

810803088

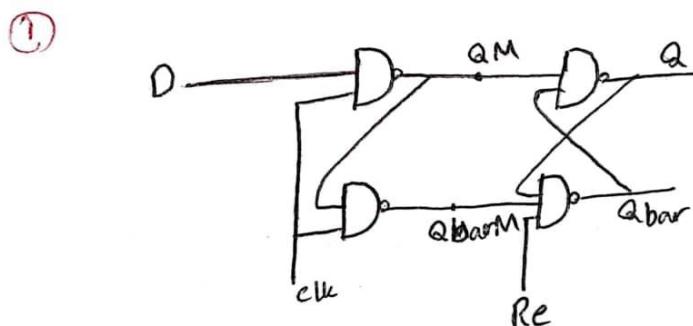
P1

```
● ● ●  
1 module DLR(input D,clk,Re , output Q);  
2   wire QM, QbarM , Qbar;  
3   nand(QM , D , clk); //if we want to reset while clock is on we can add a Re here which is overwrite  
4   nand(QbarM , QM , clk);  
5   nand(Qbar , Q , QbarM , Re);  
6   nand(Q,QM,Qbar);  
7 endmodule
```

Amirali Mokhtarian

CA 4

810803088



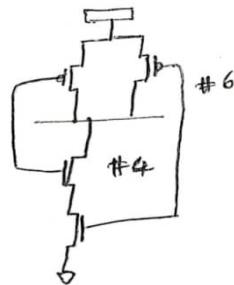
Scanned with CamScanner

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:

None:



to 0 delay:

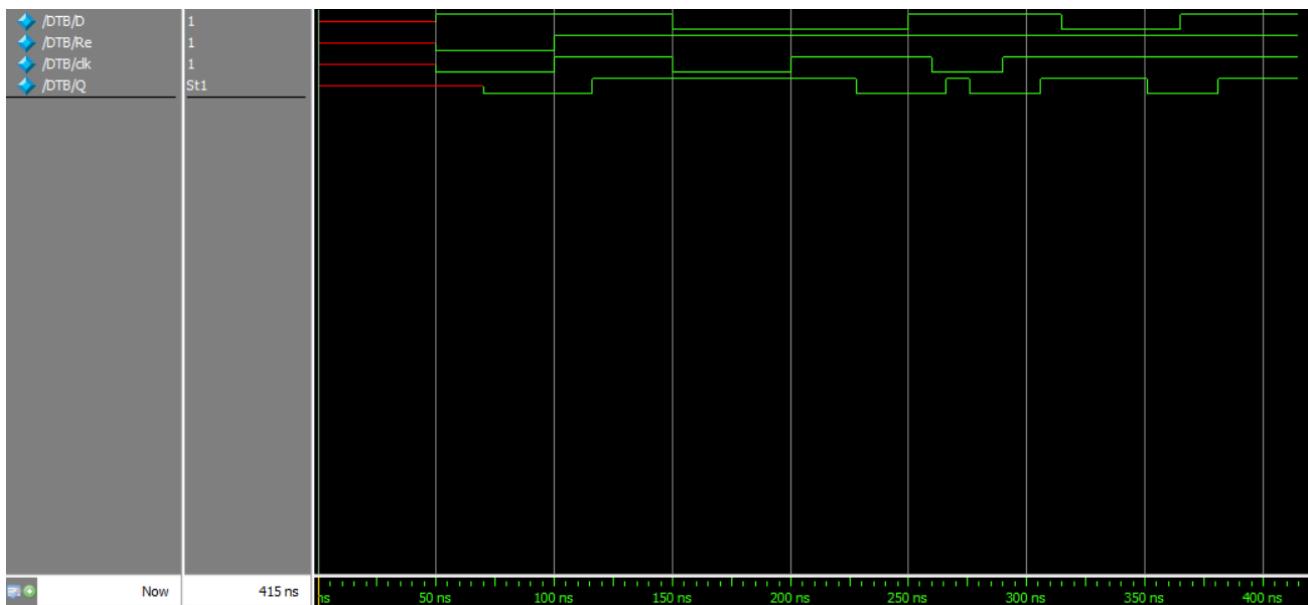
$$4 + 4 = 8 \text{ ns} = \text{to 1 & to 0 delay}$$

