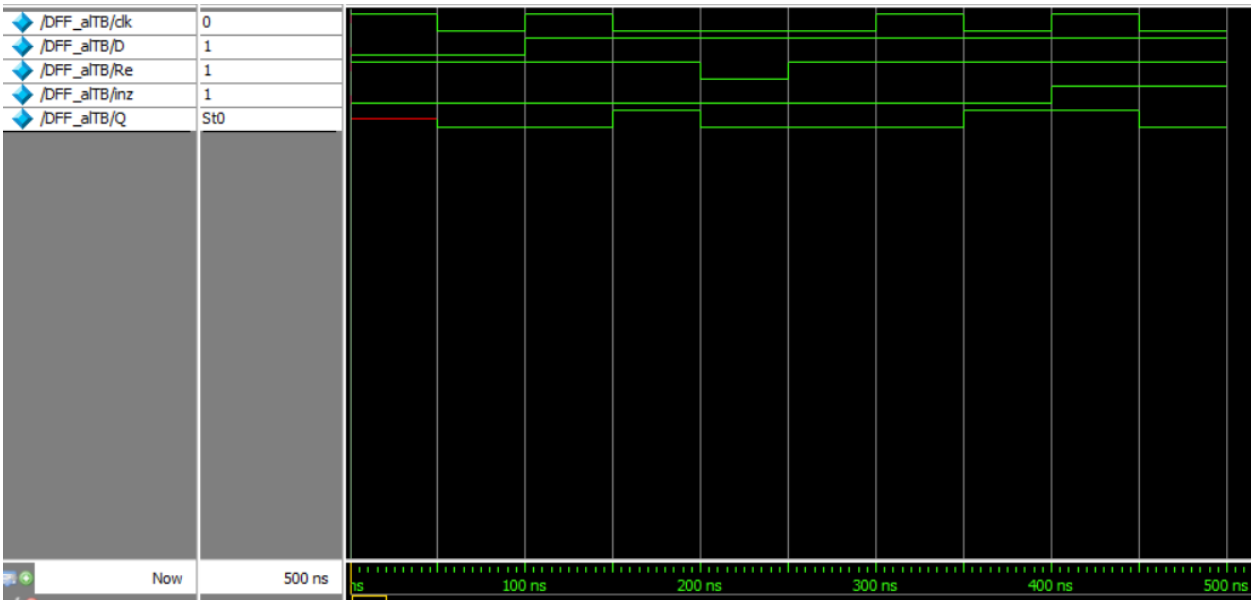
Waveform of 8



P9

```verilog
module DFF_al(input clk, Re, inz , D , output reg Q);
    reg QM;
    always @(posedge clk) begin
        QM = D;
    end

    always @(negedge clk) begin

        if(Re==1'b0)begin
            Q<=1'b0;
        end

        else begin
            if(inz)
                Q<=1'b0;
            else
                Q<=QM;
        end
    end

    assign Q = Q ? Re : 1'b0;

endmodule
```

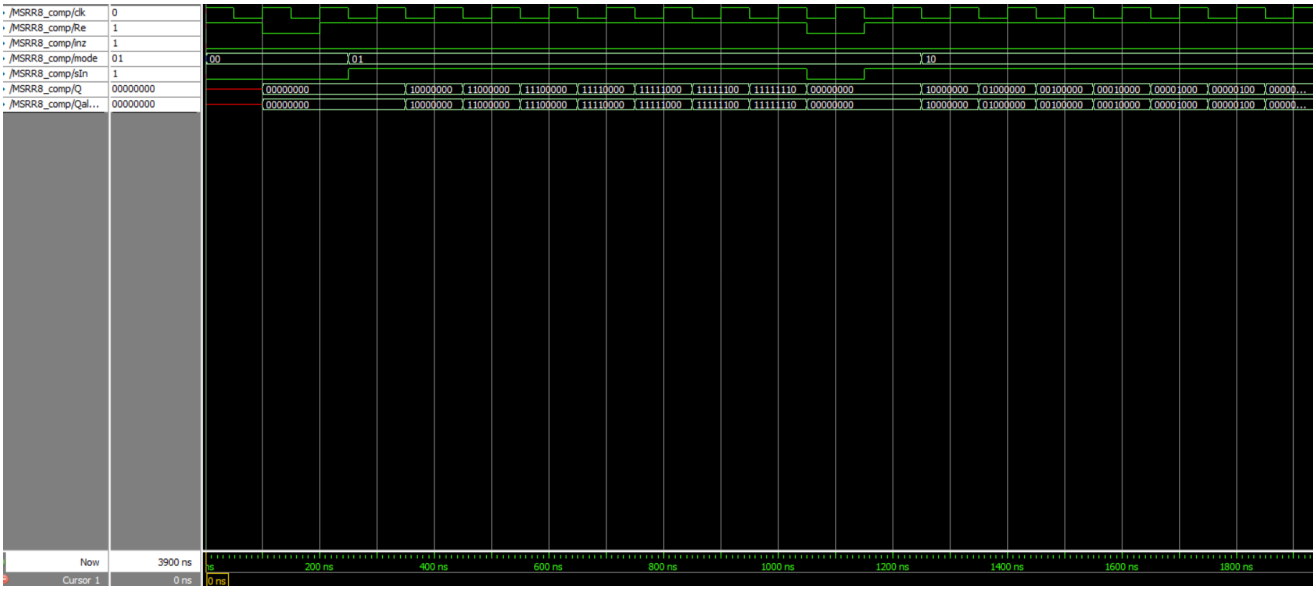
```
1  `timescale 1ns/1ns
2
3  module DFF_alTB();
4      reg clk, D, Re , inz;
5      wire Q;
6      DFF_al DF(.clk(clk), .Re(Re), .inz(inz) , .D(D) , .Q(Q));
7      initial begin
8          D=1'b0; inz=1'b0; Re=1'b1; clk=1'b1; #50
9
10         clk=1'b0; #50;
11         D=1'b1; clk=1'b1; #50
12         clk=1'b0; #50;
13
14         Re=1'b0; #50;
15         Re=1'b1; #50;
16
17         clk=1'b1; D=1'b1; #50;
18         clk=1'b0; #50;
19
20         clk=1'b1; inz=1'b1; #50;
21         clk=1'b0; #50;
22
23         $finish;
24
25     end
26 endmodule
```

