实验名称：实验六 寄存器

姓名：张涵之

学号：191220154

班级：周一 5-6

邮箱：191220154@smail.nju.edu.cn

实验时间：2020/10/12

6.3.1 算术移位和逻辑移位寄存器

请根据表 6-3，用 Verilog HDL 语言设计移位寄存器，进行仿真查看移位寄存器的功能。对于移位寄存器的实现细节，请自行复习数电教科书 8.5 节内容，参考 8.5.9 节内容实现。

实验目的：设计一个带功能选择端的移位寄存器。

实验原理：利用 case 语句实现功能选择，用 Verilog HDL 语言很描述移位寄存器，如：Q <= {Q[0],Q[7：1]}；表示循环右移，Q <= {Q[7],Q[7：1]}；表示算术右移。
其中左端串行输入 1 位数值，并行输出 8 位数值是指每个时钟到来时右移一位，移入的最左位由外部开关决定，输出同其他情况一样为同时输出 8 位。

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

实验环境/器材：实验箱一个，笔记本电脑一台。

程序代码或流程图：

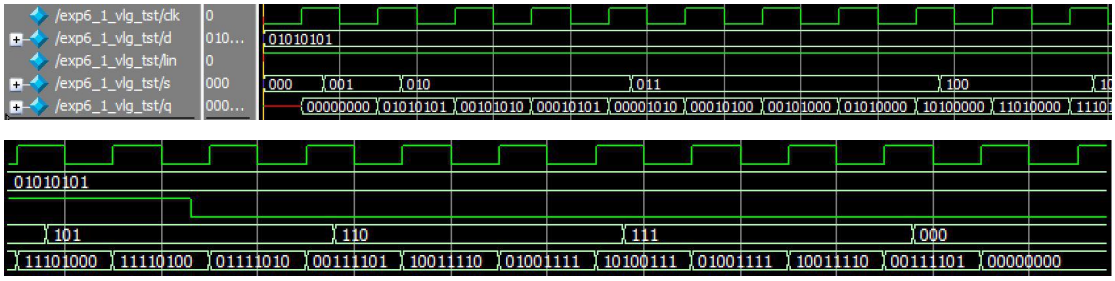
```
module exp6_1(clk,lin,s,d,q);
   input clk,lin;
   input [2:0] s;
   input [7:0] d;
   output reg [7:0] q;

always @ (posedge clk)
   case(s)
      0: q <= 0;
      1: q <= d;
      2: q <= {1'b0, q[7:1]};
      3: q <= {q[6:0], 1'b0};
      4: q <= {q[7], q[7:1]};
      5: q <= {lin, q[7:1]};
      6: q <= {q[0], q[7:1]};
      7: q <= {q[6:0], q[7]};
      default: q <= 8'bx;
   endcase
endmodule
```

实验步骤/过程：用 case 语句进行功能选择，采用赋值符号<=进行非阻塞赋值。
用 SW0~SW7 作数据端，SW9 为左端串行输入位，LEDR0~LEDR7 为输出显示，button KEY0 模拟时钟信号，每当时钟信号为上升沿时执行赋值语句，写测试代码进行仿真模拟。

测试方法：

```
initial
begin
// code that executes only once
// insert code here --> begin
   clk = 0; d = 7'b01010101; lin = 1; s = 000; #8;
                                      s = 001; #10;
                                      s = 010; #30;
                                      s = 011; #40;
                                      s = 100; #20;
                                      s = 101; #15;
                              lin = 0;         #15;
                                      s = 110; #30;
                                      s = 111; #30;
                                      s = 000; #20;
   $stop;
// --> end
//$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
   #5 clk = ~clk;
//@eachvec;
// --> end
end
endmodule
```