3-input Nand:

$$4 + 4 + 4 = 12 \text{ ns} \text{ to 1, to 0 delay ✓}$$

P3

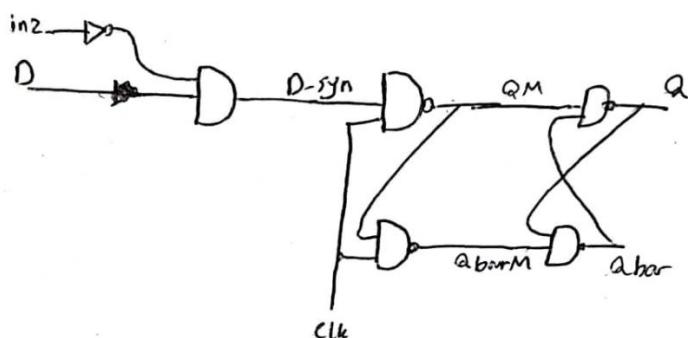
```
● ○ ● ●  
1 `timescale 1ns/1ns  
2 module DTB();  
3   reg D,Re,clk;  
4   wire Q;  
5   DLR2 DT(.D(D), .Re(Re), .clk(clk), .Q(Q));  
6  
7   initial begin  
8     #50;  
9     D=1'b1; clk=1'b0;  
10    Re=1'b0; #50;  
11  
12    Re=1'b1; clk=1'b1; #50;  
13  
14    clk=1'b0; D=1'b0; #50;  
15    clk=1'b1; #50;  
16  
17    //glitch  
18    D=1'b1; #10;  
19    clk=1'b0; #30;  
20    clk=1'b1; #25;  
21  
22    //transparency  
23    D=1'b0; #50;  
24    D=1'b1; #50;  
25  
26    $finish;  
27  end  
28 endmodule
```



P4

```
● ● ●  
1 module DLR_inz(input D,clk,Re,inz , output Q);  
2     wire QM, QbarM , Qbar;  
3     wire D_syn, inzBAR;  
4  
5     not #6 (inzBAR , inz);  
6  
7     and #14 (D_syn , inzBAR , D);  
8     nand #8 (QM , D_syn , clk);  
9     nand #8 (QbarM , QM , clk);  
10    nand #12(Qbar , Q , QbarM , Re);  
11    nand #8 (Q,QM,Qbar);  
12 endmodule
```

④



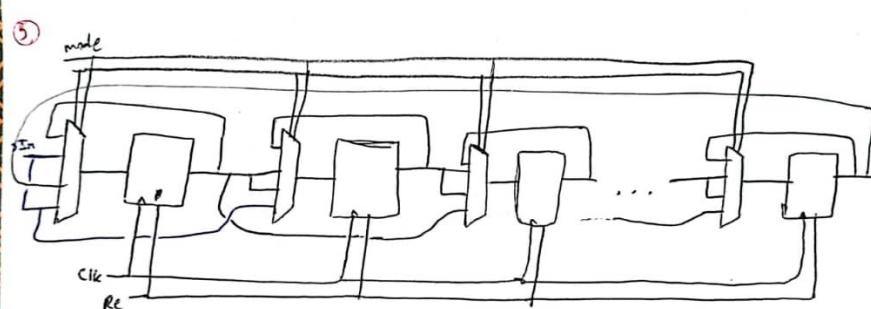
delay of inverter: 6

and delay : ~~= D - D = D~~

$$6 + 8 = \underline{14 \text{ ns}}$$

P5

```
1 module MSRR8(input clk, Re, inz, sIn , input [1:0]mode , output [7:0]Q);
2   reg [7:0] D_in;
3   generate
4     genvar i;
5     for(i=0 ; i<8 ; i=i+1) begin
6       always @(*) begin
7         case (mode)
8           2'b00: D_in[i]=Q[i];
10          2'b01: begin
11            if(i==7)
12              D_in[i]=sIn;
13            else
14              D_in[i]=Q[i+1];
15            end
16
17          2'b10: begin
18            if(i==7)
19              D_in[i]=Q[0];
20            else
21              D_in[i]=Q[i+1];
22            end
23
24          2'b11: begin
25            if(i==7 || i==6)
26              D_in[i]=sIn;
27            else
28              D_in[i]=Q[i+2];
29            end
30
31        endcase
32      end
33
34      DLR_inz latch_i(.D(D_in[i]), .clk(clk) , .Re(Re) , .inz(inz), .Q(Q[i]));
35    end
36  endgenerate
37 endmodule
```

