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：	秦嘉俊
学　　院：	竺可桢学院
系：	所在系
专　　业：	计算机科学与技术
学　　号：	3210106182
指导教师：	董亚波

2022 年 12 月 19 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: 寄存器和寄存器传输设计

学生姓名： 秦嘉俊 专业： 计算机科学与技术 学号： 3210106182

同组学生姓名： 钟梓航 指导老师： 董亚波

实验地点： 东 4-509 实验日期： 2022 年 11 月 23 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

实验设备

- 装有 Xilinx ISE 14.7 的计算机 1 台
- SWORD 开发板 1 套

内容

1. 任务 1：设计 8 位带并行输入的右移移位寄存器
2. 任务 2：设计主板 LED 灯驱动模块
3. 任务 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 移位寄存器的基本概念

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

使用 D 触发器构成串行输入的右移移位寄存器。原理图如下所示：

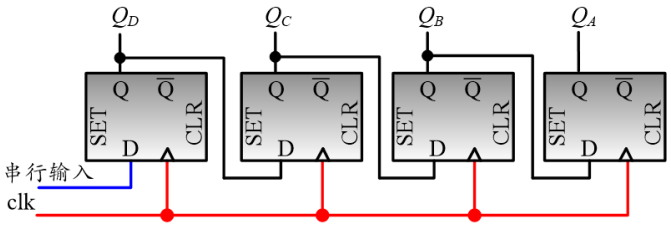


图 1: 串行输入右移移位寄存器原理图

2.1.3 循环右移移位寄存器

将上图中 D_A 的输出 Q_A 与 D_D 的输入相连接，就构成了循环右移移位寄存器。原理图如下所示：

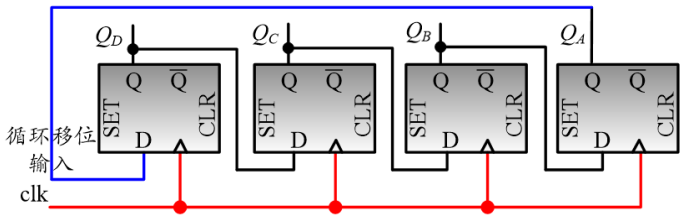


图 2: 循环右移移位寄存器原理图

2.2 带并行输入的移位寄存器

2.2.1 带并行输入的右移移位寄存器

数据输入方式：串行输入、并行输入。原理图如下所示：

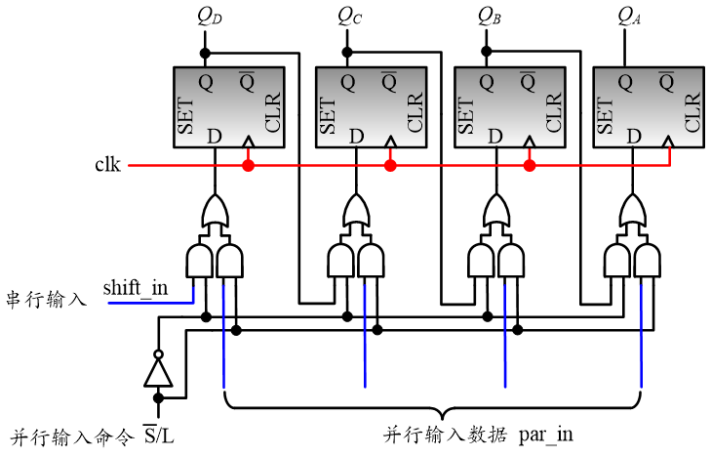
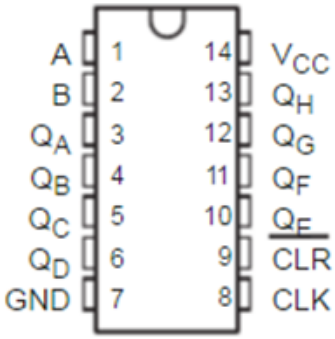


