

به نام خدا
درس: طراحی سیستم های دیجیتال
نیم سال تحصیلی دوم ۹۹۰۰
مدرس: بهاروند

تمرین شماره: ۴		موعده تحویل: ۱۴۰۰/۰۲/۲۰ ساعت ۲۳:۵۵
نام و نام خانوادگی:	سینا مظاهری	شماره: ۹۸۱۷۱۱۵۹
		دانشجویی:

راه حل در همین فایل ارائه شود.

فایل به PDF تبدیل و در سایت بارگذاری شود.

**عنوان
تمرین**

- یک مدار LFSR با مشخصات زیر طراحی کنید:
- (۱) شیفت رجیستر به صورت پارامتری طراحی شود. یعنی تعداد بیت های آن توسط پارامتری در ابتدای ماجول تعریف شود.
 - (۲) شیفت به سمت راست صورت می گیرد.
 - (۳) بیت کم ارزش آن دارای اندیس ۱ و بیت پر ارزش آن دارای ارزش n است که n تعداد بیت های شیفت رجیستر است.
 - (۴) با استفاده از generate کد را به نحوی بنویسید که بتواند دو چند جمله ای زیر را بسته به مقدار n پیاده سازی نماید. مقدار n نمایانگر نوع و تعداد بیت های LFSR است.
- $$x^8 + x^6 + x^5 + x^1 + 1$$
- $$x^{17} + x^3 + 1$$
- (۵) برای LFSR یک ریست آسنکرون به صورت active low در نظر بگیرید.
 - (۶) پریود کلاک را ۱۰ در نظر بگیرید و کلاک active high است.
 - (۷) در testbench مقدار اولیه LFSR با استفاده سیگنال load_seed و به صورت تصادفی (random) بارگذاری شود.
 - (۸) در testbench مقادیر محتوای LFSR را پس از بارگذاری مقدار seed و شروع به کار مدار، چاپ کنید. خروجی تقریباً شبیه شکل زیر مد نظر است. خروجی یک بیتی LFSR در این تمرین، همان بیت کم ارزش (سمت راست) است.

```

4 -----
5 LFSR WIDTH = 17
6 The current module name is = lfsr_test
7 Clock period is: 10 in the mentioned time unit
8 Clock frequency is: 100 MHz
9 Simulation started...
10
11 @ 0.0000 ns: feedback = 0, lfsr_reg = 00000000000000000, lfsr_out = 0,
12 @ 35.0000 ns: feedback = 0, lfsr_reg = 00000000000000000, lfsr_out = 0,
13 @ 55.0000 ns: feedback = 1, lfsr_reg = 10000000000000000, lfsr_out = 0,
14 @ 65.0000 ns: feedback = 1, lfsr_reg = 11000000000000000, lfsr_out = 0,
15 @ 75.0000 ns: feedback = 0, lfsr_reg = 01100000000000000, lfsr_out = 0,
16 @ 85.0000 ns: feedback = 0, lfsr_reg = 00110000000000000, lfsr_out = 0,
17 @ 95.0000 ns: feedback = 1, lfsr_reg = 10011000000000000, lfsr_out = 0,
18 @ 105.0000 ns: feedback = 1, lfsr_reg = 11001100000000000, lfsr_out = 0,
19 @ 115.0000 ns: feedback = 1, lfsr_reg = 11100110000000000, lfsr_out = 0,
20 @ 125.0000 ns: feedback = 0, lfsr_reg = 01110011000000000, lfsr_out = 0,
21 @ 135.0000 ns: feedback = 0, lfsr_reg = 00111001100000000, lfsr_out = 0,
22 @ 145.0000 ns: feedback = 1, lfsr_reg = 10011100110000000, lfsr_out = 0,
23 @ 155.0000 ns: feedback = 0, lfsr_reg = 01001110011000000, lfsr_out = 0,
24 @ 165.0000 ns: feedback = 0, lfsr_reg = 00100111001100000, lfsr_out = 0,
25 @ 175.0000 ns: feedback = 0, lfsr_reg = 00010011100110000, lfsr_out = 0,
26 @ 185.0000 ns: feedback = 0, lfsr_reg = 00001001110011000, lfsr_out = 0,
27 @ 195.0000 ns: feedback = 1, lfsr_reg = 00000100111001100, lfsr_out = 0,
28 @ 205.0000 ns: feedback = 0, lfsr_reg = 10000010011100110, lfsr_out = 0,
29 @ 215.0000 ns: feedback = 0, lfsr_reg = 01000001001110011, lfsr_out = 1,
30 @ 225.0000 ns: feedback = 1, lfsr_reg = 10100000100111001, lfsr_out = 1,
31 @ 235.0000 ns: feedback = 1, lfsr_reg = 01010000010011100, lfsr_out = 0,
32 @ 245.0000 ns: feedback = 0, lfsr_reg = 10101000001001110, lfsr_out = 0,
33 @ 255.0000 ns: feedback = 1, lfsr_reg = 01010100000100111, lfsr_out = 1,
34 @ 265.0000 ns: feedback = 0, lfsr_reg = 00101010000010011, lfsr_out = 1,
35 @ 275.0000 ns: feedback = 1, lfsr_reg = 10010101000001001, lfsr_out = 1,
36 @ 285.0000 ns: feedback = 1, lfsr_reg = 01001010100000100, lfsr_out = 0,
37 @ 295.0000 ns: feedback = 0, lfsr_reg = 00100101010000010, lfsr_out = 0,

