به نام خدا

University of Tehran

Electronic and Computer Engineering

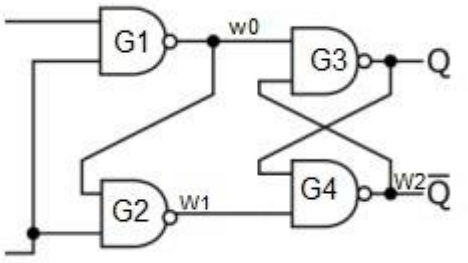
Digital Systems 1

Computer Assignment 4

نام و نام خانوادگی	امیرحسین یاوری خو
شماره دانشجویی	810199514
تاریخ ارسال گزارش	1401/3/8

Q1

Using 4 identical Nand gates, we implement D-latch as following:

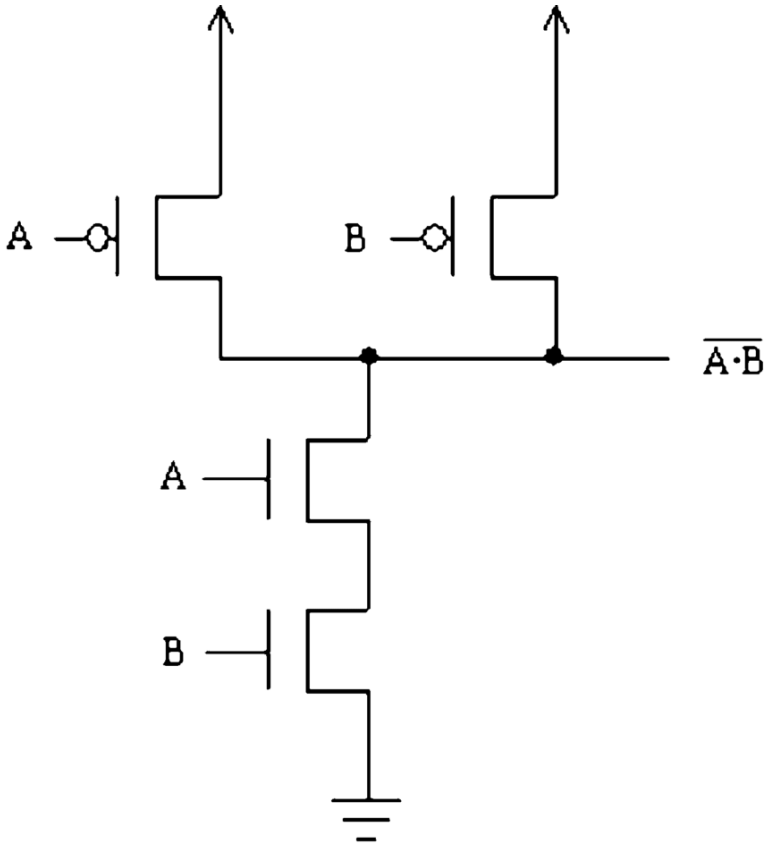



```
1   `timescale 1ns/1ns
2   module memory1 (input D,clk,output Q);
3           wire w0,w1,w2;
4           nand G1(w0,D,clk),G2(w1,w0,clk),G3(Q,w0,w2),G4(w2,w1,Q);
5   endmodule
6
```

Note that numbers in code correspond with design.

Q2

To find out delay of Nand gate we refer to CMOS structure:



To1:

Pull up delay: 6ns

Pull down delay: 8ns

To1 worst case delay:8ns

To0:

Pull up delay:6ns

Pull down delay: 8ns

To0 worst case delay:8ns

So we add #8 delay to the previous code:

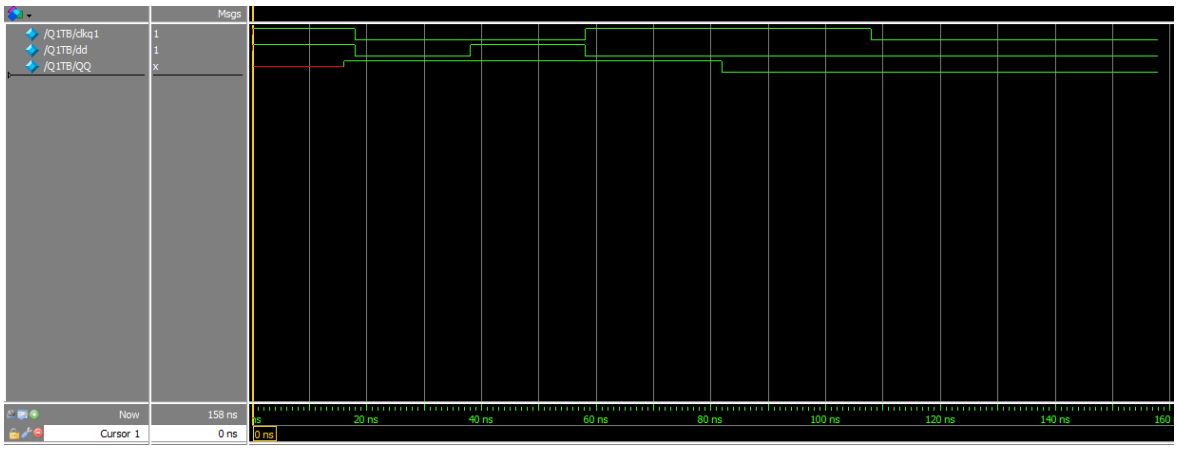
```
1   `timescale 1ns/1ns
2   module memory1 (input D,clk,output Q);
3           wire w0,w1,w2;
4           nand #8 G1(w0,D,clk),G2(w1,w0,clk),G3(Q,w0,w2),G4(w2,w1,Q);
5   endmodule
6
```

Q3

First we use a testbench to see if the memory element works correctly:

```
Ln#
1   `timescale 1ns/1ns
2   /*module clock(output logic w);
3           always
4                   #38 w = ~w;
5                   initial
6                   w = 1;
7   endmodule*/
8
9   module Q1TB();
10          reg dd,clkq1,QQ;
11          //clock myclock(clkq1);
12          memory1 m1(.D(dd),.clk(clkq1),.Q(QQ));
13          initial begin
14          clkq1=1;
15          dd=1;
16          #18 clkq1=0;
17          dd=0;
18          #20 dd=1;
19          #20 dd=0;
20          clkq1=1;
21          #50;
22          clkq1=0;
23          #50;
24          end
25  endmodule
```

The waveform result is shown below:

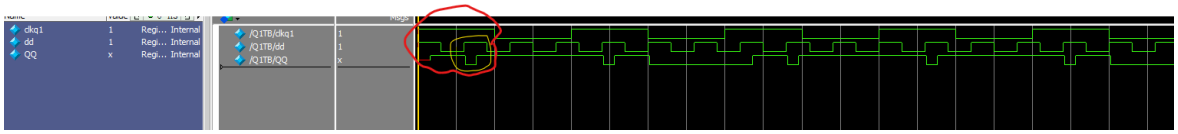


This testbench can confirm under suitable circumstances, the memory element works correctly. Note that this circumstance means the delay for the longest path in memory element(which is 3 NAND gates that equals 24 ns) is considered for the setup and hold time. In this testbench, the condition which both clock and data are 0 isn't tested because it sends the circuit into glitch.

Now for the glitch part, we use the code below:

```
Ln#
 1  `timescale 1ns/1ns
 2  module clock(output logic w);
 3           always
 4                   #100 w = ~w;
 5                   initial
 6                   w = 1;
 7  endmodule
 8
 9  module Q1TB();
10          reg dd,clkq1,QQ;
11          clock myclock(clkq1);
12          memoryl m1(.D(dd),.clk(clkq1),.Q(QQ));
13          initial begin
14          dd=1;
15          repeat (100) #30 dd=~dd;
16          end
17 endmodule
```