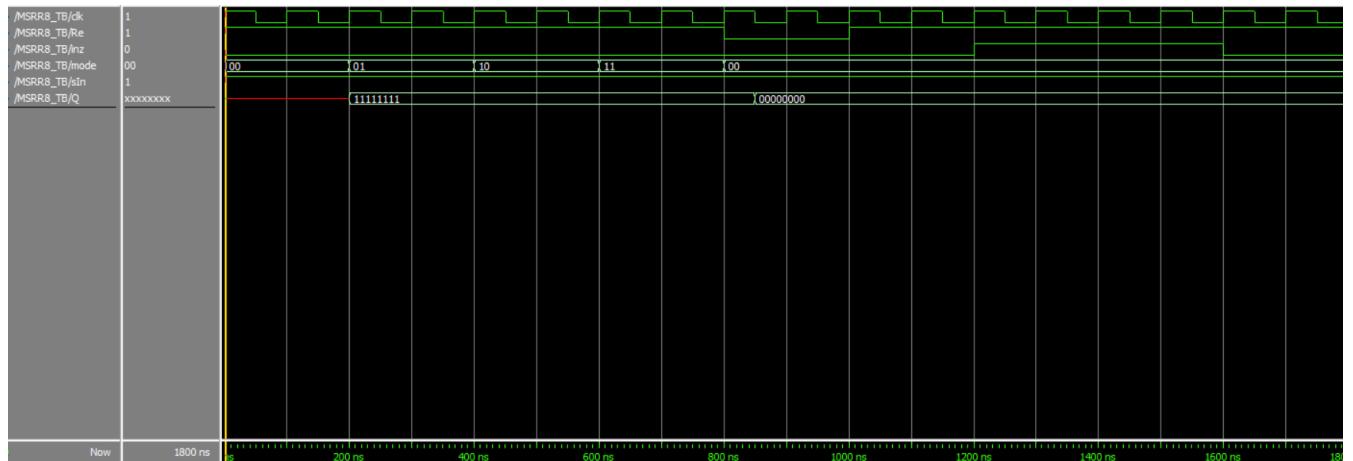


6) It doesn't work as expected because this MSRR is made by latches which are transparent (more detail is in uploaded pdf report!)

P6

```
● ● ●

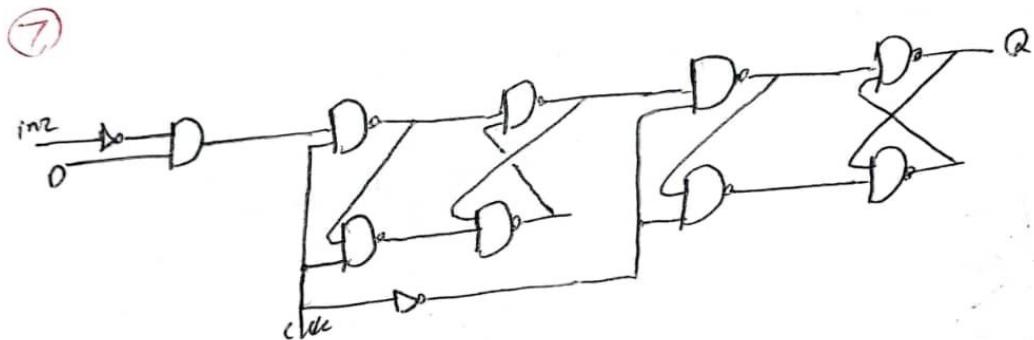
1  `timescale 1ns/1ns
2  module MSRR8_TB();
3      reg clk, Re, inz;
4      reg [1:0] mode;
5      reg sIn;
6      wire [7:0] Q;
7
8      MSRR8 dut(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Q));
9
10     initial begin
11         clk = 1'b0;
12         forever begin
13             clk = ~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

