




2020-9-30

## 数字电路实验六 寄存器

秦嘉余

191220088

1348288404@qq.com



# 目录

- 1.实验目的 .....1
- 2.实验原理 .....1
- 3.实验环境/器材 .....2
- 4.算术移位和逻辑移位寄存器 .....2
  - 1.设计思路 .....2
  - 2. 实验代码.....2
  - 3.RTL 视图 .....4
  - 4.引脚分配图 .....4
  - 5.实验结果 .....5
- 5. 利用移位寄存器实现随机数发生器 .....5
  - 1.设计思路 .....5
  - 2. 实验代码.....5
  - 3.RTL 视图 .....6
  - 4.引脚分配图 .....6
  - 5.实验结果 .....6
- 6.实验实验中遇到的问题及解决办法.....7

## 1.实验目的

本次实验的目的是熟悉数字电路中的常用原件的寄存器, 并使用 verilog 语言来实现。同时, 利用寄存器作为基本原理, 完成移位寄存器等一系列原件, 最后完成一个随机数生成器。

## 2.实验原理

寄存器是数字电路中的重要器件, 用于存储比特信号, 在 D 触发器的基础上用一定的逻辑门作为输入就可以构成最简单的寄存器。在此门电路可以涵盖清零, 置数的功能。在此基础上, 将多个 D 触发器链接在一起就可以构成移位寄存器, 然后根据需求拓展不同的功能。完成移位寄存器后通过

<i>n</i>	<i>Feedback Equation</i>	Table 8-21 Feedback equations for linear feedback shift-register counters.
2	$X_2 = X_1 \oplus X_0$	
3	$X_3 = X_1 \oplus X_0$	
4	$X_4 = X_1 \oplus X_0$	
5	$X_5 = X_2 \oplus X_0$	
6	$X_6 = X_1 \oplus X_0$	
7	$X_7 = X_3 \oplus X_0$	
8	$X_8 = X_4 \oplus X_3 \oplus X_2 \oplus X_0$	
12	$X_{12} = X_6 \oplus X_4 \oplus X_1 \oplus X_0$	
16	$X_{16} = X_5 \oplus X_4 \oplus X_3 \oplus X_0$	
20	$X_{20} = X_3 \oplus X_0$	
24	$X_{24} = X_7 \oplus X_2 \oplus X_1 \oplus X_0$	
28	$X_{28} = X_3 \oplus X_0$	
32	$X_{32} = X_{22} \oplus X_2 \oplus X_1 \oplus X_0$	

上表知  $X_4 \text{ xor } X_3 \text{ xor } X_2 \text{ xor } X_0$  可以得到随机数序列

### 3.实验环境/器材

本次实验的环境为 Quartus17.1 版本

本次实验的器材为 DE10 Standard 开发板

## 4.算术移位和逻辑移位寄存器

### 1.设计思路

本次实验的形式与 ALU 类似，同一个器件通过功能选择端完成多种功能，所以大框架使用 case 语句，always 语句检测 clk 信号，case 检测 select，根据不同的 select 选择不同的功能。而每种功能只需要根据 select 对 out 进行不同赋值即可。