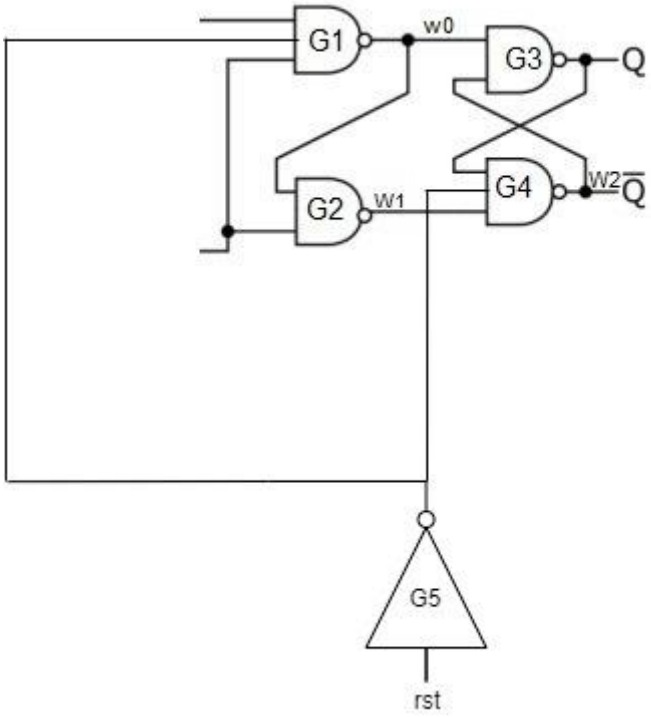
Note that the timing is different from the previous code. As we can see in the waveform below, after the rising edge the output shouldn't change because theoretically the circuit gets input only when clock is raised to 1 but we can see that by changing input while clock is 1, the output changes and this can confirm transparency.



Ps. To simulate this code we should hit break button because the way I described clock signal with always, clock will change infinitely and the simulation results won't show.

Q4

To implement reset function, we should add reset to two gates as following:



This way, when reset is active (1) it forces ~Q to be 1 and w0 to be 1 which makes Q to be 0. When reset is 0, the circuit will function like before and can be used as memory.

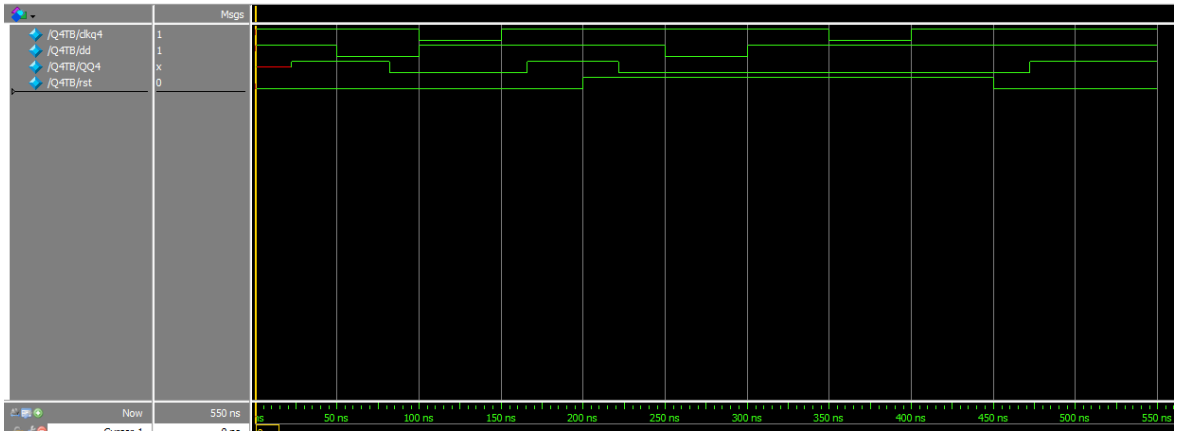
Code for this circuit is shown below:

```
Ln#
1   `timescale 1ns/1ns
2   module memoryrst (input D,clk,rst,output Q);
3           wire w0,w1,w2,w3;
4           nand #8 G1(w0,D,clk,w3),G2(w1,w0,clk),G3(Q,w0,w2),G4(w2,w1,Q,w3);
5           not #6 G5(w3,rst);
6   endmodule
7
```

Testbench to verify the operation of rst:

```
module Q4TB();
        reg dd,clkq4,QQ4,rst;
        //clock myclock4(clkq4);
        memoryrst UUT(.D(dd),.clk(clkq4),.Q(QQ4),.rst(rst));
        initial begin
        rst=0;
        clkq4=1;
        dd=1;
        #50 dd=~dd;
        #50 clkq4=0;
        dd=~dd;
        #50 clkq4=1;
        #50 rst=1;
        #50 dd=0;
        #50 dd=1;
        #50 clkq4=0;
        #50 clkq4=1;
        #50 rst=0;
        #100;
        end
endmodule
```

