实验六

寄存器的设计

实验报告　　日期：2020 年 10 月 23 日

姓名： 朱嘉琦

学号： 191220185

班级： 数电实验一班

邮箱： 1477194584@qq.com

实验六报告——寄存器

191220185 朱嘉琦

一、实验目的

本实验的目的是通过了解几中常用寄存器的设计方法，复习寄存器的原理，学习寄存器和常用的移位寄存器的设计。

二、实验原理

1. 寄存器

D 触发器可以用于存储比特信号，给 D 触发器加上置数功能就变成了一 位寄存器，如图6 1所示。由图中可以看出，如果load信号为1，则输入信号in被送入或门中，或门的另一个输入端为0，此时D=in，所以在下一个时钟里q=in。当load值为0时，q值被反馈到或门中，或门的另一个输入值为0，此时D=q，因此在下一个时钟周期里q值保持先前的值不变。

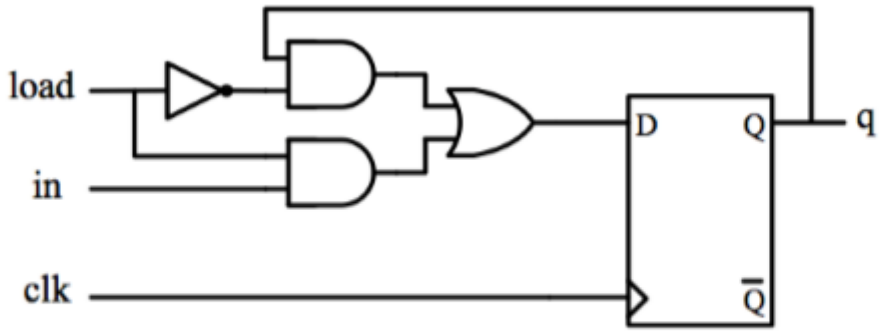
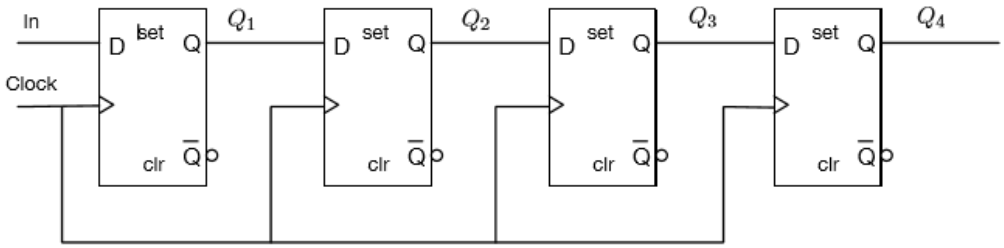


图 6-1: 1 位寄存器

将2个或者2个以上的1位寄存器组合在一起，这些寄存器共用一个时钟信号，这就构成了多位寄存器，寄存器常被用在计算机中存储数据，如指令寄存器、数据寄存器等。

2. 移位寄存器

移位寄存器是一类寄存器，它在时钟的触发沿，根据其控制信号，将存储 在其中的数据向某个方向移动一位。移位寄存器也是数字系统的常用器件。 图6 5(a)中是一个由4个D触发器构成的简单向右移位寄存器，数据从移位寄存器的左端输入，每个触发器的内容在时钟的正跳变沿（上升沿）将数据传到下一个触发器。图6 5(b)是一个此移位寄存器的序列传递实例。



(a) 移位寄存器框图

	In	Q1	Q2	Q3	Q4=Out
t0	1	0	0	0	0
t1	0	1	0	0	0
t2	1	0	1	0	0
t3	1	1	0	1	0
t4	1	1	1	0	1
t5	0	1	1	1	0
t6	0	0	1	1	1
t7	0	0	0	1	1

(b) 移位实例

图 6–5: 移位寄存器

3. 算术移位和逻辑移位

算数移位是指考虑到符号位的移位，算术移位要保证符号位不改变，算数左移同逻辑左移一样，算术右移最左面的空位补符号位。逻辑移位不管是向左移位还是向右移位都是空缺处补 0。循环是将一出去的那一位补充道空出的最高/最低位的移位方式。

三、实验环境/器材

实验环境是Quarters 17.1 Lite，实验器材是DE10 开发板

四、程序代码或流程图

1. 算术移位和逻辑移位寄存器

工作方式

表 6-3: 移位寄存器的工作方式

控制位	工作方式
0 0 0	清 0
0 0 1	置数
0 1 0	逻辑右移
0 1 1	逻辑左移
1 0 0	算术右移
1 0 1	左端串行输入 1 位值，并行输出 8 位值
1 1 0	循环右移
1 1 1	循环左移

```
module sftReg(in,clk,left,op,out);
   input [7:0]in;
   input [2:0]op;
   input clk,left;
   output reg[7:0]out;

   always@(posedge clk) begin
      case(op)
         0:out=0;
         1:out=in;
         2:out<={1'b0,out[7:1]};        //shift right
         3:out<={out[6:0],1'b0};        //shift left
         4:out<={out[7],out[7:1]};      //shift arithmetic right
         5:out<={left,out[7:1]};
         6:out<={out[0],out[7:1]};      //shift circular right
         7:out<={out[6:0],out[7]};      //shift circular left
      endcase
   end


endmodule
```