```
● ● ●  
1 module DLR(input D,clk,Re , output Q);  
2     wire QM, QbarM , Qbar;  
3     nand(QM , D , clk);  
4     nand(QbarM , QM , clk);  
5     nand(Qbar , Q , QbarM , Re);  
6     nand(Q,QM,Qbar);  
7 endmodule  
8  
9 module DFF(input D , clk , inz , Re , output Q);  
10    wire D_syn , inzBAR , clkBAR , Q1;  
11    not(inzBAR , inz);  
12    and(D_syn , D , inzBAR);  
13  
14    DLR dlr1(D_syn , clk , Re , Q1);  
15    not(clkBAR , clk);  
16    DLR dlr2(Q1 , clkBAR , Re , Q);  
17 endmodule  
18  
19
```

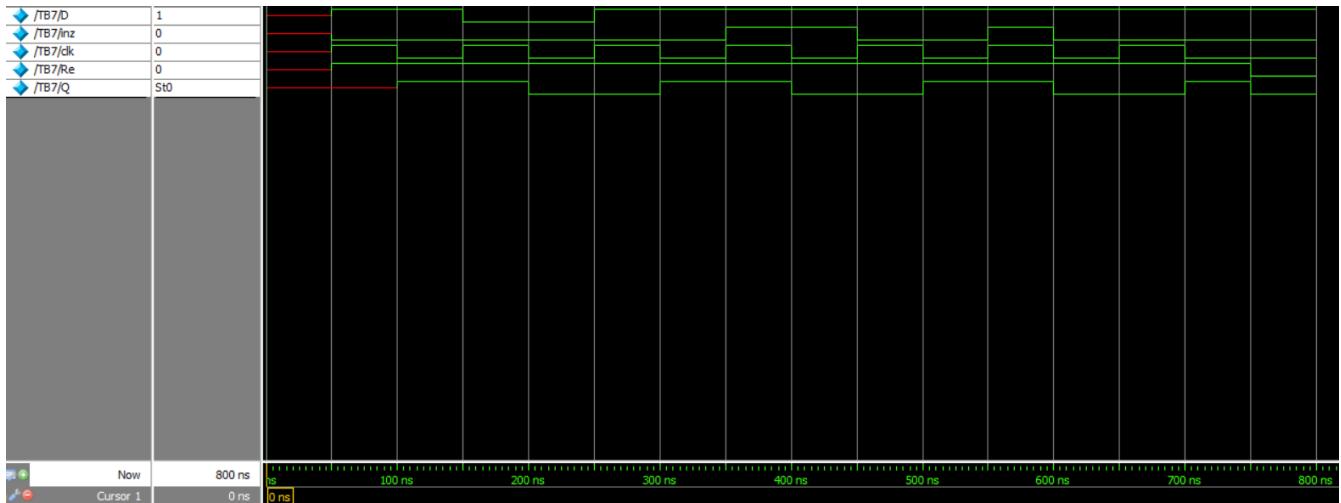


