浙江大学

本科实验报告

课程名称：	计算机逻辑设计基础
姓　　名：	李瀚轩
学　　院：	竺可桢学院
系：	所在系
专　　业：	计算机科学与技术
学　　号：	3220106039
指导教师：	董亚波

2023 年 12 月 20 日

浙江大学实验报告

课程名称： 计算机逻辑设计基础 实验类型： 综合

实验项目名称: 寄存器及寄存器传输设计

学生姓名： 李瀚轩 专业： 计算机科学与技术 学号： 3220106039

同组学生姓名： 指导老师： 董亚波

实验地点： 东四 509 实验日期： 2023 年 12 月 14 日

一、实验目的和要求

1. 掌握支持并行输入的移位寄存器的工作原理
2. 掌握支持并行输入的移位寄存器的设计方法

二、实验内容和原理

内容

1. 设计 8 位带并行输入的右移移位寄存器
2. 设计主板 LED 灯驱动模块
3. 设计主板七段数码管驱动模块

原理

2.1 移位寄存器

2.1.1 基本概念

每来一个时钟脉冲，寄存器中的数据按顺序向左或向右移动一位。必须采用主从触发器或边沿触发器，不能采用锁存器。数据移动方式：左移，右移，循环移位数据输入输出方式：串行输入，串行输出；串行输入，并行输出；并行输入，串行输出

2.1.2 串行输入右移移位寄存器

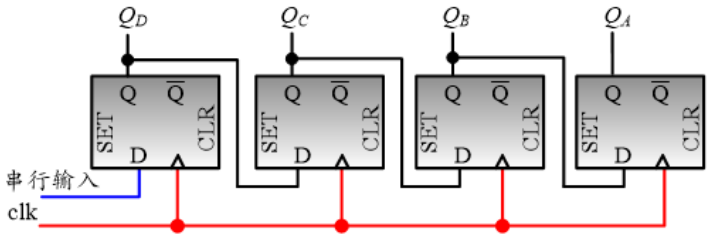

图 1: 串行输入右移移位寄存器

如上图所示，使用 D 触发器构成串行输入右移移位寄存器，每来一个时钟脉冲，寄存器中的数据向右移动一位，最右边的数据被丢弃，新的数据从最左边进入。

2.1.3 循环右移移位寄存器

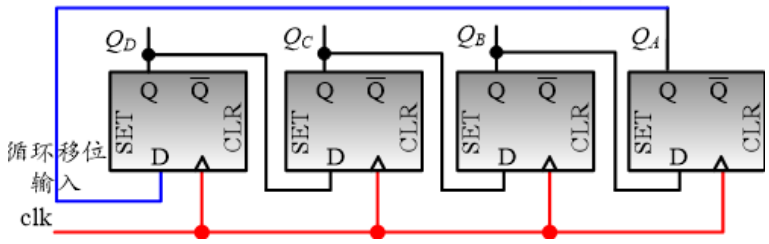

图 2: 循环右移移位寄存器

2.1.4 带并行输入的右移移位寄存器

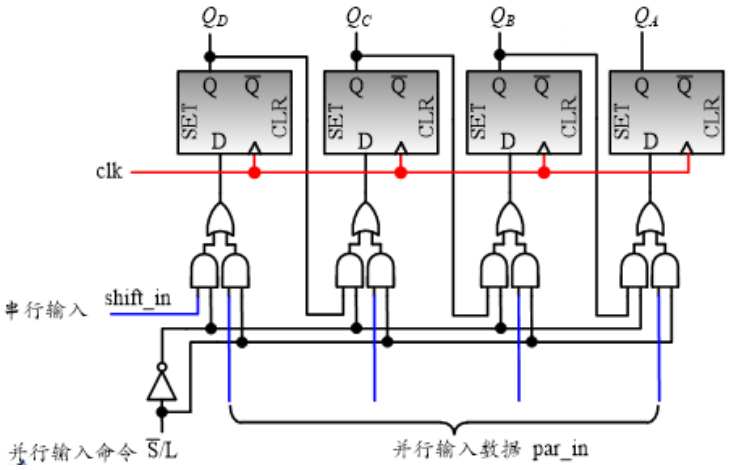

图 3: 带并行输入的右移移位寄存器

2.2 并行串行转换器

start 启动信号拉高以后，加载并行输入 D0-D7，启动左移串行输出，等 D0 输出后自动停止移位操作 finish 输出为 0 表示当前正在进行左移，为 1 表示移位停止