### 2. 实验代码

模块名为 shift\_register，时钟信号为 clk，输入值为 cin，选择端为 select，载入输入端为 load，输出端为 8 路 out

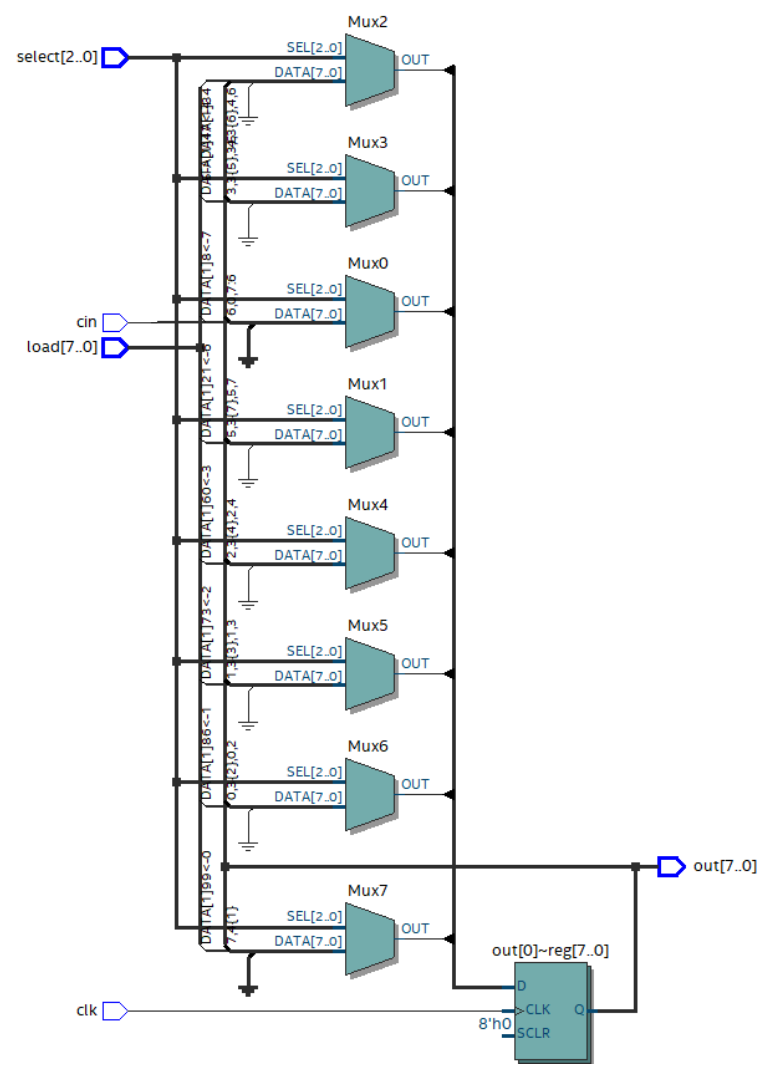
```

module shift_register(clk,cin,select,load,out);
    input clk,cin;
    input [2:0] select;
    input [7:0] load;
    output reg [7:0] out;

    always @(posedge clk)
    begin
        case(select)
            3'b000:out<=8'b00000000;
            3'b001:out<=load;
            3'b010:out<={1'b0,out[7:1]};
            3'b011:out<={out[6:0],1'b0};
            3'b100:out<={out[7],out[7:1]};
            3'b101:out<={cin,out[7:1]};
            3'b110:out<={out[0],out[7:1]};
            3'b111:out<={out[6:0],out[7]};
            default:out<=8'b11111111;
        endcase
    end
endmodule

```

3.RTL 视图



4.引脚分配图

in	cin	Input	PIN_AB30
in	clk	Input	PIN_AJ4
in	load[7]	Input	PIN_AA30
in	load[6]	Input	PIN_AC29
in	load[5]	Input	PIN_AD30
in	load[4]	Input	PIN_AC28
in	load[3]	Input	PIN_V25
in	load[2]	Input	PIN_W25
in	load[1]	Input	PIN_AC30
in	load[0]	Input	PIN_AB28
out	out[7]	Output	PIN_AF24
out	out[6]	Output	PIN_AE24
out	out[5]	Output	PIN_AF25
out	out[4]	Output	PIN_AG25
out	out[3]	Output	PIN_AD24
out	out[2]	Output	PIN_AC23
out	out[1]	Output	PIN_AB23
out	out[0]	Output	PIN_AA24
in	select[2]	Input	PIN_AA15
in	select[1]	Input	PIN_AA14
in	select[0]	Input	PIN_AK4

# 5.实验结果

已在开发板上验收

## 5. 利用移位寄存器实现随机数发生器

### 1.设计思路

在上实验中我们已经实现了移位寄存器,在本次实验中我们只需要将移位寄存器选择端调至101,选取输入作为移入信号,而移入信号的选取可以根据  $X_8 = X_4 \oplus X_3 \oplus X_2 \oplus X_0$ 。这样当初始状态不为00000000,即可出现除0外全部状态。最后,设计译码器将8位输出的高四位和第四位接入7段数码管显示。