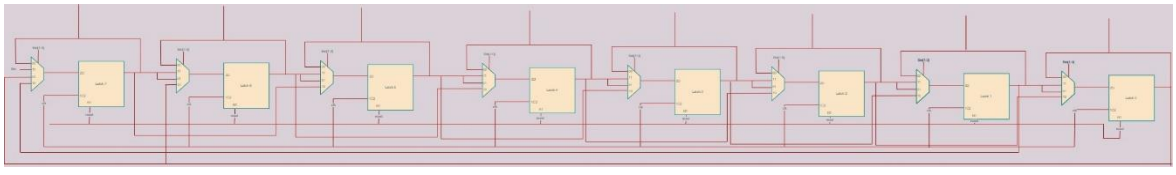
Waveform result:



Waveform confirms the design.

Q5

The main design for this circuit is as following:

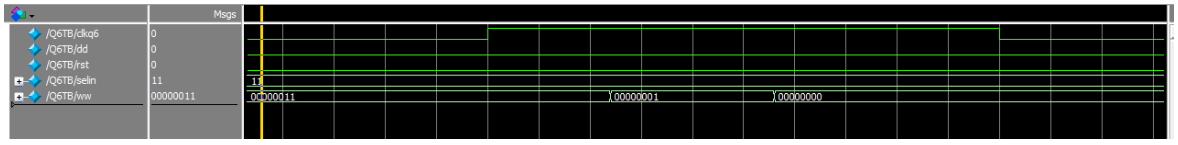


The code for such diagram is shown below:

```
Ln#
 1  `timescale 1ns/1ns
 2  module mux (input a,b,c,d,input[1:0] sel,output w);
 3          assign w=sel[1]?(sel[0]?d:c):(sel[0]?b:a);
 4  endmodule
 5  module msrr8latch(input [1:0]sel,input clk,rst,Sin,output[7:0]Po);
 6          wire [7:0] Pin;
 7          mux mux7(.a(Po[7]),.b(Po[0]),.c(Po[1]),.d(Sin),.w(Pin[7]),.sel(sel));
 8          mux mux6(.a(Po[6]),.b(Po[7]),.c(Po[0]),.d(Po[7]),.w(Pin[6]),.sel(sel));
 9          mux mux5(.a(Po[5]),.b(Po[6]),.c(Po[7]),.d(Po[6]),.w(Pin[5]),.sel(sel));
10          mux mux4(.a(Po[4]),.b(Po[5]),.c(Po[6]),.d(Po[5]),.w(Pin[4]),.sel(sel));
11          mux mux3(.a(Po[3]),.b(Po[4]),.c(Po[5]),.d(Po[4]),.w(Pin[3]),.sel(sel));
12          mux mux2(.a(Po[2]),.b(Po[3]),.c(Po[4]),.d(Po[3]),.w(Pin[2]),.sel(sel));
13          mux mux1(.a(Po[1]),.b(Po[2]),.c(Po[3]),.d(Po[2]),.w(Pin[1]),.sel(sel));
14          mux mux0(.a(Po[0]),.b(Po[1]),.c(Po[2]),.d(Po[1]),.w(Pin[0]),.sel(sel));
15          memoryrst m7(.D(Pin[7]),.clk(clk),.rst(rst),.Q(Po[7]));
16          memoryrst m6(.D(Pin[6]),.clk(clk),.rst(rst),.Q(Po[6]));
17          memoryrst m5(.D(Pin[5]),.clk(clk),.rst(rst),.Q(Po[5]));
18          memoryrst m4(.D(Pin[4]),.clk(clk),.rst(rst),.Q(Po[4]));
19          memoryrst m3(.D(Pin[3]),.clk(clk),.rst(rst),.Q(Po[3]));
20          memoryrst m2(.D(Pin[2]),.clk(clk),.rst(rst),.Q(Po[2]));
21          memoryrst m1(.D(Pin[1]),.clk(clk),.rst(rst),.Q(Po[1]));
22          memoryrst m0(.D(Pin[0]),.clk(clk),.rst(rst),.Q(Po[0]));
23          assign w=Po;
24  endmodule
25
```