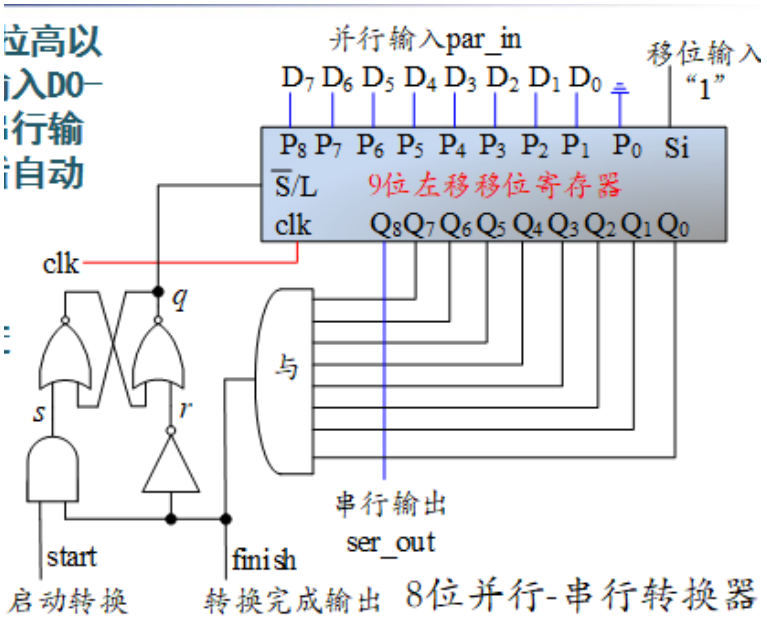

图 4: 并行串行转换器

三、实验过程和数据记录

3.1 设计 8 位带并行输入的右移移位寄存器

1. 新建工程 ShiftReg8b

2. 输入如下代码：

```
module shift_reg(
input wire clk,S_L,s_in,
input wire [7:0] p_in,
output wire [7:0] Q
);
 FD m0(.C(clk),.D((!S_L&Q[1])|(S_L&p_in[0])),.Q(Q[0]));
 FD m1(.C(clk),.D((!S_L&Q[2])|(S_L&p_in[1])),.Q(Q[1]));
 FD m2(.C(clk),.D((!S_L&Q[3])|(S_L&p_in[2])),.Q(Q[2]));
 FD m3(.C(clk),.D((!S_L&Q[4])|(S_L&p_in[3])),.Q(Q[3]));
 FD m4(.C(clk),.D((!S_L&Q[5])|(S_L&p_in[4])),.Q(Q[4]));
 FD m5(.C(clk),.D((!S_L&Q[6])|(S_L&p_in[5])),.Q(Q[5]));
 FD m6(.C(clk),.D((!S_L&Q[7])|(S_L&p_in[6])),.Q(Q[6]));
 FD m7(.C(clk),.D((!S_L&s_in)|(S_L&p_in[7])),.Q(Q[7]));
endmodule
```

3. 仿真，仿真代码如下：

```
module shift_reg_sim;

// Inputs
reg clk;
reg S_L;
reg s_in;
reg [7:0] p_in;

// Outputs
wire [7:0] Q;
```

```
// Instantiate the Unit Under Test (UUT)
shift_reg uut (
        .clk(clk),
        .S_L(S_L),
        .s_in(s_in),
        .p_in(p_in),
        .Q(Q)
);

initial begin
        // Initialize Inputs
        clk = 0;
        S_L = 0;
        s_in = 0;
        p_in = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        S_L=0;
        s_in=1;
        p_in=0;
        #200;
        S_L=1;
        s_in=0;
        p_in=8'b0101_0101;
        #500;

end
always begin
        clk=0;#20;
        clk=1;#20;
        end
```