```

● ● ●

1  `timescale 1ns/1ns
2  module TB7();
3      reg D,inz,clk,Re;
4      wire Q;
5      DFF dfftb(.D(D) , .inz(inz) , .Re(Re) , .clk(clk) , .Q(Q));
6      initial begin
7          #50;
8          {D,inz,clk,Re}=4'b1011; #50;
9          {D,inz,clk,Re}=4'b1001; #50;
10         {D,inz,clk,Re}=4'b0011; #50;
11         {D,inz,clk,Re}=4'b0001; #50;
12         {D,inz,clk,Re}=4'b1011; #50;
13         {D,inz,clk,Re}=4'b1001; #50;
14
15         {D,inz,clk,Re}=4'b1111; #50;
16         {D,inz,clk,Re}=4'b1101; #50;
17
18         {D,inz,clk,Re}=4'b1011; #50;
19         {D,inz,clk,Re}=4'b1001; #50;
20
21         {D,inz,clk,Re}=4'b1111; #50;
22         {D,inz,clk,Re}=4'b1001; #50; //inz=0 so it looks like it shouldn't reset
23         //but the value 1 doesn't propagate yet because of edge triggering
24
25         //Re checking:
26         {D,inz,clk,Re}=4'b1011; #50;
27         {D,inz,clk,Re}=4'b1001; #50;
28         {D,inz,clk,Re}=4'b1000; #50;
29     end
30
31 endmodule

```



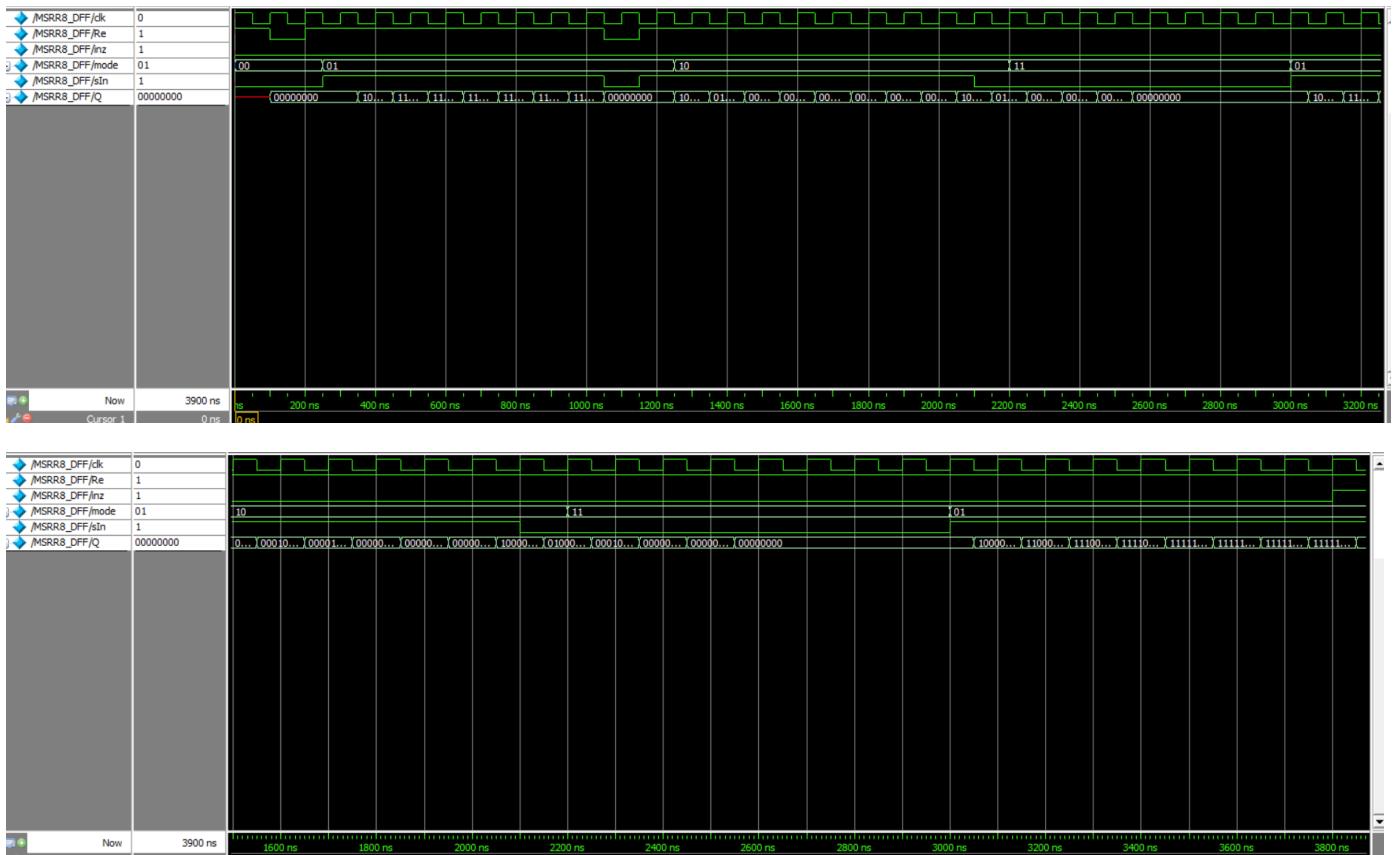
P8