```

برای شروع طراحی ابتدا یک ماژول به نام **lfsr** در نظر می گیریم. این ماژول شامل سه ورودی می باشد. دو ورودی برای **reset** و **clk** و نیز برای **seed_value** که مقدار تصادفی مورد نظر را مادامی که لبه مثبت کلاک می آید و سیگنال **load_seed** فعال می باشد، بارگذاری می کند. بعد از آن متناسب با پارامتر انتخاب شده (تعداد بیت های **LFSR**)، در بلوک **generate** معادله ی مورد نظر انتخاب شده و سیگنال **feedback_value** را در سطح **Data flow**، مقدار دهی می کند. از آن جایی که این مدار یک مدار ترتیبی می باشد، ۳ بلوک **always** مجزا در نظر گرفته شده تا به ترتیب مدار ترکیبی حالت بعد، گذر حالت و مدار ترکیبی خروجی را نشان دهد. دو حالت به نام **current_state** و **next_state** در نظر می گیریم که به ترتیب حالت فعلی و بعدی را نشان می دهد. در بلوک **always** اول، با تغییر حالت و ورودی **feedback_value**، مقدار ثبات **current_state** را از بیت با بیشترین ارزش تا بیت دوم در خانه های متناظر $N - 1$ تا یکم می ریزیم و سپس مقدار **feedback_value** را در جایگاه **N** ام قرار می دهیم. بدین ترتیب یک واحد به سمت راست مقدار ثبات شیفتمی می خورد. همانطور که در عکس زیر مشخص است، با ارزش ترین بیت اندیس **N** و کم ارزش ترین آن اندیس ۱ را دارد.

```
module lfsr #(parameter N = 4)
(
    input wire [N:1] seed_value,
    input wire clk, reset, load_seed,
    output reg [N:1] output_value
);

wire feedback_value;
reg [N:1] current_state, next_state;
generate
    begin
        case (N)
            2: assign feedback_value = current_state[2] ^ current_state[1];
            3: assign feedback_value = current_state[3] ^ current_state[2];
            4: assign feedback_value = current_state[4] ^ current_state[3];
            5: assign feedback_value = current_state[5] ^ current_state[3];
            6: assign feedback_value = current_state[6] ^ current_state[5];
            7: assign feedback_value = current_state[7] ^ current_state[6];
            8: assign feedback_value = current_state[8] ^ current_state[6] ^ current_state[5] ^ current_state[1];
            9: assign feedback_value = current_state[9] ^ current_state[5];
            10: assign feedback_value = current_state[10] ^ current_state[7];
            11: assign feedback_value = current_state[11] ^ current_state[9];
            12: assign feedback_value = current_state[12] ^ current_state[11] ^ current_state[10] ^ current_state[4];
            13: assign feedback_value = current_state[13] ^ current_state[12] ^ current_state[11] ^ current_state[8];
            14: assign feedback_value = current_state[14] ^ current_state[13] ^ current_state[12] ^ current_state[2];
            15: assign feedback_value = current_state[15] ^ current_state[14];
            16: assign feedback_value = current_state[16] ^ current_state[15] ^ current_state[13] ^ current_state[4];
            17: assign feedback_value = current_state[17] ^ current_state[3]; // considered
            18: assign feedback_value = current_state[18] ^ current_state[11];
            19: assign feedback_value = current_state[19] ^ current_state[18] ^ current_state[17] ^ current_state[14];
            20: assign feedback_value = current_state[20] ^ current_state[17];
            21: assign feedback_value = current_state[21] ^ current_state[19];
            22: assign feedback_value = current_state[22] ^ current_state[21];
            23: assign feedback_value = current_state[23] ^ current_state[18];
            24: assign feedback_value = current_state[24] ^ current_state[23] ^ current_state[22] ^ current_state[17];
            default: assign feedback_value = 1'b0;
        endcase
    end
endgenerate
```