endmodule

得到波形图如下：

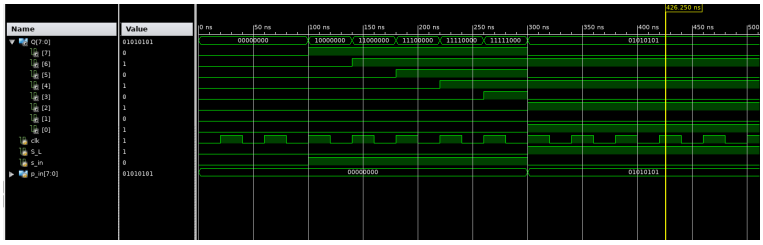

图 5: 仿真波形图

由波形图可以看到，S_L=0 时每个时钟周期寄存器向右移一位，S_L=1 时加载外部输入。仿真结果符合预期。

3.2 设计主板 LED 灯驱动模块

1. 新建工程 LEDP2S
2. 要求设计 4 个可自增的 4 位寄存器，汇总成总线 num[15:0], 显示在小实验板的 7 段数码管上；改造任务一中的 ShiftReg8b 为左移移位寄存器；利用 2 个 SLReg8b 和 1 个触发器，设计 LED-DRV,Verilog 代码如下：

```
module LED_DRV(
input wire clk,
input wire [15:0] SW,
output LED_CLK,
output LED_CLR,
output LED_EN,
output LED_DO,
output wire [15:0] num,
output wire [15:0] reg_num
);

wire [18:0] tmp;
wire finish, start, SL;
```

```
    assign LED_CLK = clk | finish;
    assign LED_CLR = 1'b1;
    assign LED_DO = tmp[16];
    assign LED_EN = 1'b1;

    assign finish=tmp[15] & tmp[14] & tmp[13] & tmp[12] & tmp[
    SR_LATCH m7(.S(start & finish), .R(~finish), .Q(SL));
    Regtrans4b m0(.clk(clk), .SW1(SW[0]), .SW2(SW[14]),.num(re
    Regtrans4b m1(.clk(clk), .SW1(SW[1]), .SW2(SW[14]),.num(re
    Regtrans4b m2(.clk(clk), .SW1(SW[2]), .SW2(SW[14]),.num(re
    Regtrans4b m3(.clk(clk), .SW1(SW[3]), .SW2(SW[14]),.num(re

    SLReg9b m4(.clk(clk), .S_L(SL), .s_in(1'b1), .p_in({reg_nu
    SLReg9b m5(.clk(clk), .S_L(SL), .s_in(tmp[8]), .p_in({1'b0

    LED m8(.clk(LED_CLK), .s_in(LED_DO), .num(num));
    Load_Gen m6(.clk(clk), .btn_in(SW[15]), .Load_out(start));

endmodule
```

要注意修改一下 SR_LATCH 的原理图，使用或非门实现。其中 Regtran4b 的 Verilog 代码如下：

```
module Regtrans4b(
input clk,
input wire SW1,
input wire SW2,
output wire [3:0] num
);
    wire Load_A;
    wire [3:0] A,A_IN,A1;
    wire [31:0] clk_div;
    assign num=A;

    MyRegister4b RegA(.clk(clk), .IN(A_IN), .Load(Load_A), .OU
    Load_Gen m0(.clk(clk), .btn_in(SW1),.Load_out(Load_A));
    clkdiv m3(clk, 1'b0, clk_div);
```

```
        AddSub4b m4(.A(A), .B(4'b0001), .Ctrl(1'b0), .S(A1));
        assign A_IN = (SW2 == 1'b0)? A1: 4'b0000;
endmodule
```

LED 模块代码如下：

```
    module LED(
        input wire clk,
        input wire s_in,
        output wire [15:0] num
    );
        reg [15:0] Register;
        always @(posedge clk) begin
                Register<={Register[14:0],s_in};
                end
                assign num=Register;
endmodule
```