图 3: 带并行输入的右移移位寄存器原理图

2.2.2 74LV164A 芯片

74LV164A 是实验板上使用的芯片。它是一个 8 位串行右移移位寄存器，可以实现串-并转换。



(a) 74LV164A 引脚分布

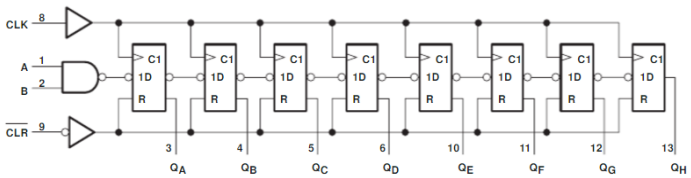
FUNCTION TABLE

INPUTS				OUTPUTS		
$\overline{CLR}$	CLK	A	B	Q_A	$Q_B \ldots Q_H$	
L	X	X	X	L	L	L
H	L	X	X	Q_{A0}	Q_{B0}	Q_{H0}
H	↑	H	H	H	Q_{An}	Q_{Gn}
H	↑	L	X	L	Q_{An}	Q_{Gn}
H	↑	X	L	L	Q_{An}	Q_{Gn}

Q_{A0}, Q_{B0}, Q_{H0} = the level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.

Q_{An}, Q_{Gn} = the level of Q_A or Q_G before the most recent ↑ transition of the clock; indicates a 1-bit shift.

(b) 74LV164A 的功能表



(c) 74LV164A 的逻辑原理图

图 4: 74LV164A 芯片

2.2.3 接口说明：实验板 16 位 LED 灯

实验板上，采用 2 个 74LV164A 构成 16 位串行输入并行输出移位寄存器，寄存器的并行输出控制 16 个 LED 灯。

图 5: 实验板上寄存器移位方向

实验板的电路图如下所示：

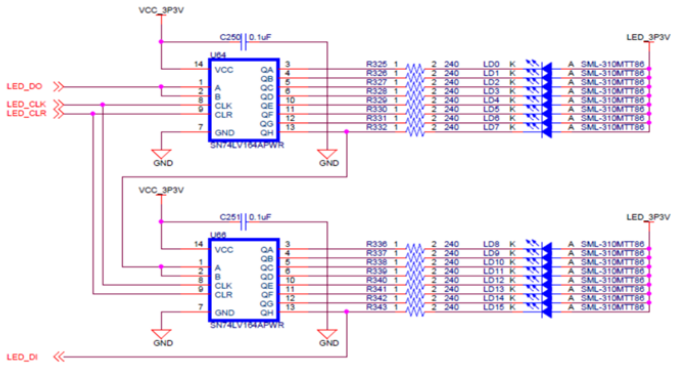


图 6: 实验板的逻辑电路图

引脚约束如下所示：

```
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 ;
```

- LED_CLK：16 位 LED 灯的时钟；
- LED_CLR：清零，使所有 LED 亮；
- LED_DO：16 位 LED 数据串行输入，输入 0 使 LED 亮；
- LED_EN：控制 LED 电源，1 为使能 LED 模块；
  16 位串行输入顺序是 LED15, LED14,...,LED1, LED0.

如下是一个输入示例:

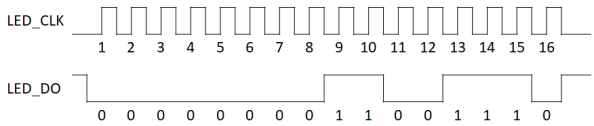


图 7: 输入示例

那么，从 LED15 到 LED0，应该呈现成：
亮暗亮暗亮亮亮亮暗暗亮亮暗亮暗亮暗
也就是说与 LED_D0 相对应。这里有一个问题。我们在 16 个周期后一定要把 LED_CLK 停下来，否则 74LV164A 内的数据就会一直在移位，就不会得到我们希望的结果。因此我们需要在移位结束后把时钟停掉。

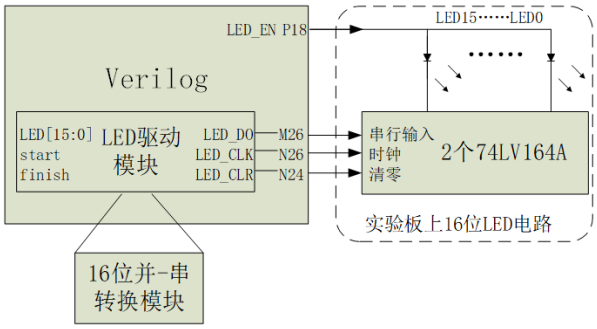


图 8: 实验板 16 位 LED 灯逻辑电路图

2.2.4 主板七段数码管

实验板上，8 个 74LS164A 的并行输出控制 8 个 7 段数码管的段码。七段数码管和 16 位 LED 灯本质上是一样的，只不过有 8 位 7 段数码管，共 64 位，因此用了 8 片 164 芯片串联在一起，实现了一个 64 位串入并出的 64 位移位寄存器。每一个数码管上面都带着一颗 164 芯片，来提供 8 位数据。如果希望 7 段数码管的某一位亮起来，移位进去的值应该是 0，反之则是 1。