حال به سراغ بلوک دوم می رویم. این بلوک **always** دوم زمانی فعال می شود که سیگنال **reset** به شکل آسنکرون روی لبه ی پایین رونده باشد و یا سیگنال کلاک روی لبه ی بالا رونده خود باشد. در این صورت شرایط داخل بلوک بررسی می شود. ابتدا در صورتی که سیگنال **reset** ، صفر شده باشد مقدار ۰ را در ثبات بارگذاری می کنیم. حال در صورتی که روی لبه مثبت باشیم و سیگنال **load_seed** فعال باشد، مقدار **seed_value** بر روی ثبات بارگذاری می شود و در غیر اینصورت به حالت بعدی می رویم. خروجی نیز در بلوک **always** سوم مشخص می شود و همان مقدار ثبات را می گیرد.

```
always @(current_state or feedback_value)
begin
    next_state[N] = feedback_value;
    next_state[N - 1:1] = current_state[N:2];
end
always @(posedge clk or negedge reset)
begin
    if (!reset)
        current_state <= 0;
    else if (load_seed)
        current_state <= seed_value;
    else
        current_state <= next_state;
end

always @(current_state)
    output_value = current_state;

endmodule
```

به سراغ **testbench** می رویم. در این **testbench** سیگنال کلاک با دوره زمانی **10ns** در نظر گرفته شده تا مدار مطابق آن چه که خواسته شده کار کند. در ابتدا یک نمونه از مازول می سازیم و سپس سیگنال های مورد نظر را به آن متصل می کنیم و در نهایت نیز با صفر کردن ریست، مدار را ریست می کنیم و مقدار **load_seed** را مشخص می کنیم. ۲ نانو ثانیه بعد مقدار سیگنال **seed_value** را فعال و منتظر بارگذاری ثبات با لبه مثبت کلاک می مانیم و بعد از آن سیگنال **seed_value** را صفر می کنیم. نتایج این شبیه سازی در خروجی کنسول و شکل موج ، برای دو آزمایش ۸ بیتی و ۱۷ بیتی آمده است:

کد **testbench** :

```

module tb(
);
reg clk, reset, load_seed;
reg [17:1] seed_value;
wire [17:1] output_value;

lfsr #(17) lfsr0(.reset(reset), .clk(clk), .output_value(output_value), .seed_value(seed_value), .load_seed(load_seed));

initial
begin
    clk = 1'b0;
    forever #5 clk = ~clk;
end

initial
begin
    seed_value = 17'b10000000000000000;
    load_seed = 1'b0;
    reset = 1'b0;
    #2 reset = 1'b1;
    load_seed = 1'b1;
    #4 load_seed = 1'b0;
end

initial
$monitor($time, " ns feedback = %b, lfsr_reg = %b, lfsr_out = %b", output_value[17], output_value, output_value[1]);

endmodule

```

نتیجه خروجی کنسول و شکل موج برای ۱۷ بیت:

```

0 ns feedback = 0, lfsr_reg = 00000000000000000, lfsr_out = 0
5 ns feedback = 1, lfsr_reg = 10000000000000000, lfsr_out = 0
15 ns feedback = 1, lfsr_reg = 11000000000000000, lfsr_out = 0
25 ns feedback = 1, lfsr_reg = 11100000000000000, lfsr_out = 0
35 ns feedback = 1, lfsr_reg = 11110000000000000, lfsr_out = 0
45 ns feedback = 1, lfsr_reg = 11111000000000000, lfsr_out = 0
55 ns feedback = 1, lfsr_reg = 11111100000000000, lfsr_out = 0
65 ns feedback = 1, lfsr_reg = 11111110000000000, lfsr_out = 0
75 ns feedback = 1, lfsr_reg = 11111111000000000, lfsr_out = 0
85 ns feedback = 1, lfsr_reg = 11111111100000000, lfsr_out = 0
95 ns feedback = 1, lfsr_reg = 11111111110000000, lfsr_out = 0
105 ns feedback = 1, lfsr_reg = 11111111111000000, lfsr_out = 0
115 ns feedback = 1, lfsr_reg = 11111111111100000, lfsr_out = 0
125 ns feedback = 1, lfsr_reg = 11111111111110000, lfsr_out = 0
135 ns feedback = 1, lfsr_reg = 11111111111111000, lfsr_out = 0
145 ns feedback = 1, lfsr_reg = 11111111111111100, lfsr_out = 0
155 ns feedback = 0, lfsr_reg = 01111111111111110, lfsr_out = 0
165 ns feedback = 1, lfsr_reg = 10111111111111111, lfsr_out = 1
175 ns feedback = 0, lfsr_reg = 01011111111111111, lfsr_out = 1
185 ns feedback = 1, lfsr_reg = 10101111111111111, lfsr_out = 1
195 ns feedback = 0, lfsr_reg = 01010111111111111, lfsr_out = 1
205 ns feedback = 1, lfsr_reg = 10101011111111111, lfsr_out = 1
215 ns feedback = 0, lfsr_reg = 01010101111111111, lfsr_out = 1
225 ns feedback = 1, lfsr_reg = 10101010111111111, lfsr_out = 1
235 ns feedback = 0, lfsr_reg = 01010101011111111, lfsr_out = 1
245 ns feedback = 1, lfsr_reg = 10101010101111111, lfsr_out = 1
255 ns feedback = 0, lfsr_reg = 01010101010111111, lfsr_out = 1
265 ns feedback = 1, lfsr_reg = 10101010101011111, lfsr_out = 1
275 ns feedback = 0, lfsr_reg = 01010101010101111, lfsr_out = 1
285 ns feedback = 1, lfsr_reg = 10101010101010111, lfsr_out = 1
295 ns feedback = 0, lfsr_reg = 01010101010101011, lfsr_out = 1
305 ns feedback = 0, lfsr_reg = 00101010101010101, lfsr_out = 1
315 ns feedback = 1, lfsr_reg = 10010101010101010, lfsr_out = 0
325 ns feedback = 1, lfsr_reg = 11001010101010101, lfsr_out = 1
335 ns feedback = 0, lfsr_reg = 01100101010101010, lfsr_out = 0
345 ns feedback = 0, lfsr_reg = 00110010101010101, lfsr_out = 1
355 ns feedback = 1, lfsr_reg = 10011001010101010, lfsr_out = 0