Q6

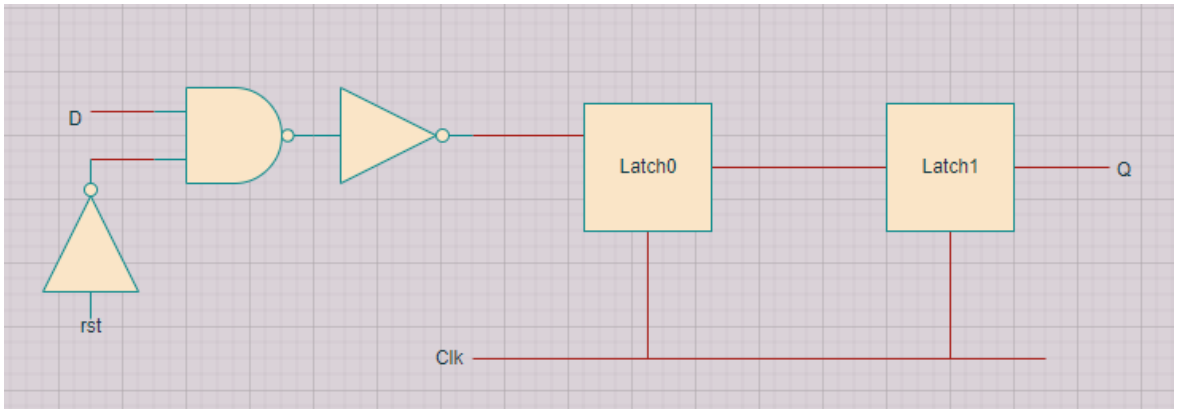
This circuit won't work like we expected. Because we are using latches as memory elements. With latches, we have transparency and that would be a problem when we connect multiple memory elements to each other. Instead, we should use flip flops so that we can build a functional circuit.



We can see that output changes in the middle of the clock. So we can say that we have transparency.

Q7

To build a flip flop, we connect two D latches to each other.



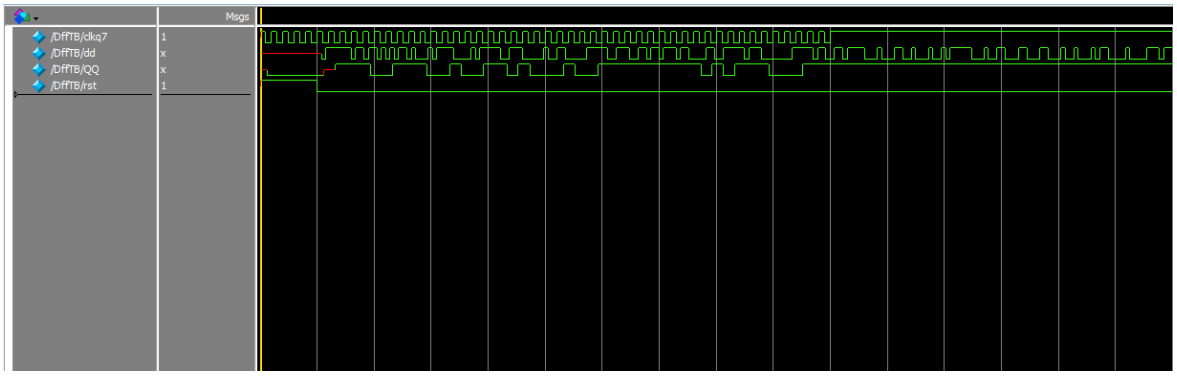
The code for this diagram is shown below:

```
Ln#
1  `timescale 1ns/1ns
2  module Dff (input D,clk,rst,output Qout);
3          wire L0,L1,L2,L3;
4          nand #8 G2(L2,D,L4);
5          not #6 G3(L0,L2),G1(L3,clk),G4(L4,rst);
6          memory1 m1(.D(L0),.clk(clk),.Q(L1));
7          memory1 m2(.D(L1),.clk(L3),.Q(Qout));
8  endmodule
9
```