3. 仿真，仿真代码如下：

```
        module LED_DRV_sim;

        // Inputs
        reg clk;
        reg [15:0] SW;

        // Outputs
        wire LED_CLK;
        wire LED_CLR;
        wire LED_EN;
        wire LED_DO;
        wire [15:0] num;
        wire [15:0] reg_num;

        // Instantiate the Unit Under Test (UUT)
        LED_DRV uut (
                .clk(clk),
                .SW(SW),
```

```
        .LED_CLK(LED_CLK),
        .LED_CLR(LED_CLR),
        .LED_EN(LED_EN),
        .LED_DO(LED_DO),
        .num(num),
        .reg_num(reg_num)
);
integer i;
initial begin
        // Initialize Inputs
        clk = 0;
       SW = 0;

        // Wait 100 ns for global reset to finish
         SW[14] = 1;
         SW[3] = 1; SW[2]=1; SW[1]=1; SW[0]=1; #20
         SW[3] = 0; SW[2]=0; SW[1]=0; SW[0]=0; #20

         SW[14] = 0;
          for (i=0;i<4;i=i+1)begin
         SW[3] = 0;#20 SW[3] = 1;#20;
          end
          for (i=0;i<3;i=i+1)begin
         SW[2] = 0;#20 SW[2] = 1;#20;
          end
          for (i=0;i<2;i=i+1)begin
         SW[1] = 0;#20 SW[1] = 1;#20;
          end

         SW[0] = 0;#20 SW[0] = 1;#20;
         SW[14] = 0;
        SW[15] = 1;#20
         SW[15] = 0;

          end
```

```
        always begin
        clk = 1; #10
        clk = 0; #10;

        // Add stimulus here
        end
endmodule
```

得到波形图如下：

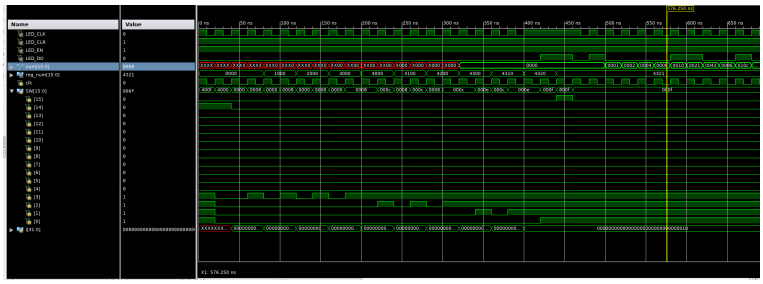

图 6: 仿真波形图

由波形图可以看到，SW[15]=0 时 16 位数据不断向左移位，最右边的填充输入为 1，SW[15]=1 时读入并行输入。

SW[0] 到 SW[3] 分别代表低位到高位的四个数字，每次波动开关从 0 到 1 到 0，就会往对应的寄存器写入新的值。SW[14]=1 时若拨动开关，会将寄存器的值置为 0. 所以我们可以通过该方法将寄存器初值设为 0，此后令 SW[14]=0。在仿真波形中，我将四个寄存器初始化，同时 tmp 每个周期左移一位，它的低十六位结果均为 1，因此 finish=1，移位结束。此时将 SW[15]=1, 两个周期后 num 通过移位的方式显示初始化在 reg_num 的数字。仿真结果符合预期。

4. 下载验证，引脚约束文件如下：

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "SW[0]" LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "SW[1]" LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "SW[2]" LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "SW[3]" LOC = AA12 | IOSTANDARD = LVCMOS15;
NET "SW[4]" LOC = Y13 | IOSTANDARD = LVCMOS15;
NET "SW[5]" LOC = Y12 | IOSTANDARD = LVCMOS15;
NET "SW[6]" LOC = AD11 | IOSTANDARD = LVCMOS15;
```