```


run 1000ns

```
0 ns feedback = 0, lfsr_reg = 00000000, lfsr_out = 0
5 ns feedback = 1, lfsr_reg = 10000000, lfsr_out = 0
15 ns feedback = 1, lfsr_reg = 11000000, lfsr_out = 0
25 ns feedback = 1, lfsr_reg = 11100000, lfsr_out = 0
35 ns feedback = 0, lfsr_reg = 01110000, lfsr_out = 0
45 ns feedback = 0, lfsr_reg = 00111000, lfsr_out = 0
55 ns feedback = 0, lfsr_reg = 00011100, lfsr_out = 0
65 ns feedback = 1, lfsr_reg = 10001110, lfsr_out = 0
75 ns feedback = 1, lfsr_reg = 11000111, lfsr_out = 1
85 ns feedback = 0, lfsr_reg = 01100011, lfsr_out = 1
95 ns feedback = 0, lfsr_reg = 00110001, lfsr_out = 1
105 ns feedback = 1, lfsr_reg = 10011000, lfsr_out = 0
115 ns feedback = 0, lfsr_reg = 01001100, lfsr_out = 0
125 ns feedback = 0, lfsr_reg = 00100110, lfsr_out = 0
135 ns feedback = 1, lfsr_reg = 10010011, lfsr_out = 1
145 ns feedback = 1, lfsr_reg = 11001001, lfsr_out = 1
155 ns feedback = 0, lfsr_reg = 01100100, lfsr_out = 0
165 ns feedback = 1, lfsr_reg = 10110010, lfsr_out = 0
175 ns feedback = 1, lfsr_reg = 11011001, lfsr_out = 1
185 ns feedback = 1, lfsr_reg = 11101100, lfsr_out = 0
195 ns feedback = 0, lfsr_reg = 01110110, lfsr_out = 0
205 ns feedback = 0, lfsr_reg = 00111011, lfsr_out = 1
215 ns feedback = 1, lfsr_reg = 10011101, lfsr_out = 1
225 ns feedback = 1, lfsr_reg = 11001110, lfsr_out = 0
235 ns feedback = 1, lfsr_reg = 11100111, lfsr_out = 1
245 ns feedback = 1, lfsr_reg = 11110011, lfsr_out = 1
255 ns feedback = 0, lfsr_reg = 01111001, lfsr_out = 1
265 ns feedback = 1, lfsr_reg = 10111100, lfsr_out = 0
275 ns feedback = 1, lfsr_reg = 11011110, lfsr_out = 0
285 ns feedback = 0, lfsr_reg = 01101111, lfsr_out = 1
295 ns feedback = 0, lfsr_reg = 00110111, lfsr_out = 1
305 ns feedback = 1, lfsr_reg = 10011011, lfsr_out = 1
315 ns feedback = 1, lfsr_reg = 11001101, lfsr_out = 1
325 ns feedback = 0, lfsr_reg = 01100110, lfsr_out = 0
335 ns feedback = 1, lfsr_reg = 10110011, lfsr_out = 1
345 ns feedback = 0, lfsr_reg = 01011001, lfsr_out = 1
```
