به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

سیستم‌های دیجیتال 1

استاد: دکتر نوابی

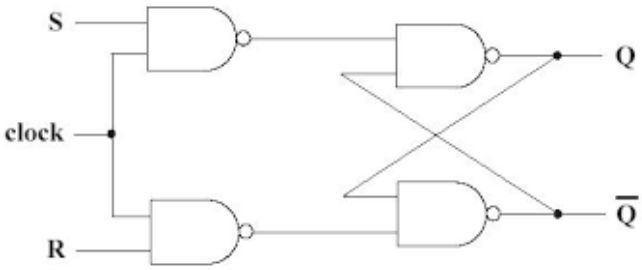
تمرین کامپیوتری شماره 4

فربد سیاه‌کلی

شماره دانشجویی: 810198510

فروردین 1400

1) شکل مدار ساخته شده که در ورودی‌ها نیز با استفاده از گیت nand، از یک اینورتر برای active low کردن آنها استفاده شده:



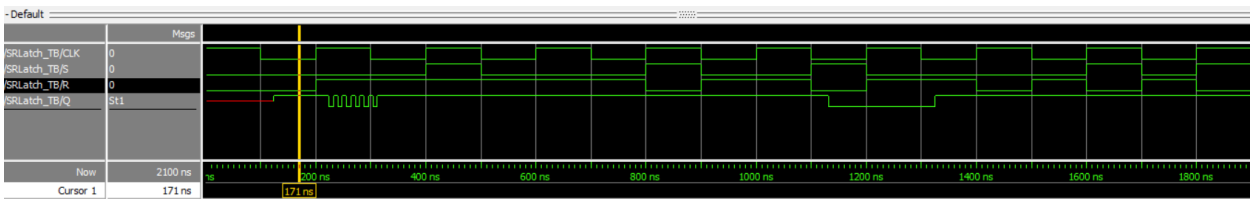
کد مربوطه:

```
`timescale 1ns/1ns
module SRLatch(input S, R, CLK, output Q, Qbar);
 wire x, y, CLKbar, Rbar, Sbar;
 nand #8 (CLKbar, CLK, CLK);
 nand #8 (Rbar, R, R);
 nand #8 (Sbar, S, S);
 nand #8 (x, Sbar, CLKbar);
 nand #8 (y, Rbar, CLKbar);
 nand #8 (Qbar, Q, y);
 nand #8 (Q, Qbar, x);
endmodule
```

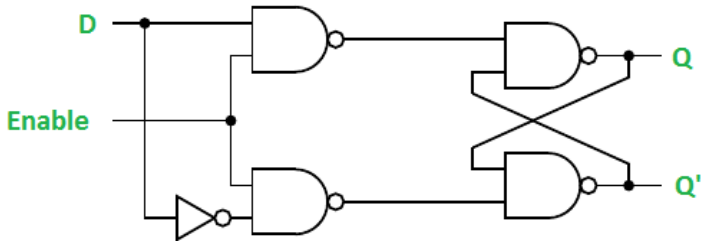
2) تست بنچ مربوطه:

```
`timescale 1ns/1ns
module SRLatch_TB();
  logic CLK = 1;
  logic S = 0, R = 0;
  wire Qb, Q;
  integer i;
  SRLatch UUT(S, R, CLK, Q, Qb);
  always #100 CLK <= ~CLK;
    initial begin
    for(i=0; i<20; i=i+1) begin
         #100 {S,R} = $random();
    end
    #100 $stop;
  end
endmodule
```

شکل موج حاصله از تست بنچ که در حالت صفر بودن هر دو ورودی (که با توجه به active low بودن) memory loss مشاهده می‌شود:



3) مدار D Latch ساخته شده یا استفاده از SR Latch بخش قبل که تنها لام است اینورت شده S را به R وصل کنیم:



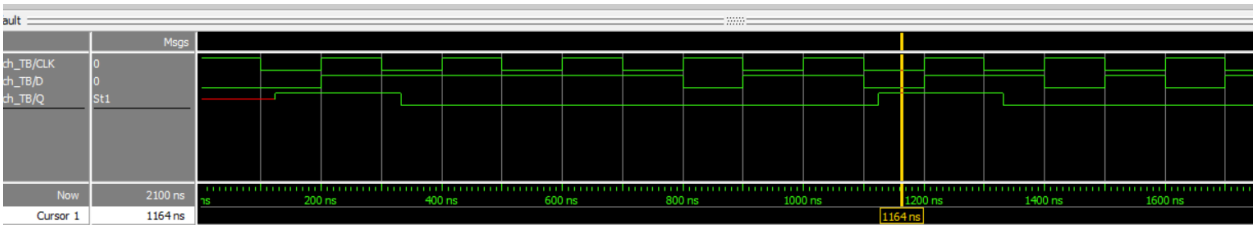
کد ماژول برای مدار:

```
module DLatch(input D, CLK, output Q, Qbar);
 wire Dbar;
 not #6 (Dbar, D);
 SRLatch SR(D, Dbar, CLK, Q, Qbar);
endmodule
```

تست بنچ مربوطه که در آن با یک حلقه مقدار D را به صورت رندوم تغییر دادیم:

```
`timescale 1ns/1ns
module DLatch_TB();
  logic CLK = 1;
  logic D = 0;
  wire Qb, Q;
  integer i;
  DLatch UUT(D, CLK, Q, Qb);
  always #100 CLK <= ~CLK;
    initial begin
    for(i=0; i<20; i = i+1) begin
         #100 D = $random();
    end
    #100 $stop;
  end
endmodule
```

شکل موج خروجی که چند مثال را نشان می‌دهد:



4) مدار shift register ساخته شده با cascade کردن D latch:

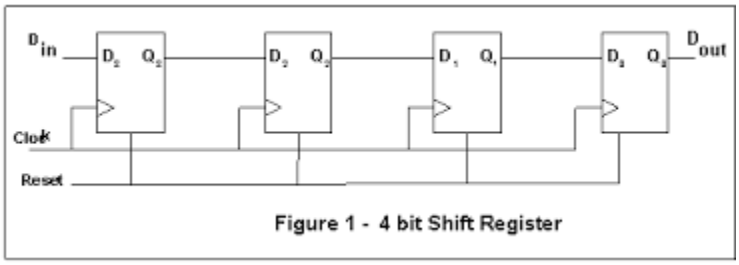


Figure 1 - 4 bit Shift Register

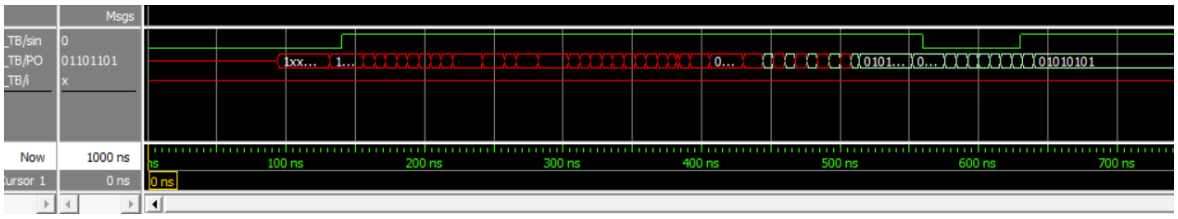
کد مربوطه:

```
module Shift_Rg_Latch(input sin, CLK, output [7:0] PO);
  wire [7:0] w;
  DLatch D1(sin, CLK, PO[7], w[0]);
  genvar i;
  generate
    for(i = 7; i >= 0; i = i-1) begin
       DLatch Di(PO[i], CLK, PO[i-1], w[0]);
    end
  endgenerate
endmodule
```

5) تست بنچ شیفت رجیستر ساخته شده با D latch:

```
`timescale 1ns/1ns
module ShifRegister_Latch_TB();
  logic sin = 0 , CLK = 0;
  wire [7:0] PO;
  Shift_Rg_Latch UUT(sin, CLK, PO);
  always #80 CLK = ~CLK;
  initial begin
    #600 $stop;
  end
endmodule
```

بخشی از شکل موج:



در این مدار در لحظه‌ای که کلاک منفی می‌شود، مقادیر از D latch ها عبور می‌کنند و سریعا از D latch بعدی نیز می‌گذرند و تا زمانی که کلاک مثبت شود این اتفاق روی می‌دهد. در نتیجه خروجی درستی گرفته نمی‌شود.