```
NET "SW[7]" LOC = AD10 | IOSTANDARD = LVCMOS15;
NET "SW[8]" LOC = AE10 | IOSTANDARD = LVCMOS15;
NET "SW[9]" LOC = AE12 | IOSTANDARD = LVCMOS15;
NET "SW[10]" LOC = AF12 | IOSTANDARD = LVCMOS15;
NET "SW[11]" LOC = AE8 | IOSTANDARD = LVCMOS15;
NET "SW[12]" LOC = AF8 | IOSTANDARD = LVCMOS15;
NET "SW[13]" LOC = AE13 | IOSTANDARD = LVCMOS15;
NET "SW[14]" LOC = AF13 | IOSTANDARD = LVCMOS15;
NET "SW[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
NET "SEGMENT[0]"LOC = AB22 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[2]" LOC = AD23 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[3]" LOC = Y21 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[4]" LOC = W20 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[5]" LOC = AC24 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[6]" LOC = AC23 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33;
NET "AN[0]" LOC = AD21 | IOSTANDARD = LVCMOS33;
NET "AN[1]" LOC = AC21 | IOSTANDARD = LVCMOS33;
NET "AN[2]" LOC = AB21 | IOSTANDARD = LVCMOS33;
NET "AN[3]" LOC = AC22 | IOSTANDARD = LVCMOS33;
NET "LED_CLK" LOC = N26 | IOSTANDARD = LVCMOS33 ;
NET "LED_CLR" LOC = N24 | IOSTANDARD = LVCMOS33 ;
NET "LED_DO" LOC = M26 | IOSTANDARD = LVCMOS33 ;
NET "LED_EN" LOC = P18 | IOSTANDARD = LVCMOS33 ;
```

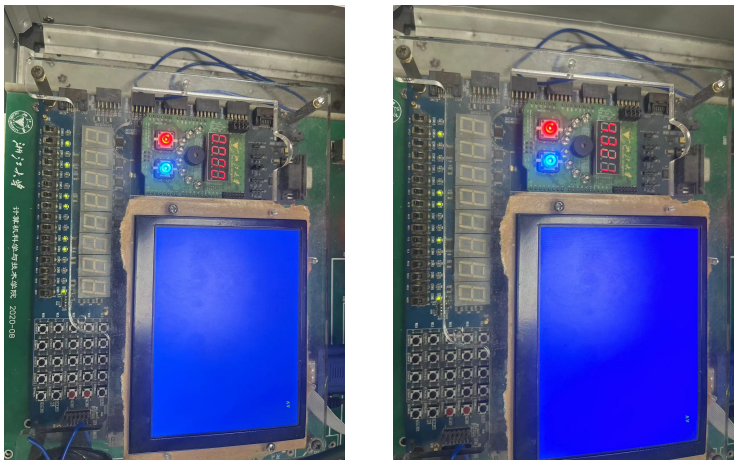

图 7: 验证主板 16 位 LED 驱动模块

用 BTNX4Y0 到 BTNX4Y4 作为自增按键，设置 4 位七段数码管的初值 4321，用 SW[15] 控制将 4 位七段数码管的数据输出到 LED 灯，如图 2 所示。图 1 是在我验证了其它数据之后将数码管调成 0000 的，所以 LED 灯有显示。但是结果正确性已在验收时得到验证。

3.3 设计主板七段数码管驱动模块

1. 新建工程 SEGP2S
2. 利用实验 12 任务一的电路，设计 8 个可自增的 4 位寄存器，接入总线 [31:0]，调用 8 个 MyMC14495 模块进行段码译码，利用 2 个 SLReg8b 和 1 个触发器设计主板 8 位数码管驱动模块 SEG_DRV
3. 新建源文件 SEGP2S，设为 top-module，verilog 代码如下：

```
module SEGP2S(
input wire clk,
input wire [15:0] SW,
output SEG_CLK,
output SEG_CLR,
output SEG_EN,
output SEG_DT
output wire [31:0] reg_num,
output wire [63:0] num,
//output wire [64:0] Q
```

```
);

        wire finish, start, SL;
        //wire [31:0] reg_num;
        //wire [63:0] num;
        wire [64:0] Q;

        assign SEG_CLR = 1'b1;
        assign SEG_EN = 1'b1;
        assign SEG_CLK = clk | finish;
        assign SEG_DT = Q[64];

        assign finish = &Q[63:0];

        Regtrans4b m0(.clk(clk), .SW1(SW[0]), .SW2(SW[14]),.num(reg
        Regtrans4b m1(.clk(clk), .SW1(SW[1]), .SW2(SW[14]),.num(reg
        Regtrans4b m2(.clk(clk), .SW1(SW[2]), .SW2(SW[14]),.num(reg
        Regtrans4b m3(.clk(clk), .SW1(SW[3]), .SW2(SW[14]),.num(reg
        Regtrans4b m4(.clk(clk), .SW1(SW[4]), .SW2(SW[14]),.num(reg
        Regtrans4b m5(.clk(clk), .SW1(SW[5]), .SW2(SW[14]),.num(reg
        Regtrans4b m6(.clk(clk), .SW1(SW[6]), .SW2(SW[14]),.num(reg
        Regtrans4b m7(.clk(clk), .SW1(SW[7]), .SW2(SW[14]),.num(reg