实验结果：



通过观察对比，移位寄存器的输出符合预期。经接入实验箱检验，显示也符合预期。

实验中遇到的问题及解决办法：实验非常顺利，没有碰到任何问题。

实验得到的启示：无。

意见和建议：无。

6.3.2 利用移位寄存器实现随机数发生器

我们可以利用 8 位移位寄存器来实现一个简单的随机数发生器。参考教科书第 534 页 LFSR 反馈方程设计一个 n=8，共有 255 种状态的随机数发生器。

请将 8 位二进制数以十六进制显示在数码管上，在 DE10-Standard 开发板上观察生成的随机数序列。系统需要能够自启动。

实验目的：实现一个简单的随机数发生器，以十六进制显示在数码管上。

实验原理：参考教科书第 534 页 LFSR 反馈方程，LSFR 计数器有 2^n-1 种有效状态（非零状态），查表可知 n=8 时的反馈方程为 X8 = X4⊕X3⊕X2⊕X0，通过 6.3.1 中移位寄存器的左端串行输入功能生成 8 位二进制数，每四位转化为十六进制显示在数码管上。

实验环境/器材：实验箱一个，笔记本电脑一台。

程序代码或流程图：

```
module clk_1s(clk,clk_1s);
   input clk;
   output reg clk_1s = 0;
   reg [24:0] count_clk = 1;

always @(posedge clk)
   if(count_clk == 25000000)
      begin
         count_clk <= 1;
         clk_1s <= ~clk_1s;
      end
   else
      count_clk <= count_clk + 1;
endmodule
```

```
module shift_register(clk,lin,s,d,q);
   input clk,lin;
   input [2:0] s;
   input [7:0] d;
   output reg [7:0] q = 8'b00000001;

always @ (posedge clk)
   case(s)
      0: q <= 0;
      1: q <= d;
      2: q <= {1'b0, q[7:1]};
      3: q <= {q[6:0], 1'b0};
      4: q <= {q[7], q[7:1]};
      5: q <= {lin, q[7:1]};
      6: q <= {q[0], q[7:1]};
      7: q <= {q[6:0], q[7]};
      default: q <= 8'bx;
   endcase
endmodule
```

```
module hex(in,out);
   input [3:0] in;
   output reg [6:0] out;

always @ (*)
   case(in)
      0: out = 7'b1000000;
      1: out = 7'b1111001;
      2: out = 7'b0100100;
      3: out = 7'b0110000;
      4: out = 7'b0011001;
      5: out = 7'b0010010;
      6: out = 7'b0000010;
      7: out = 7'b1111000;
      8: out = 7'b0000000;
      9: out = 7'b0010000;
      10: out = 7'b0001000;
      11: out = 7'b0000011;
      12: out = 7'b1000110;
      13: out = 7'b0100001;
      14: out = 7'b0000110;
      15: out = 7'b0001110;
      default: out = 7'b1111111;
   endcase
endmodule
```

```
module random(clk,hex1,hex2);
    input clk;
    wire clk_1s;
    wire [7:0] result;
    output [6:0] hex1;
    output [6:0] hex2;

    clk_1s c(clk,clk_1s);
    shift_register sr(clk,result[0]^result[2]^result[3]^result[4],
        3'b101,8'b00000000,result);
    hex h1(result[3:0],hex1);
    hex h2(result[7:4],hex2);
endmodule
```

*此处 sr 参数直接代入 clk 为 RTL 测试用，便于在短时间内（拖动较短的进度条内）观察到更多组输出，实际测试（使用实验箱接入时钟信号）时改为 clk_1s。