Testbench for verifying operation of this memory:

```
Ln#
1  `timescale 1ns/1ns
2  module DffTB();
3          reg dd,clkq7,rst,QQ;
4          Dff UUT(.D(dd),.rst(rst),.clk(clkq7),.Qout(QQ));
5          initial begin
6          clkq7=1;
7          repeat (100) #100 clkq7=~clkq7;
8          end
9          initial begin
10         $monitor("monitor clk:%b rst:%b d:%b Q=%b",clkq7,rst,dd,QQ);
11         rst=1;
12         #1000 rst=0;
13         repeat (200) #75 dd=$random;
14         end
15 endmodule
16
```

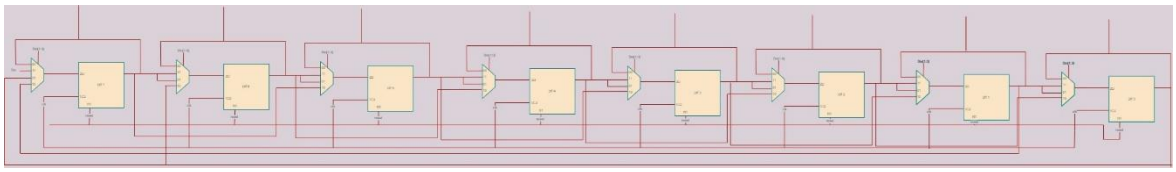
Waveform for previous testbench:



We can see edge triggering and reset works as intended.

Q8

This design is the same design previously mentioned in Q5. The only difference is that instead of latches, we use flip flops.



Code for this design is shown below:

```
Ln#
 1  `timescale 1ns/1ns
 2  module mux (input a,b,c,d,input[1:0] sel,output w);
 3          assign w=sel[1]?(sel[0]?d:c):(sel[0]?b:a);
 4  endmodule
 5  module msrr8dff(input [1:0]sel,input clk,rst,Sin,output[7:0]Po);
 6          wire [7:0] Pin;
 7          genvar i,k;
 8                  generate
 9                          for(i=0;i<6;i=i+1) begin:mm
10                                  mux mr(.a(Po[i]),.b(Po[i+1]),.c(Po[i+2]),.d(Po[i+1]),.w(Pin[i]),.sel(sel));
11                          end
12                          for (k=0;k<6;k=k+1) begin:ff
13                                  Dff dr(.D(Pin[k]),.clk(clk),.rst(rst),.Qout(Po[k]));
14                          end
15                  endgenerate
16          mux mux7(.a(Po[7]),.b(Po[0]),.c(Po[1]),.d(Sin),.w(Pin[7]),.sel(sel));
17          mux mux6(.a(Po[6]),.b(Po[7]),.c(Po[0]),.d(Po[7]),.w(Pin[6]),.sel(sel));
18          Dff m7(.D(Pin[7]),.clk(clk),.rst(rst),.Qout(Po[7]));
19          Dff m6(.D(Pin[6]),.clk(clk),.rst(rst),.Qout(Po[6]));
20  endmodule
21
```

With this testbench we can verify operations of the component.

```
Ln#
 1   `timescale 1ns/1ns
 2   module Q8TB();
 3           reg dd,clkq8,rst;
 4           reg [7:0] ww;
 5           reg[1:0] selin;
 6           msrr8dff UUT(.sel(selin),.Sin(dd),.clk(clkq8),.rst(rst),.Po(ww));
 7           initial begin
 8           clkq8=1;
 9           repeat (500) #200 clkq8=~clkq8;
10           end
11           initial begin
12           repeat (50) #1000 rst=$random;
13           end
14           initial begin
15           repeat (500) #182 selin=$random;
16           end
17           initial begin
18           $monitor("monitor clk:%b rst:%b d:%b Po: %b",clkq8,rst,dd,ww);
19           repeat (800) #132 dd=$random;
20           end
21  endmodule
22
```