2. 利用移位寄存器实现随机数发生器

```
module rand(clk,num1,num0,rand_num);
   input clk;
   output [6:0]num1,num0;
   output reg[7:0]rand_num=255;

   always@(posedge clk) begin
      rand_num<={rand_num[4]^rand_num[3]^rand_num[2]^rand_num[0],rand_num[7:1]};
   end


   decoder d1(rand_num[3:0],num0);
   decoder d2(rand_num[7:4],num1);


endmodule
```

encoder模块（复用实验5）

```
module decoder(A,B);
    input [3:0]A;
    output reg[6:0]B;
    always @(*) begin
        case(A)
             0: B<=7'b1000000;              //0
             1: B<=7'b1111001;              //1
             2: B<=7'b0100100;              //2
             3: B<=7'b0110000;              //3
             4: B<=7'b0011001;              //4
             5: B<=7'b0010010;              //5
             6: B<=7'b0000010;              //6
             7: B<=7'b1111000;              //7
             8: B<=7'b0000000;              //8
             9: B<=7'b0010000;              //9
             10:B<=7'b0001000;              //A
             11:B<=7'b0000011;              //B
             12:B<=7'b1000110;              //C
             13:B<=7'b0100001;              //D
             14:B<=7'b0000110;              //E
             15:B<=7'b0001110;              //F


        default: B<=7'b1000000;             //0

    endcase
    end
endmodule
```

五、实验步骤

- 根据实验讲义复习寄存器和移位寄存器的相关内容。
- 了解如何利用verilog语言设计寄存器

表 6-2: 4 位寄存器代码

```
1  module register4(load,clk,clr,d,q);
2    input  load,clr,clk;
3    input  [3:0] d;
4    output reg [3:0] q;
5
6    always @(posedge clk)
7      if (clr==1)
8           q <= 0;
9      else if (load == 1)
10          q <= d;
11 endmodule
```

表8-40　功能扩展的8位移位寄存器的Verilog模块

```
module Vrshftreg ( CLK, CLR, RIN, LIN, S, D, Q );
  input CLK, CLR, RIN, LIN;
  input [2:0] S;
  input [7:0] D;
  output [7:0] Q;
  reg [7:0] Q;

  always @ (posedge CLK or posedge CLR)
    if (CLR == 1) Q <= 0;
    else case (S)
      0: Q <= Q;                // Hold
      1: Q <= D;                // Load
      2: Q <= {RIN, Q[7:1]};    // Shift right
      3: Q <= {Q[6:0], LIN};    // Shift left
      4: Q <= {Q[0], Q[7:1]};   // Shift circular right
      5: Q <= {Q[6:0], Q[7]};   // Shift circular left
      6: Q <= {Q[7], Q[7:1]};   // Shift arithmetic right
      7: Q <= {Q[6:0], 1'b0};   // Shift arithmetic left
      default Q <= 8'bx;        // should not occur
      endcase
endmodule
```

- 根据以上内容设计一个具有以下工作方式，可以通过控制位选择功能的寄存器。

表 6-3: 移位寄存器的工作方式

控制位	工作方式
000	清 0
001	置数
010	逻辑右移
011	逻辑左移
100	算术右移
101	左端串行输入 1 位值，并行输出 8 位值
110	循环右移
111	循环左移

- 进行分析与综合，检查是否有语法错误，最终编译通过。
- 利用ModelSim进行功能仿真，编写testbench文件，进行仿真测试 。
- 利用system builder分配引脚。

```
//=======================================================
//  REG/WIRE declarations
//=======================================================

sftReg r1(SW[7:0],KEY[3],SW[8],KEY[2:0],LEDR[7:0]);


//=======================================================
//  Structural coding
//=======================================================
```

- 进行编译，待编译通过后将二进制文件写入开发板 。

- 在开发板上进行硬件验证。
- 根据移位寄存器实现一个有255种状态的随机数发生器，并且将八位二进制数以十六进制显示在数码管上，系统需要能自启动。
- 根据书上的LFSR反馈方程来设计移位寄存器，复用实验五的七段码编码器的模块，利用system builder分配引脚。

表8-26　线性反馈移位寄存器计数器的反馈方程

n	反馈方程
2	$X2 = X1 \oplus X0$
3	$X3 = X1 \oplus X0$
4	$X4 = X1 \oplus X0$
5	$X5 = X2 \oplus X0$
6	$X6 = X1 \oplus X0$
7	$X7 = X3 \oplus X0$
8	$X8 = X4 \oplus X3 \oplus X2 \oplus X0$
12	$X12 = X6 \oplus X4 \oplus X1 \oplus X0$
16	$X16 = X5 \oplus X4 \oplus X3 \oplus X0$
20	$X20 = X3 \oplus X0$
24	$X24 = X7 \oplus X2 \oplus X1 \oplus X0$
28	$X28 = X3 \oplus X0$
32	$X32 = X22 \oplus X2 \oplus X1 \oplus X0$

```
//=======================================================
//  REG/WIRE declarations
//=======================================================

rand r1(KEY[3],HEX0,HEX1,LEDR);

//=======================================================
//  Structural coding
//=======================================================
```

- 进行分析与综合，检查是否有语法错误，最终编译通过。
- 进行编译，待编译通过后将二进制文件写入开发板 。
- 在开发板上进行硬件验证。

六、测试方法

1. 运行仿真测试，给输入信号赋予不同的值，观察图中输出与理论输出是否一致 。
2. 将sof文件导入开发板中实际测试电路，观察输出是否符合预期。

七、实验结果

寄存器和随机数生成器都正常工作。（已验收）

八、思考

如何生成更加复杂的伪随机数序列?

- 用更多的状态（触发器）可以得到随机性更强的随机数 。
- 初始值可以与时间相关联，转移函数也可与时间相关联 。

九、实验中遇到的问题及解决方法

在做第一个移位寄存器的时候分不清几个移位的区别，在书上找到了相关介绍和实例实现，根据书上的代码修改后得到了正确的代码，实验结果也正确。

十、意见和建议

无