clk_1s 生成时钟信号 -> 时钟信号接入 shift_register 生成序列

-> hex 将生成序列转化为七段数码管的输入 -> 接入七段数码管显示。

实验步骤/过程：

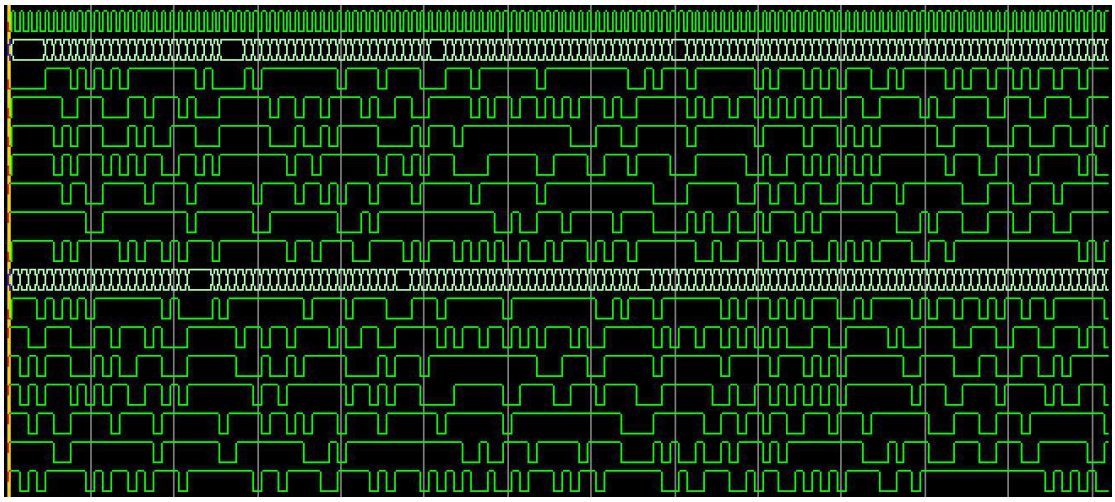
单独写出四个模块，把四个模块合并，写测试文件对顶层实体进行仿真，发现没有出现预期的效果，根据对自己的认识，推测是 1 秒钟生成器的问题。

对模块单独进行仿真，发现问题并修改，重新测试，在实验箱上进行操作。

测试方法：

```
initial
begin
// code that executes only once
// insert code here --> begin
    clk = 0; #1000;
    $stop;
// --> end
//$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
    #1; clk = ~clk;
//@eachvec;
// --> end
end
endmodule
```

实验结果：



通过观察，生成随机数行为符合预期。经接入实验箱检验，显示也符合预期。

实验中遇到的问题及解决办法：

1. 没有分别测试每个模块，直接测试了顶层模块，发现输出数据从头至尾保持不变。

   解决方法：将最不熟悉（不确定）的 1 秒时钟生成器（做实验 6 在实验 5 之前）设为顶层实体，循环终止条件 25000000 改为 2（便于测试），写测试代码进行仿真实验，观察到 clk_1s 的输出始终为不确定值（红色）。原来是没有初始化 clk_1s 和 count_clk 导致程序完全没有在计数，clk_1s = ~clk_1s 代码始终无效。

2. 直接将 1 秒钟生成器的输出用作时钟接到顶层模块，会导致仿真时需要很长时间数据才会发生一次变化，需要大量拖动进度条，给仿真结果的观察造成困难。

   解决方法：分开测试 1 秒钟生成器和顶层模块，便于写测试代码和观察仿真结果，实际想要实现的 1 秒生成 1 次随机数操作效果直接用实验箱进行测试。

实验得到的启示：

1. 不要把所有模块全部写完、链接好再开始测试，出现问题时会造成排查调试非常困难，最好每写完一个模块就对应地测试一次。
2. 可以适当进行分模块测试，最后直接在实验箱上进行综合。

意见和建议：无。

思考题：

生成的伪随机数序列仍然有一定的规律，如何能够生成更加复杂的伪随机数序列？

为了生成更加复杂的伪随机数序列，可以考虑对单纯左端移入产生的序列进行一些打乱重排操作，如生成的序列为 8 位，取序列高四位设为 a，低四位设为 b，则 a 和 b 各自表示一个大小在 0 和 15 之间的数，取序列的第 a%8 和第 b%8 位交换，得到更复杂的伪随机数。