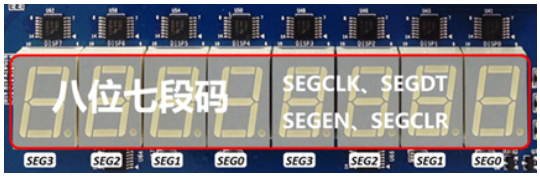


图 9: 8 位 7 段码

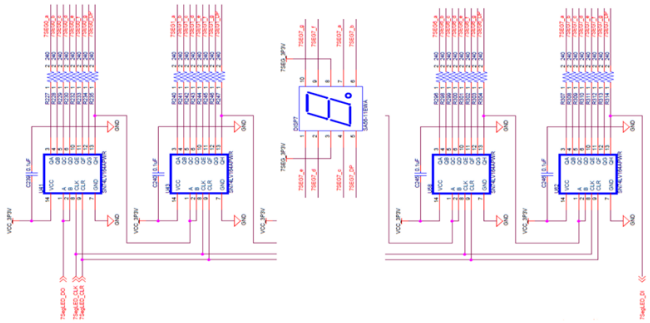


图 10: 主板七段数码管逻辑电路图

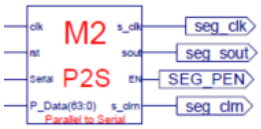


图 11: 逻辑符号图

通过并串转换电路输出：P_Data[63:0]=SEGMENT[63:0] 七段码移位输出的引脚约束如下所示：

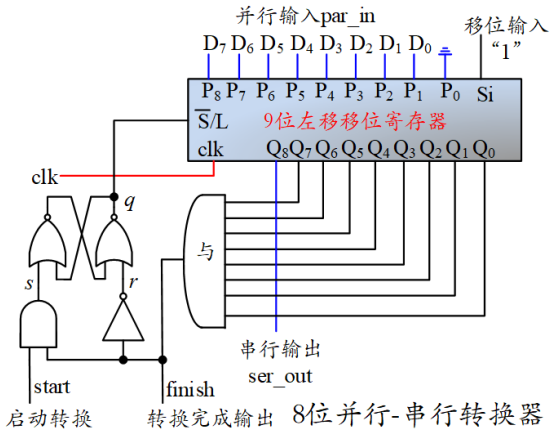
```
#七段码移位输出引脚约束
NET "SEGCLK" LOC = M24   | IOSTANDARD = LVCMOS33 ;
NET "SEGCLR" LOC = M20   | IOSTANDARD = LVCMOS33 ;
NET "SEGDT" LOC = L24    | IOSTANDARD = LVCMOS33 ;
NET "SEGEN" LOC = R18    | IOSTANDARD = LVCMOS33 ;
```

- SEGCLK：64 位串-并转换模块的时钟；
- SEGCLR：清零，所有段亮；
- SEGDT：数据串行输入，输入 0 亮；
- SEGEN：控制数码管电源，1 为使能；
- 主板上 8 位数码管显示采用的是静态显示，不是动态扫描方式
  - 实验板上用 8 个 74LV164A 构成 64 位串-并转换模块，并行输出控制 8 个 7 段数码管
  - 通过 SEGCLR 和 SEGDT 串行接收 8 个数码管 *8 段码，共计 64 位数据，移位先后顺序为SEG7_DP，SEG7_g，SEG7_f，...，SEG0_b，SEG0_a
  - 数码管共阳接法，段码为 0 时对应段亮
- 参考 16 位 LED 驱动模块，扩展设计 8 位数码管驱动模块
- **注意发送完成后停止时钟**

2.3 并行-串行转换器

2.3.1 没有启动命令

8 位并行-串行转换器如下图所示。start 启动信号拉高以后，自动加载 8 位并行输入，启动串行输出，等输出结束后自动停止移位。



首先 start 信号是控制启停的。0 表示不需要开始启动转换。或非门构成 SR 锁存器。移位寄存器里面假设并行输入中，有一位是 0，那么与门输出就是 0，那么经过非门之后锁存器 R 段是 1，锁存器输出 0。这时寄存器就开始移位功能。经过若干时钟周期后，并行输出全部为 1，与门输出为 1，锁存器 R 端变为 0，锁存器处于保存功能，这时 Finish 信号变为 1，说明移位已经完成。电路达到稳定状态。