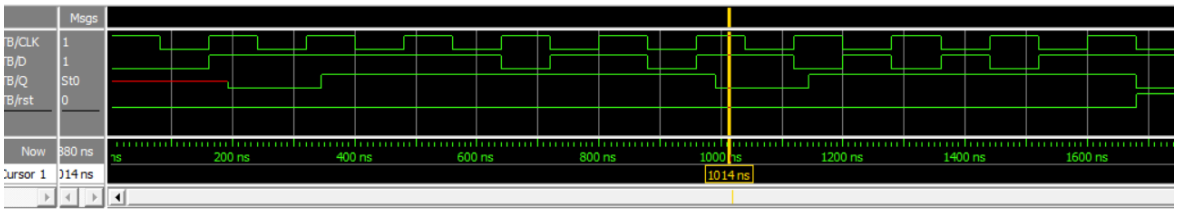
6 و 7) ساخت یک master slave D flip flop که ورودی reset نیز دارد:

```
module D_FlipFlop(input D, CLK, rst, output logic Q, Qbar);
  wire QM, QbarM;
  DLatch Master(D, CLK, QM, QbarM);
  DLatch Slave(QM, ~CLK, Q, Qbar);
  assign Q = ~rst & Q;
  assign Qbar = ~Q;
endmodule
```

تست بنچ مربوط:

```
`timescale 1ns/1ns
module FF_TB();
  logic CLK = 1;
  logic D = 0;
  wire Qb, Q;
  integer i;
  logic rst = 0;
  D_FlipFlop UUT(D, CLK, rst, Q, Qb);
  always #80 CLK <= ~CLK;
  initial begin
    for(i=0; i<20; i = i+1) begin
        #80 D = $random();
    end
    #80 rst = 1;
    #80 $stop;
  end
endmodule
```

خروجی تست بنچ:



8) ساخت شیفت رجیستر اینبار با استفاده از D Flip Flop:

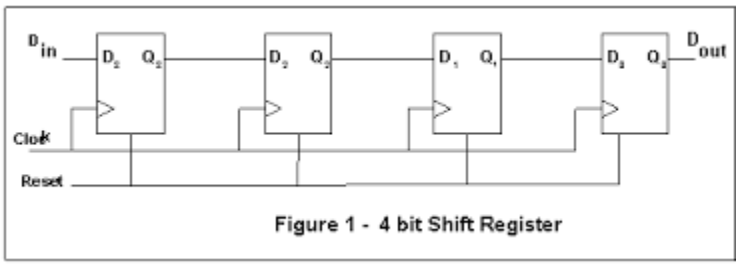


Figure 1 - 4 bit Shift Register

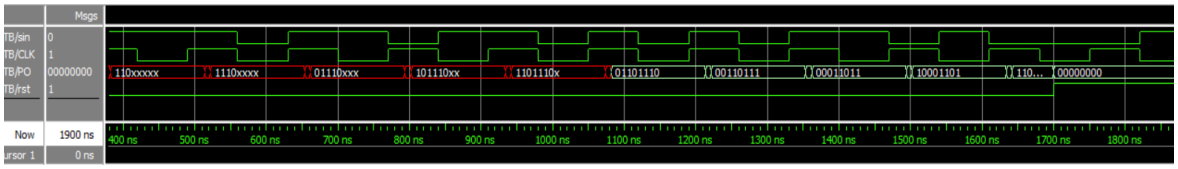
کد مربوطه:

```
module Shift_Rg_FF(input sin, CLK, rst, output reg [7 : 0] PO);
  wire [7:0] w;
  D_FlipFlop D1(sin, CLK, rst ,PO[7], w[0]);
  genvar i;
  generate
     for(i = 7; i >= 0; i = i-1) begin
        D_FlipFlop Di(PO[i], CLK, rst, PO[i-1], w[0]);
     end
  endgenerate
endmodule
```