```
● ● ●

1  module MSRR8_FF (input clk, Re, inz, sIn, input [1:0]mode , output [7:0]Q);
2      reg [7:0]D_in;
3      generate
4          genvar i;
5          for(i=0 ; i<8 ; i=i+1) begin
6              always @(sIn, clk, Q) begin
7                  case (mode)
8
9                      2'b00: D_in[i]=Q[i];
10
11                     2'b01: begin
12                         if(i==7)
13                             D_in[i]=sIn;
14                         else
15                             D_in[i]=Q[i+1];
16                     end
17
18                     2'b10: begin
19                         if(i==7)
20                             D_in[i]=Q[0];
21                         else
22                             D_in[i]=Q[i+1];
23                     end
24
25                     2'b11: begin
26                         if(i==7 || i==6)
27                             D_in[i]=sIn;
28                         else
29                             D_in[i]=Q[i+2];
30                     end
31                 endcase
32             end
33             DFF DD(.D(D_in[i]), .clk(clk) , .Re(Re) , .inz(inz), .Q(Q[i]));
34         end
35     endgenerate
36 endmodule
```

```
1 `timescale 1ns/1ns
2 module MSRR8_DFF();
3   reg clk, Re, inz;
4   reg [1:0] mode;
5   reg sIn;
6   wire [7:0] Q;
7
8   MSRR8_FF MSTB(.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   mode = 2'b00;
19   sIn = 1'b0;
20   inz = 1'b0;
21   Re = 1'b1;
22   #100;
23
24   Re = 1'b0;
25   #100;
26
27   Re=1'b1; #50;
28   sIn=1'b1; mode=2'b01; #800;
29   sIn=1'b0;
30   Re=1'b0; #100;
31   Re=1'b1;
32   sIn=1; #100;
33
34   mode=2'b10; #850;
35
36   sIn=1'b0; #100;
37   mode=2'b11; #800;
38
39   sIn=1'b1;
40   mode = 2'b01; #800;
41   inz=1'b1; #100;
42   $finish;
43 end
44 endmodule
45
```

Waveform of 8



P9

```

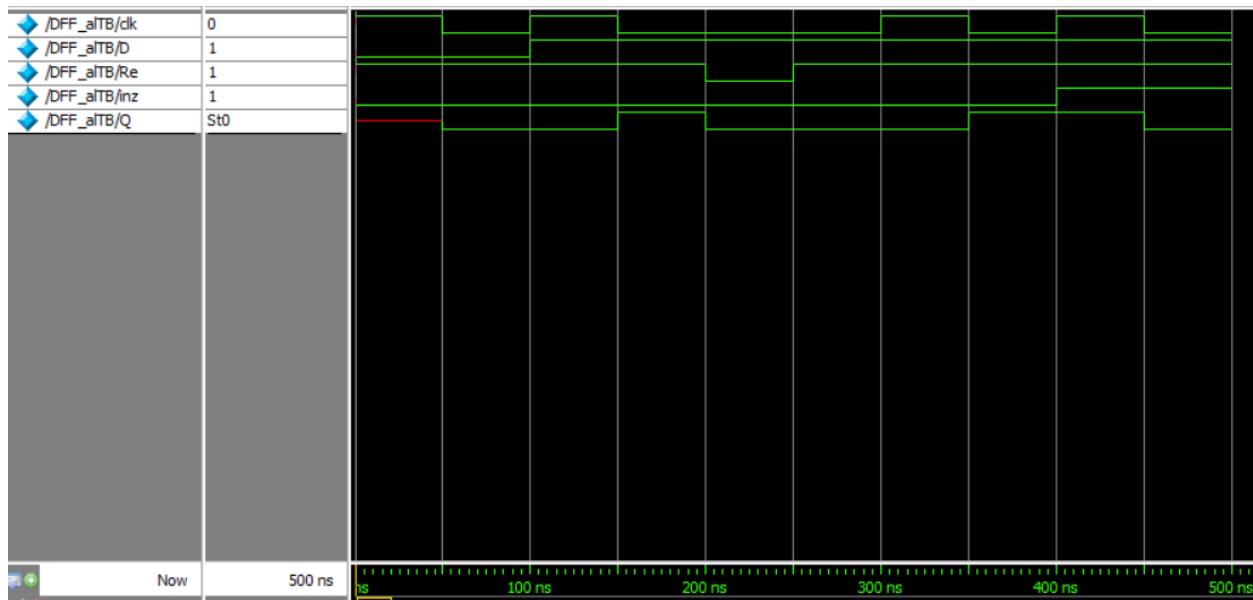
1  module DFF_al(input clk, Re, inz , D , output reg Q);
2      reg QM;
3      always @(posedge clk) begin
4          QM = D;
5      end
6
7      always @(negedge clk) begin
8
9          if(Re==1'b0)begin
10              Q<=1'b0;
11          end
12
13          else begin
14              if(inz)
15                  Q<=1'b0;
16              else
17                  Q<=QM;
18          end
19      end
20
21      assign Q = Q ? Re : 1'b0;
22
23 endmodule

```