2.3.2 有启动命令

有启动命令时，即 start 为 1，锁存器 s 端变成 1，r 端是 0，锁存器输出 q 等于 1，那么移位寄存器的 $\overline{S}/L$ 就是 1，移位寄存器进行并行加载的功能，也就是把 $P_0 \sim P_8$ 的九位数据送到 $Q_o \sim Q_8$ 上面去。由于 P_8 是 0，与门的输出一定是 0，finish 信号会变成 0，锁存器 r 端变成 1，那么 start 清零之后 q 变成 0，$\overline{S}/L$ 也是 0，就开始了移位操作。开始移位操作之后，最高位的 0 每个周期就会向右移一次，直到把 0 移出，标志移位结束。

根据这个 finish 信号，可以与 clk 信号组合起来，就可以提供给 LED 信号，可以有效提供转换结束的标志信号。

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, 工程名称用 ShfitReg8b. Top Level Source Type 用 HDL

2. 用结构化描述设计

   输入如下的 Verilog 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. 波形仿真

   输入如下仿真代码:

```
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)
```

```
13          shift_reg uut (
14              .clk(clk),
15              .S_L(S_L),
16              .s_in(s_in),
17              .p_in(p_in),
18              .Q(Q)
19          );
20
21          initial begin
22              // Initialize Inputs
23              clk = 0;
24              S_L = 0;
25              s_in = 0;
26              p_in = 0;
27
28              // Wait 100 ns for global reset to finish
29              #100;
30
31              // Add stimulus here
32              S_L = 0;
33              s_in = 1;
34              p_in =0;
35              #200;
36              S_L = 1;
37              s_in = 0;
38              p_in = 8'b0101_0101;
39              #500;
40          end
41
42          always begin
43              clk = 0; #20;
44              clk = 1; #20;
45          end
46
47      endmodule
```

得到波形图如下：

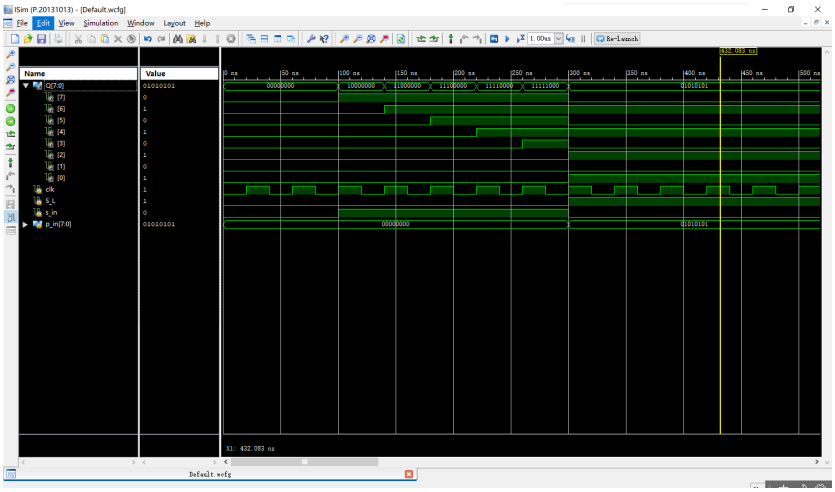


图 12: 仿真波形图

可以看到，`S_L=0` 时每个周期右移一位，`S_L=1` 时会读取外部输入，而且`S_L`如果不变，那么会每个周期都会读取外部输入，在这里我们没有改变外部输入因此最后的输出就一直不变。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, 工程名称用 LEDP2S. Top Level Source Type 用 HDL
2. 用行为描述设计
   要求:

   - 简化实验 12 任务一的电路，设计 4 个可设自增的 4 位寄存器，汇总成总线`num[15:0]`，显示在小实验板的 4 位七段数码管上
   - 改造 ShiftReg8b 模块为左移寄存器 SLReg8b
   - 利用 2 个 SLReg8b 模块和 1 个触发器，设计 16 位 LED 驱动模块`LED_DRV`