        SegmentDecoder m8(.hex(reg_num[3:0]), .Segment(num[7:0]));
        SegmentDecoder m9(.hex(reg_num[7:4]), .Segment(num[15:8]));
        SegmentDecoder m10(.hex(reg_num[11:8]), .Segment(num[23:16
        SegmentDecoder m11(.hex(reg_num[15:12]), .Segment(num[31:2
        SegmentDecoder m12(.hex(reg_num[19:16]), .Segment(num[39:3
        SegmentDecoder m13(.hex(reg_num[23:20]), .Segment(num[47:4
        SegmentDecoder m14(.hex(reg_num[27:24]), .Segment(num[55:4
        SegmentDecoder m15(.hex(reg_num[31:28]), .Segment(num[63:5

        SR_LATCH m16(.S(start & finish), .R(~finish),.Q(SL));

        wire [31:0] clk_div;
```

```
    clkdiv m26(clk, 1'b0, clk_div);
    Load_Gen m17(.clk(clk), .btn_in(SW[15]), .Load_out(start),

    SLReg9b m18(.clk(clk), .S_L(SL), .s_in(1'b1), .p_in({num[7:
    SLReg8b m19(.clk(clk), .S_L(SL), .s_in(Q[8]), .p_in(num[15:
    SLReg8b m20(.clk(clk), .S_L(SL), .s_in(Q[16]), .p_in(num[23
    SLReg8b m21(.clk(clk), .S_L(SL), .s_in(Q[24]), .p_in(num[31
    SLReg8b m22(.clk(clk), .S_L(SL), .s_in(Q[32]), .p_in(num[39
    SLReg8b m23(.clk(clk), .S_L(SL), .s_in(Q[40]), .p_in(num[47
    SLReg8b m24(.clk(clk), .S_L(SL), .s_in(Q[48]), .p_in(num[55
    SLReg8b m25(.clk(clk), .S_L(SL), .s_in(Q[56]), .p_in(num[63
endmodule
```

其中的 SegmentDecoder 是将十六进制数字转化为数码管亮暗的编号，代码如下：

```
                module SegmentDecoder(
                input [3:0] hex,
                output reg [7:0] Segment
);
        always @*
        begin
                case(hex)
                        4'h0: Segment[7:0] <= 8'b01000000;
                        4'h1: Segment[7:0] <= 8'b01111001;
                        4'h2: Segment[7:0] <= 8'b00100100;
                        4'h3: Segment[7:0] <= 8'b00110000;
                        4'h4: Segment[7:0] <= 8'b00011001;
                        4'h5: Segment[7:0] <= 8'b00010010;
                        4'h6: Segment[7:0] <= 8'b00000010;
                        4'h7: Segment[7:0] <= 8'b01111000;
                        4'h8: Segment[7:0] <= 8'b00000000;
                        4'h9: Segment[7:0] <= 8'b00010000;
                        4'hA: Segment[7:0] <= 8'b00001000;
                        4'hB: Segment[7:0] <= 8'b00000011;
                        4'hC: Segment[7:0] <= 8'b01000110;
```

```
                4'hD: Segment[7:0] <= 8'b00100001;
                4'hE: Segment[7:0] <= 8'b00000110;
                4'hF: Segment[7:0] <= 8'b00001110;
            endcase
        end
endmodule
```

4. 仿真，要求在 Top 模块将 Num 总线输出，以 16 进制显示，操作 SW[7:0] 将 8 个寄存器初值设为学号后 8 位，拨动 SW[15] 启动移位，观察 SEGCLK 和 SEGDT 的输出，仿真代码如下：

```
module SEGP2S_sim;

// Inputs
reg clk;
reg [15:0] SW;

// Outputs
wire SEG_CLK;
wire SEG_CLR;
wire SEG_EN;
wire SEG_DT;
wire [31:0] reg_num;
wire [63:0] num;