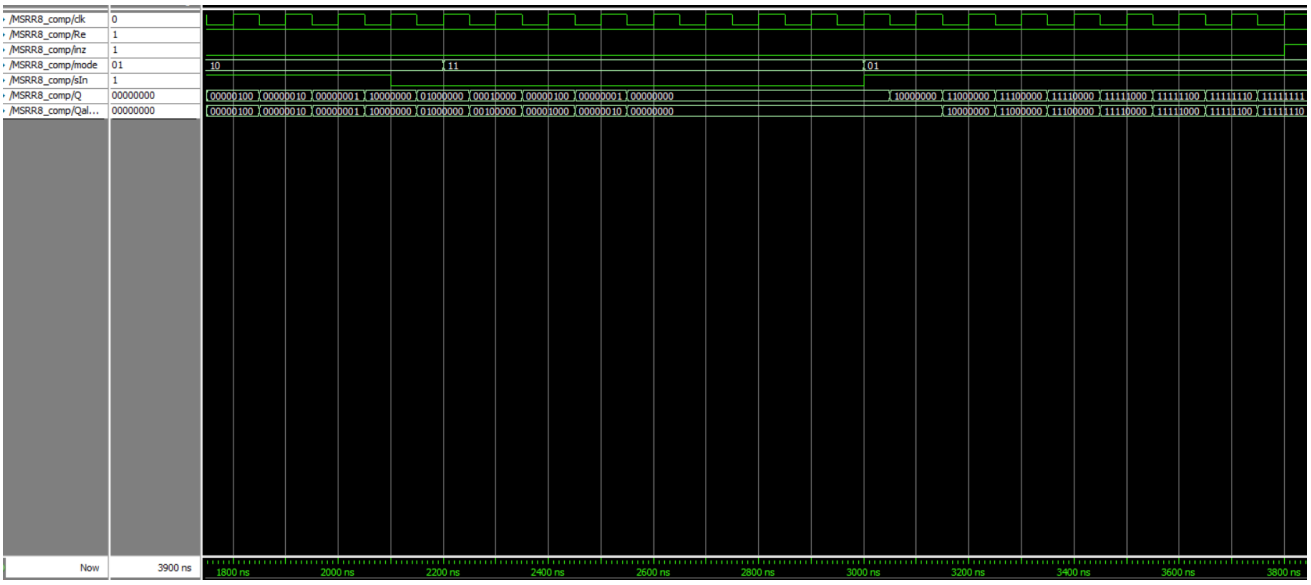


P10

```
module MSRR8(input clk, Re, inz, sIn , input [1:0]mode , output [7:0]Q);
    reg [7:0] D_in;
    generate
        genvar i;
        for(i=0 ; i<8 ; i=i+1) begin
            always @(*) begin
                case (mode)

                    2'b00: D_in[i]=Q[i];

                    2'b01: begin
                    if(i==7)
                        D_in[i]=sIn;
                    else
                        D_in[i]=Q[i+1];
                    end

                    2'b10: begin
                    if(i==7)
                        D_in[i]=Q[0];
                    else
                        D_in[i]=Q[i+1];
                    end

                    2'b11: begin
                        if(i==7 || i==6)
                            D_in[i]=sIn;
                        else
                            D_in[i]=Q[i+2];
                    end

                endcase
            end

            DLR_inz latch_i(.D(D_in[i]), .clk(clk) , .Re(Re) , .inz(inz), .Q(Q[i]));
        end
    endgenerate
endmodule
```

```
`timescale 1ns/1ns
module MSRR8_comp();
    reg clk, Re, inz;
    reg [1:0] mode;
    reg sIn;
    wire [7:0] Q,Qalways;

    MSRR8_FF MSTB(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Q));
    MSRR8_alw MALW(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Qalways));

    initial begin
        clk = 1'b0;
        forever begin
            clk = ~clk; #50;
            end
        end

    initial begin
        mode = 2'b00;
        sIn  = 1'b0;
        inz  = 1'b0;
        Re   = 1'b1;
        #100;

        Re = 1'b0;
        #100;

        Re=1'b1; #50;
        sIn=1'b1; mode=2'b01; #800;
        sIn=1'b0;
        Re=1'b0; #100;
        Re=1'b1;
        sIn=1; #100;

        mode=2'b10; #850;

        sIn=1'b0; #100;
        mode=2'b11; #800;

        sIn=1'b1;
        mode = 2'b01; #800;
        inz=1'b1; #100;
        $finish;
    end
endmodule

```





They aren’t match exactly because In the gate-level circuit, signals pass through several NAND gates, causing small timing shifts or brief glitches that affect the output. The always-based register has no delays and produces clean, synchronous transitions. The main point is that always_based is an ideal MSRR(with no delays) and it’s functionally correct(not in a real world).