   新建 Verilog 文件，命名为 `LED_DRV`，并设为 top module。设置如下代码：

```
1          module LED_DRV(
2              input wire clk,
3              input wire [15:0] SW,
4              output LED_CLK,
5              output LED_CLR,
6              output LED_EN,
7              output LED_DO,
```

```
8           output wire [15:0] num,
9           output wire [15:0] reg_num
10
11          );
12
13          wire [18:0] tmp;
14          wire finish, start, SL;
15          assign LED_CLK = clk | finish;
16          assign LED_CLR = 1'b1;
17          assign LED_DO = tmp[16];
18          assign LED_EN = 1'b1;
19
20          assign finish = tmp[15] & tmp[14] & tmp[13] & tmp[
               12] & tmp[11] & tmp[10] & tmp[9] & tmp[8] & tmp[
               7] & tmp[6] & tmp[5] & tmp[4] & tmp[3] & tmp[2]
               & tmp[1] & tmp[0];
21
22          SR_LATCH m7(.S(start & finish), .R(~finish), .Q(SL)
               );
23          Regtrans4b m0(.clk(clk), .SW1(SW[0]), .SW2(SW[14]),
                .num(reg_num[3:0]));
24          Regtrans4b m1(.clk(clk), .SW1(SW[1]), .SW2(SW[14]),
                .num(reg_num[7:4]));
25          Regtrans4b m2(.clk(clk), .SW1(SW[2]), .SW2(SW[14]),
                .num(reg_num[11:8]));
26          Regtrans4b m3(.clk(clk), .SW1(SW[3]), .SW2(SW[14]),
                .num(reg_num[15:12]));
27
28          SLReg9b m4(.clk(clk), .S_L(SL), .s_in(1'b1), .p_in
               ({reg_num[7:0], 1'b0}), .Q(tmp[8:0]));
29          SLReg9b m5(.clk(clk), .S_L(SL), .s_in(tmp[8]), .
               p_in({1'b0, reg_num[15:8]}), .Q(tmp[17:9]));
30
31          LED m8(.clk(LED_CLK), .s_in(LED_DO), .num(num));
32
33          Load_Gen m6(.clk(clk), .btn_in(SW[15]), .Load_out(
               start));
34
```

```
endmodule
```

其中 SR_LATCH 是一个锁存器，其原理图如下：

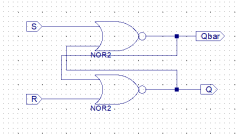


图 13: SR_LATCH

Regtrans4b 的代码改编自实验 12, 实现一个四位寄存器，如果 SW2=0 就每拨动一下自增一，SW2=1 就复位寄存器为 0, 具体代码如下

```
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), .OUT(A));
    Load_Gen m0(.clk(clk),  .btn_in(SW1),
    .Load_out(Load_A)); //寄存器 A 的 Load 信号
    clkdiv m3(clk, 1'b0, clk_div);
    AddSub4b m4(.A(A), .B(4'b0001), .Ctrl(1'b0), .S(A1)
       ); //自增/自减逻辑
    assign A_IN = (SW2 == 1'b0)? A1: 4'b0000; //2 选 1
       多路复用器, 复位寄存器初值

endmodule
```

值得注意的是，对于常规实验而言我们设计到这一步就结束了，但是因为疫情原因无法上板实验，我们需要通过仿真波形的方式验证结果。而 LED_DRV 所做的只是一个驱动模块，他通过传出 LED_CLK 和 LED_DO 来实现数字的显示。但在仿真中我们并没有实际的电子器件供我们使用，因此我们需要自行利用 LED_CLK 和 LED_DO 设计一个朴素的串行移位寄存器，以便显示我们的结果。这就是我们的 LED 模块，其代码如下

```
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. 自行设计激励代码，对驱动模块进行仿真
仿真要求：

- 在 Top 模块中将 num 总线输出，以 16 进制显示
- 操作 BTNX4Y0 到 BTNX4Y4，将 4 个寄存器初值设为 4321h，拨动SW[15]启动移位，观察LED_CLK和LED_DO的输出

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;
```

```
11          wire LED_DO;
12          wire [15:0] num;
13          wire [15:0] reg_num;
14
15          // Instantiate the Unit Under Test (UUT)
16          LED_DRV uut (
17              .clk(clk),
18              .SW(SW),
19              .LED_CLK(LED_CLK),
20              .LED_CLR(LED_CLR),
21              .LED_EN(LED_EN),
22              .LED_DO(LED_DO),
23              .num(num),
24              .reg_num(reg_num)
25          );
26          integer i;
27          initial begin
28              // Initialize Inputs
29              clk = 0;
30              SW = 0;
31
32              // Wait 100 ns for global reset to finish
33              SW[14] = 1;
34              SW[3] = 1; SW[2]=1; SW[1]=1; SW[0]=1; #20
35              SW[3] = 0; SW[2]=0; SW[1]=0; SW[0]=0; #20
36
37              SW[14] = 0;
38              for (i=0;i<4;i=i+1)begin
39              SW[3] = 0;#20 SW[3] = 1;#20;
40              end
41              for (i=0;i<3;i=i+1)begin
42              SW[2] = 0;#20 SW[2] = 1;#20;
43              end
44              for (i=0;i<2;i=i+1)begin
45              SW[1] = 0;#20 SW[1] = 1;#20;
46              end
47
48              SW[0] = 0;#20 SW[0] = 1;#20;
```

```
49
50              SW[14] = 0;
51              SW[15] = 1;#20
52              SW[15] = 0;
53
54          end
55         always begin
56                  clk = 1; #10
57                  clk = 0; #10;
58          end
59      endmodule
```