// Instantiate the Unit Under Test (UUT)
SEGP2S uut (
        .clk(clk),
        .SW(SW),
        .SEG_CLK(SEG_CLK),
        .SEG_CLR(SEG_CLR),
        .SEG_EN(SEG_EN),
        .SEG_DT(SEG_DT),
        .reg_num(reg_num),
        .num(num)
);
```

```
integer i;
initial begin
        // Initialize Inputs
        clk = 0;
        SW = 0;

        SW[14] = 1;
for(i=0;i<8;i=i+1)begin
SW[i] = 1;
end
#25
for(i=0;i<8;i=i+1)begin
SW[i] = 0;
end
#25

SW[14] = 0;
for(i=0;i<8;i=i+1)begin
SW[i] = 0;
end

SW[7] = 1; #20 SW[7] = 0; #25
SW[7] = 1; #20 SW[7] = 0; #25

SW[5] = 1; #20 SW[5] = 0; #25

for(i=0;i<6;i=i+1)begin
SW[3] = 1; #20 SW[3] = 0; #25;
end
for(i=0;i<3;i=i+1)begin
SW[1] = 1; #25 SW[2] = 0; #25;
end
for(i=0;i<9;i=i+1)begin
SW[0] = 1; #25 SW[0] = 0; #25;
end
```

```
        SW[14] = 0;
        #500
        SW[15] = 1;#20
        SW[15] = 0;

end
        always begin
                clk = 1; #10

                clk = 0; #10;
        end

endmodule
```

仿真波形图如下：

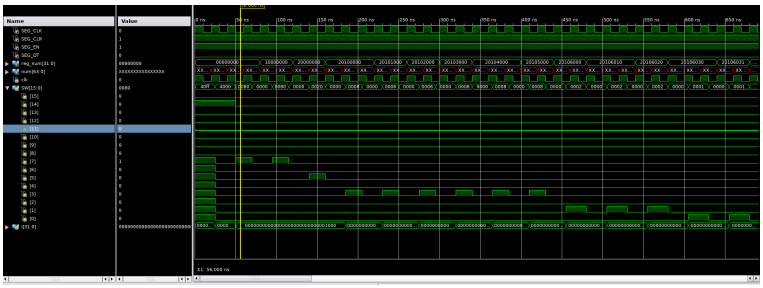

图 8: 仿真波形图 1

初始化 reg_num 为学号后 8 位 20106039.SW[14]=1 可以实现对寄存器的清零，SW[14]=1 就可以通过拨动开关对寄存器进行初始化，得到结果。

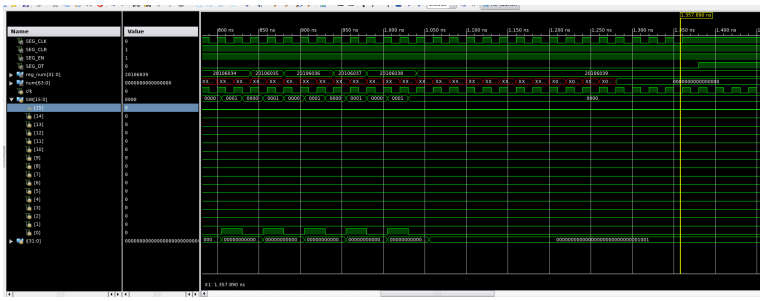

图 9: 仿真波形图 2

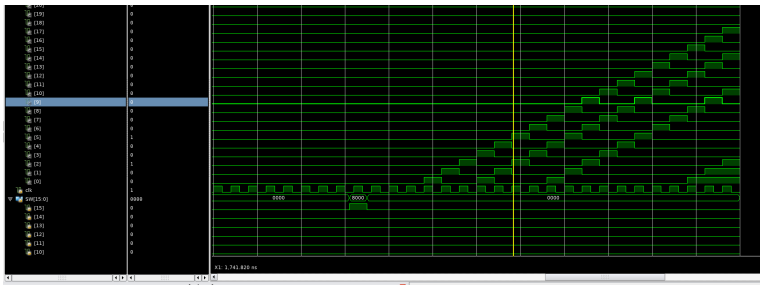

图 10: 仿真波形图 3

当输出的 num 全为 0 之后，说明移位寄存器已经将所有的 0 移除，因此 finish=1，如波形图 2 所示。此时拨动 SW[15] 之后等待两个周期开始移位，如波形图 3 所示。仿真结果符合预期。

5. 上板验证：引脚约束文件如下：

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "SW[0]" LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "SW[1]" LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "SW[2]" LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "SW[3]" LOC = AA12 | IOSTANDARD = LVCMOS15;
NET "SW[4]" LOC = Y13 | IOSTANDARD = LVCMOS15;
NET "SW[5]" LOC = Y12 | IOSTANDARD = LVCMOS15;
NET "SW[6]" LOC = AD11 | IOSTANDARD = LVCMOS15;
NET "SW[7]" LOC = AD10 | IOSTANDARD = LVCMOS15;
NET "SW[8]" LOC = AE10 | IOSTANDARD = LVCMOS15;
NET "SW[9]" LOC = AE12 | IOSTANDARD = LVCMOS15;
NET "SW[10]" LOC = AF12 | IOSTANDARD = LVCMOS15;
NET "SW[11]" LOC = AE8 | IOSTANDARD = LVCMOS15;
NET "SW[12]" LOC = AF8 | IOSTANDARD = LVCMOS15;
NET "SW[13]" LOC = AE13 | IOSTANDARD = LVCMOS15;
NET "SW[14]" LOC = AF13 | IOSTANDARD = LVCMOS15;
NET "SW[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
NET "SEG_CLK" LOC = M24 | IOSTANDARD = LVCMOS33 ;
NET "SEG_CLR" LOC = M20 | IOSTANDARD = LVCMOS33 ;
NET "SEG_DT" LOC = L24 | IOSTANDARD = LVCMOS33 ;
NET "SEG_EN" LOC = R18 | IOSTANDARD = LVCMOS33 ;
```