Q9

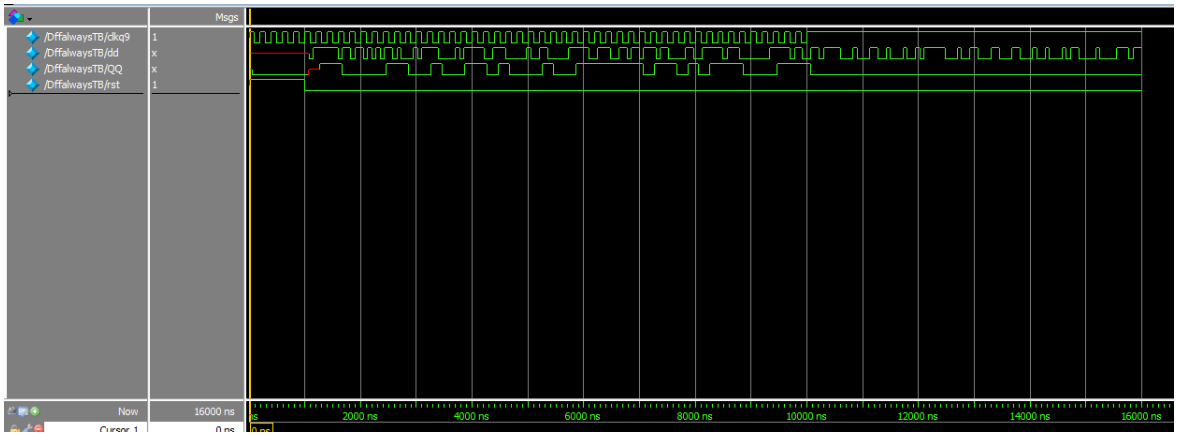
To define positive edge trigger, we use posedge in the code. If reset is 1, Q will be 0 otherwise Q will save data.

Screenshot of the code is shown below:

```
Ln#
 1   `timescale 1ns/1ns
 2   module Dffalways(input D,rst,clk,output reg Q);
 3           always @(posedge clk) begin
 4                   if(rst) #64 Q<=1'b0;
 5                   else #64 Q<=D;
 6           end
 7   endmodule
 8
```

We will use the same testbench code from Q7 to compare the results.

```
Ln#
 1  `timescale 1ns/1ns
 2  module DffalwaysTB();
 3          reg dd,clkq9,rst,QQ;
 4          Dffalways UUT(.D(dd),.rst(rst),.clk(clkq9),.Q(QQ));
 5          initial begin
 6          clkq9=1;
 7          repeat (100) #100 clkq9=~clkq9;
 8          end
 9          initial begin
10          $monitor("monitor clk:%b rst:%b d:%b Q=%b",clkq9,rst,dd,QQ);
11          rst=1;
12          #1000 rst=0;
13          repeat (200) #75 dd=$random;
14          end
15  endmodule
16
```

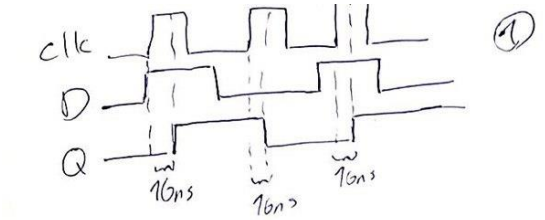


We can see that both circuits work in a similar way.

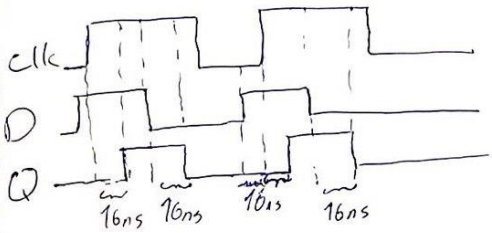
Q10

When we define flip flops with always, simulation will run much faster because it takes time to generate and propagate gates and finding out the way the circuit works.

Hand simulation:

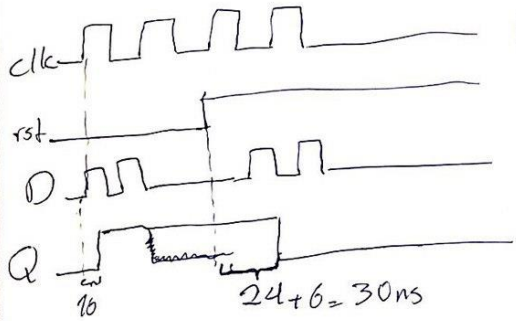


①



③

while clock is 1, data changes and after delay glitch happens.



④