得到波形图如下：

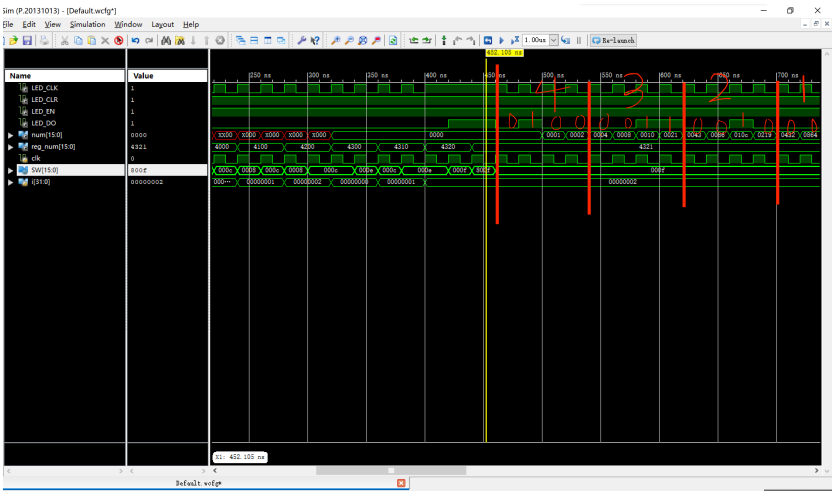


图 14: 仿真波形图

SW[15]代表我们希望模块执行的功能, SW[15]=0时 16 位数据不断向左移动一位（最右边的填充输入为 1）,SW[15]=1时读入并联输入

SW[0]到SW[3]分别代表由低到高的四个数字 (存在 reg_num 中), 每次拨动开关对应的寄存器都会存入新的值。SW[14]=1时拨动对应开关会使寄存器值初始化为 0，因此我们在最开始将所有寄存器赋为 0，随后 SW[14]=0, 此后每次拨动开关对应寄存器都会自增一。通过这样我们可以将四个寄存器实现初始化为 4321h, 初始化结束之后将 SW[14]=0. 与此同时我们的 tmp (移位寄存器的输出，并没有在仿真中输出) 每个周期也都左移了一位. 此时 tmp 低十六位均为 1(ffff) 因此 finish=1, 这时只要将 SW[15] 置为 1(start), 两个周期后 num 会通过移位的方式逐渐显示我们初始化在 reg_num 的数字 (4321h).(为什么不能立刻开始移位，可见实验结果分析部分)

3.3 设计主板七段数码管驱动模块

1. 新建工程, 工程名称用 SEGP2S. Top Level Source Type 用 HDL
2. 用行为描述设计要求如下：
   - 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`num[31:0]`
   - 调用 8 个 MyMC14495 模块进行段码译码
   - 利用 8 个 SLReg8b 模块和 1 个触发器，设计主板 8 位数码管驱动模块`SEG_DRV`