تست بنچ مربوطه:

```
`timescale 1ns/1ns
module ShifRegister_FF_TB();
  logic sin = 0 , CLK = 0;
  wire [7:0] PO;
  logic rst = 0;
  Shift_Rg_FF UUT(sin, CLK, rst, PO);
  always #70 CLK = ~CLK;
  always #70 sin = $random;;
  initial begin
    #1700 rst = 1;
    #200 $stop;
  end
endmodule
```

خروجی تست بنچ که در آن در ثانیه 1700 ریست یک شده است:

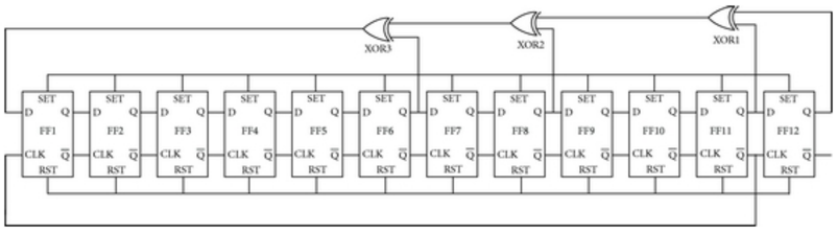


9) کد شیفت رجیستر اینبار با استفاده از always و به صورت procedural:

برای حل بخش بعد نیاز به PI بود تا مقادیر اولیه‌ای به سیم‌ها داده شوند و در ابتدا مقدارشان x نباشد.

```
module Shift_Rg_Always(input sin, CLK, input reg [7:0] PI, output reg [7 : 0] PO);
  reg [7:0] temp;
  integer count = 0;
  always@(negedge CLK) begin
    if (count == 0) begin
       temp <= PI;
    end
    else begin
       temp <= PO;
    end
    count = count + 1;
  end
  assign PO = {sin, temp[7:1]};
endmodule
```

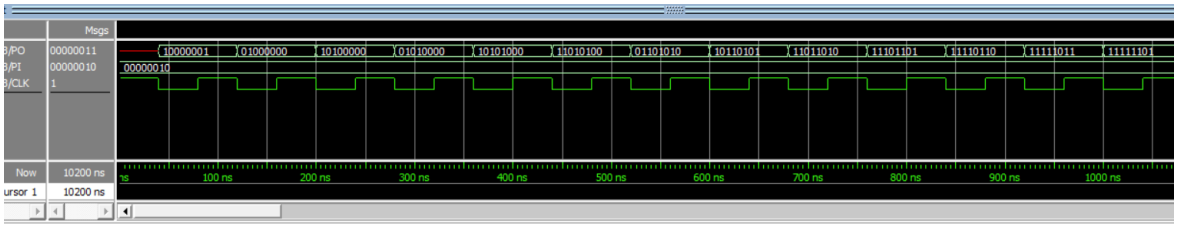
10) شکل مدار:



کد LFSR برای $x^8 + x^7 + x^6 + x^3 + 1$ که در آن عملا خروجی‌های صفر، یک، دو و پنج باهم xor (جمع) می‌شوند . بعنوان سریال ورودی از سمت چپ وارد می‌شوند:

```
module LFSR(input CLK, input reg [7:0] PI, output reg [7 : 0] PO);
  wire Sin;
  assign Sin = PO[0]^PO[1]^PO[2]^PO[5];
  Shift_Rg_Always SR1(Sin, CLK, PI, PO);
endmodule
```

خروجی کد:



این خروجی‌ها با تناوب 2^{n-1} تکرار می‌شود که در اینجا $n = 8$ است. در نتیجه تناوب هر 128 بار کلاک رخ می‌دهد.