```

1 `timescale 1ns/1ns
2
3 module DFF_aITB();
4     reg clk, D, Re , inz;
5     wire Q;
6     DFF_aI 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

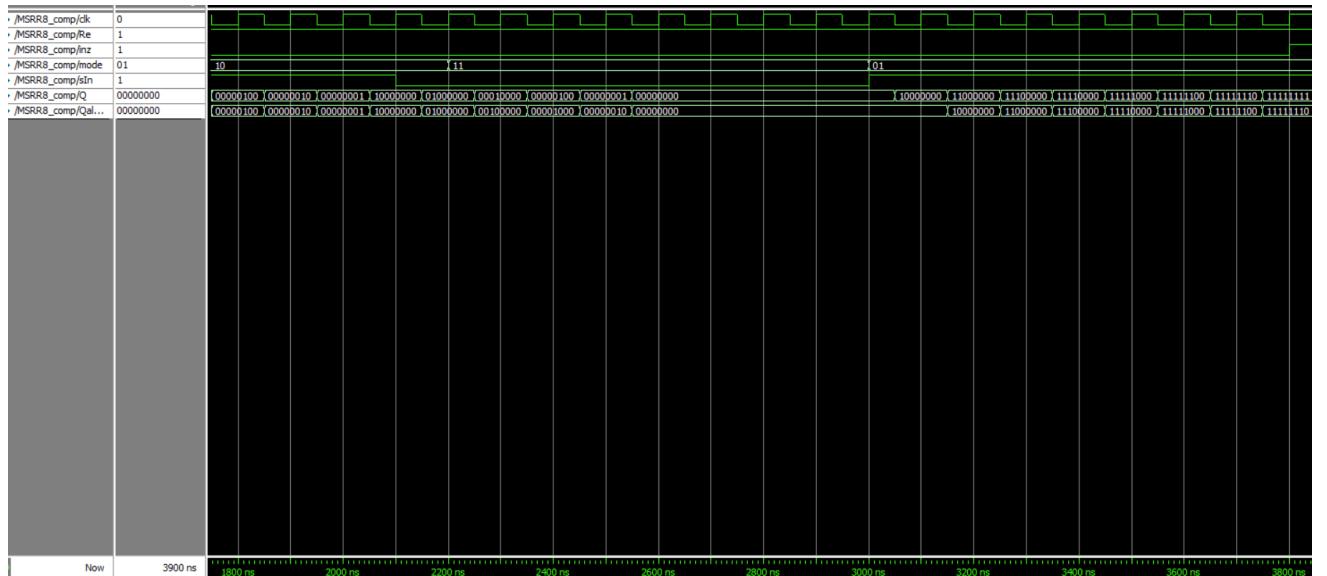
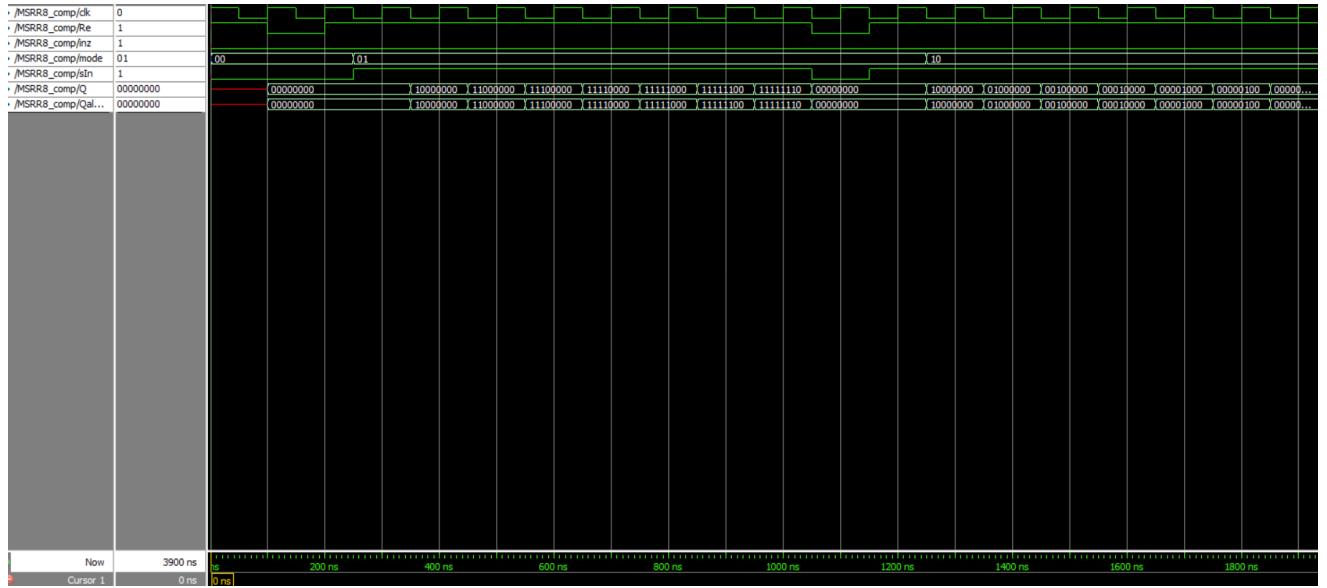
```



P10