### 2. 实验代码

顶层模块名为LFSR。其中shift\_register模块与上实现一致,decode\_hex为显示模块,xor为verilog内置的异或门模块。

```
module LFSR(  
    //////////// KEY ////////////  
    input      [3:0] KEY,  
    //////////// SW  ////////////  
    input      [9:0] SW,  
    //////////// LED  ////////////  
    output     [9:0] LEDR,  
    //////////// Seg7 ////////////  
    output     [6:0] HEX0,  
    output     [6:0] HEX1,  
    output     [6:0] HEX2,  
    output     [6:0] HEX3,  
    output     [6:0] HEX4,  
    output     [6:0] HEX5  
);  
wire c;  
  
//===== REG/WIRE declarations =====  
  
    shift_register sft1(KEY[0],c,3'b101,8'b00000000,LEDR[7:0]);  
    xor xor1(c,LEDR[2],LEDR[4],LEDR[5],LEDR[6]);  
    decode_hex hex0(LEDR[3:0],HEX0);  
    decode_hex hex1(LEDR[7:4],HEX1);
```

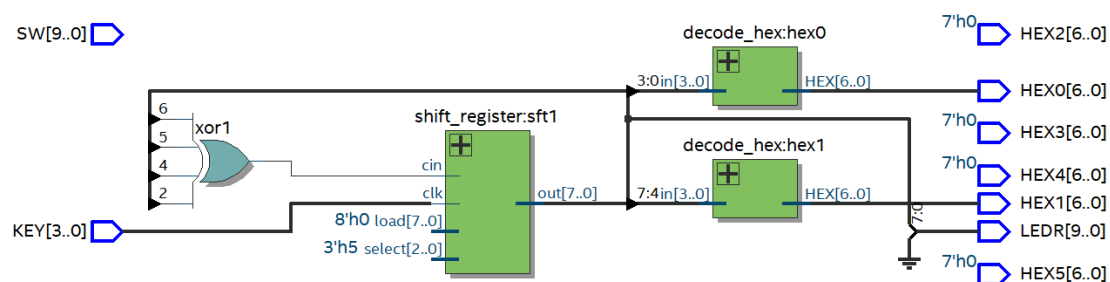
下为 decode\_hex 模块

```

module decode_hex(in,HEX);
  input [3:0] in;
  output reg [6:0] HEX;
  always @(in)
  begin
    case (in)
      0: HEX=7'b1000000;
      1: HEX=7'b1111001;
      2: HEX=7'b0100100;
      3: HEX=7'b0110000;
      4: HEX=7'b0011001;
      5: HEX=7'b0010010;
      6: HEX=7'b0000010;
      7: HEX=7'b1111000;
      8: HEX=7'b0000000;
      9: HEX=7'b0010000;
      10: HEX=7'b0001000;
      11: HEX=7'b0000011;
      12: HEX=7'b1000110;
      13: HEX=7'b0100001;
      14: HEX=7'b0000110;
      15: HEX=7'b0001110;
      default: HEX=7'b1111111;
    endcase
  end
endmodule

```

### 3.RTL 视图



### 4.引脚分配图

本项目由 build 软件自动生成，则不再展示引脚分配图

### 5.实验结果

已在开发板上验收

### 6.补充

由于题目中规定为需要自启动，为了保证全部的 256 种情况，需要增加对 00000000 的自启动规则，所以对逻辑反馈门进行改进，完成可以自启动的电路，改进后的方程如下：

```

nor nor1(D,LEDR[1],LEDR[2],LEDR[3],LEDR[4],LEDR[5],LEDR[6],LEDR[7]);
xor xor1(C,LEDR[0],LEDR[2],LEDR[3],LEDR[4],D);

```

其中 D 为反馈回的值

---

## 6. 实验实验中遇到的问题 及解决办法

---

对于随机数的生成，由于直接使用异或的门生成电路无法从  $8'b00000000$  自启动，所以需要自启动，有两种方式：1. 对反馈门电路进行修改，补充了从  $00000000$  启动的规则，最后完成了自启动 2. 将初始值改变，使其不为  $00000000$ ，这样就可以自启动了