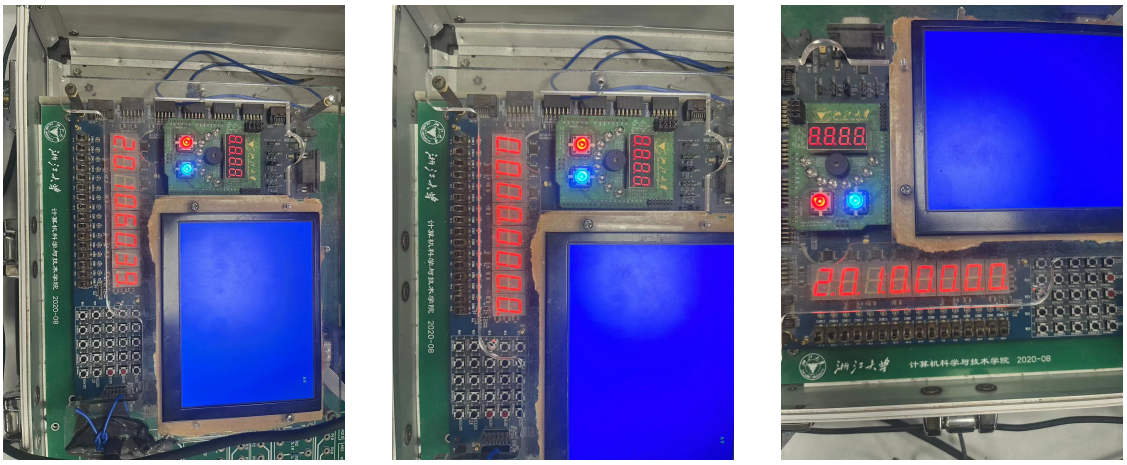

图 11: 验证主板七段 1 数码管驱动模块功能

可以看到，拨动 8 个开关 SW[7:0] 来修改每个数码管的数值使主板七段数码管设成显示学号后 8 位。而当 SW[14]=1 时，可以通过拨动对应的开关将对应的寄存器置 0。结果符合预期。

四、实验结果分析

相关结果都已经在前文写出，实验结果符合要求。

五、讨论与心得

这次实验复杂了很多，仿真和上板的代码不同，需要稍微修改，好几次我忘记改回来导致报错，也有点搞心态，也是比较晚才通过验收，但是苦尽甘来，我也学习到了很多东西，掌握了一些设计的思路，其中让移位停止的设计思路让我觉得很妙。总之，这次实验让我受益匪浅，也让我对数字电路的设计有了更深的理解。