   新建 Verilog 文件，命名为 `SEGP2S`，并设为 top module。设置如下代码：

```
1          module SEGP2S(
2              input wire clk,
3              input wire [15:0] SW,
4              output SEG_CLK,
5              output SEG_CLR,
6              output SEG_EN,
7              output SEG_DT,
8              output wire [31:0] reg_num,
9              output wire [63:0] num
10             );
11             wire [18:0] tmp;
12             wire finish, start, SL;
13             wire [63:0] disp_num;
14             wire [64:0] Segment;
15
16             assign SEG_CLK = clk | finish;
17             assign SEG_CLR = 1'b1;
18            assign SEG_EN = 1'b1;
19             assign SEG_DT = Segment[64];
20
21             assign finish = Segment[0] & Segment[1] & Segment[2
                   ] & Segment[3] & Segment[4] & Segment[5] &
                   Segment[6] & Segment[7] & Segment[8] &
22             Segment[9] & Segment[10] & Segment[11] & Segment[12
                   ] & Segment[13] & Segment[14] & Segment[15] &
                   Segment[16] & Segment[17] &
```

```
23          Segment[18] & Segment[19] & Segment[20] & Segment[
               21] & Segment[22] & Segment[23] & Segment[24] &
               Segment[25] & Segment[26] &
24          Segment[27] & Segment[28] & Segment[29] & Segment[
               30] & Segment[31] & Segment[32] & Segment[33] &
               Segment[34] & Segment[35] &
25          Segment[36] & Segment[37] & Segment[38] & Segment[
               39] & Segment[40] & Segment[41] & Segment[42] &
               Segment[43] & Segment[44] &
26          Segment[45] & Segment[46] & Segment[47] & Segment[
               48] & Segment[49] & Segment[50] & Segment[51] &
               Segment[52] & Segment[53] &
27          Segment[54] & Segment[55] & Segment[56] & Segment[
               57] & Segment[58] & Segment[59] & Segment[60] &
               Segment[61] & Segment[62] &
28          Segment[63];
29
30          Regtrans4b m0(.clk(clk), .SW1(SW[0]), .SW2(SW[14]),
                .num(reg_num[3:0]));
31          Regtrans4b m1(.clk(clk), .SW1(SW[1]), .SW2(SW[14]),
                .num(reg_num[7:4]));
32          Regtrans4b m2(.clk(clk), .SW1(SW[2]), .SW2(SW[14]),
                .num(reg_num[11:8]));
33          Regtrans4b m3(.clk(clk), .SW1(SW[3]), .SW2(SW[14]),
                .num(reg_num[15:12]));
34          Regtrans4b m4(.clk(clk), .SW1(SW[4]), .SW2(SW[14]),
                .num(reg_num[19:16]));
35          Regtrans4b m5(.clk(clk), .SW1(SW[5]), .SW2(SW[14]),
                .num(reg_num[23:20]));
36          Regtrans4b m6(.clk(clk), .SW1(SW[6]), .SW2(SW[14]),
                .num(reg_num[27:24]));
37          Regtrans4b m7(.clk(clk), .SW1(SW[7]), .SW2(SW[14]),
                .num(reg_num[31:28]));
38
39          SegmentDecoder m16(.hex(reg_num[3:0]), .Segment(
               disp_num[7:0]));
40          SegmentDecoder m17(.hex(reg_num[7:4]), .Segment(
               disp_num[15:8]));
```

```
41          SegmentDecoder m18(.hex(reg_num[11:8]), .Segment(
               disp_num[23:16]));
42          SegmentDecoder m19(.hex(reg_num[15:12]), .Segment(
               disp_num[31:24]));
43          SegmentDecoder m20(.hex(reg_num[19:16]), .Segment(
               disp_num[39:32]));
44          SegmentDecoder m21(.hex(reg_num[23:20]), .Segment(
               disp_num[47:40]));
45          SegmentDecoder m22(.hex(reg_num[27:24]), .Segment(
               disp_num[55:48]));
46          SegmentDecoder m23(.hex(reg_num[31:28]), .Segment(
               disp_num[63:56]));
47
48          SLReg9b m8(.clk(clk), .S_L(SL), .s_in(1'b1), .p_in
               ({disp_num[7:0], 1'b0}), .Q(Segment[8:0]));
49          SLReg8b m9(.clk(clk), .S_L(SL), .s_in(Segment[8]),
               .p_in(disp_num[15:8]), .Q(Segment[16:9]));
50          SLReg8b m10(.clk(clk), .S_L(SL), .s_in(Segment[16])
               , .p_in(disp_num[23:16]), .Q(Segment[24:17]));
51          SLReg8b m11(.clk(clk), .S_L(SL), .s_in(Segment[24])
               , .p_in(disp_num[31:24]), .Q(Segment[32:25]));
52          SLReg8b m12(.clk(clk), .S_L(SL), .s_in(Segment[32])
               , .p_in(disp_num[39:32]), .Q(Segment[40:33]));
53          SLReg8b m13(.clk(clk), .S_L(SL), .s_in(Segment[40])
               , .p_in(disp_num[47:40]), .Q(Segment[48:41]));
54          SLReg8b m14(.clk(clk), .S_L(SL), .s_in(Segment[48])
               , .p_in(disp_num[55:48]), .Q(Segment[56:49]));
55          SLReg8b m15(.clk(clk), .S_L(SL), .s_in(Segment[56])
               , .p_in(disp_num[63:56]), .Q(Segment[64:57]));
56
57          //assign SL = 1'b0;
58          SR_LATCH m24(.S(start & finish), .R(~finish),.Q(SL)
               );
59
60          LED m26(.clk(SEG_CLK), .s_in(SEG_DT), .num(num));
61
62          Load_Gen m25(.clk(clk), .btn_in(SW[15]), .Load_out(
               start));
```

```
63
64          endmodule
```