```
1  module MSRR8(input clk, Re, inz, sIn , input [1:0]mode , output [7:0]Q);
2    reg [7:0] D_in;
3    generate
4      genvar i;
5      for(i=0 ; i<8 ; i=i+1) begin
6        always @(*) begin
7          case (mode)
8            2'b00: D_in[i]=Q[i];
10           2'b01: begin
11             if(i==7)
12               D_in[i]=sIn;
14             else
15               D_in[i]=Q[i+1];
16           end
17
18           2'b10: begin
19             if(i==7)
20               D_in[i]=Q[0];
21             else
22               D_in[i]=Q[i+1];
23           end
24
25           2'b11: begin
26             if(i==7 || i==6)
27               D_in[i]=sIn;
28             else
29               D_in[i]=Q[i+2];
30           end
31
32         endcase
33       end
34
35       DLR_inz latch_i(.D(D_in[i]), .clk(clk) , .Re(Re) , .inz(inz), .Q(Q[i]));
36     end
37   endgenerate
38 endmodule
```

```
● ● ●
1 `timescale 1ns/1ns
2 module MSRR8_comp();
3     reg clk, Re, inz;
4     reg [1:0] mode;
5     reg sIn;
6     wire [7:0] Q,Qalways;
7
8     MSRR8_FF MSTB(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Q));
9     MSRR8_alw MALW(.clk(clk), .Re(Re), .inz(inz), .sIn(sIn), .mode(mode), .Q(Qalways));
10
11    initial begin
12        clk = 1'b0;
13        forever begin
14            clk = ~clk; #50;
15        end
16    end
17
18    initial begin
19        mode = 2'b00;
20        sIn = 1'b0;
21        inz = 1'b0;
22        Re = 1'b1;
23        #100;
24
25        Re = 1'b0;
26        #100;
27
28        Re=1'b1; #50;
29        sIn=1'b1; mode=2'b01; #800;
30        sIn=1'b0;
31        Re=1'b0; #100;
32        Re=1'b1;
33        sIn=1; #100;
34
35        mode=2'b10; #850;
36
37        sIn=1'b0; #100;
38        mode=2'b11; #800;
39
40        sIn=1'b1;
41        mode = 2'b01; #800;
42        inz=1'b1; #100;
43        $finish;
44    end
45 endmodule
46
```



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).