其中 LED 与第二部分基本相同，SegmentDecoder 是将十六进制数字转化为数码管亮暗的编号，具体可以用 case 语句实现（也可以使用 MyMC14495 模块）

```
1           module SegmentDecoder(
2               input [3:0] hex,
3               output reg [7:0] Segment
4           );
5               always @*
6               begin
7                   case(hex)
8                       4'h0: Segment[7:0] <= 8'b01000000;
9                       4'h1: Segment[7:0] <= 8'b01111001;
10                      4'h2: Segment[7:0] <= 8'b00100100;
11                      4'h3: Segment[7:0] <= 8'b00110000;
12                      4'h4: Segment[7:0] <= 8'b00011001;
13                      4'h5: Segment[7:0] <= 8'b00010010;
14                      4'h6: Segment[7:0] <= 8'b00000010;
15                      4'h7: Segment[7:0] <= 8'b01111000;
16                      4'h8: Segment[7:0] <= 8'b00000000;
17                      4'h9: Segment[7:0] <= 8'b00010000;
18                      4'hA: Segment[7:0] <= 8'b00001000;
19                      4'hB: Segment[7:0] <= 8'b00000011;
20                      4'hC: Segment[7:0] <= 8'b01000110;
21                      4'hD: Segment[7:0] <= 8'b00100001;
22                      4'hE: Segment[7:0] <= 8'b00000110;
23                      4'hF: Segment[7:0] <= 8'b00001110;
24                      endcase
25              end
26          endmodule
```

3. 自行设计激励代码，对驱动模块进行仿真
   仿真要求：

   - 在 Top 模块中将 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;

    // Wait 100 ns for global reset to finish

    // Add stimulus here
```

```
35
36              SW[14] = 1;
37              for(i=0;i<8;i=i+1)begin
38                  SW[i] = 1;
39              end
40              #25
41              for(i=0;i<8;i=i+1)begin
42                  SW[i] = 0;
43              end
44              #25
45
46              SW[14] = 0;
47              for(i=0;i<8;i=i+1)begin
48                  SW[i] = 0;
49              end
50
51              SW[7] = 1; #20 SW[7] = 0; #25
52              SW[5] = 1; #20 SW[5] = 0; #25
53              for(i=0;i<6;i=i+1)begin
54                  SW[3] = 1; #20 SW[3] = 0; #25;
55              end
56              SW[2] = 1; #25 SW[2] = 0; #25
57
58              for(i=0;i<8;i=i+1)begin
59                  SW[1] = 1; #25 SW[1] = 0; #25;
60              end
61
62              for(i=0;i<2;i=i+1)begin
63                  SW[0] = 1; #25 SW[0] = 0; #25;
64              end
65              SW[14] = 0;
66              #500
67              SW[15] = 1;#20
68              SW[15] = 0;
69
70          end
71              always begin
72                  clk = 1; #10
```

```
73                  clk = 0; #10;
74              end
75
76      endmodule
```

仿真波形如下:

(a) 最开始初始化 reg_num 为学号后八位。SW[14]=1 拨动各个开关实现对八个寄存器的清零；然后 SW[14]=1 开始初始化各个寄存器的值，每次拨动 SW[i] 可以让第 i 个寄存器的值加 1(i:0..7)

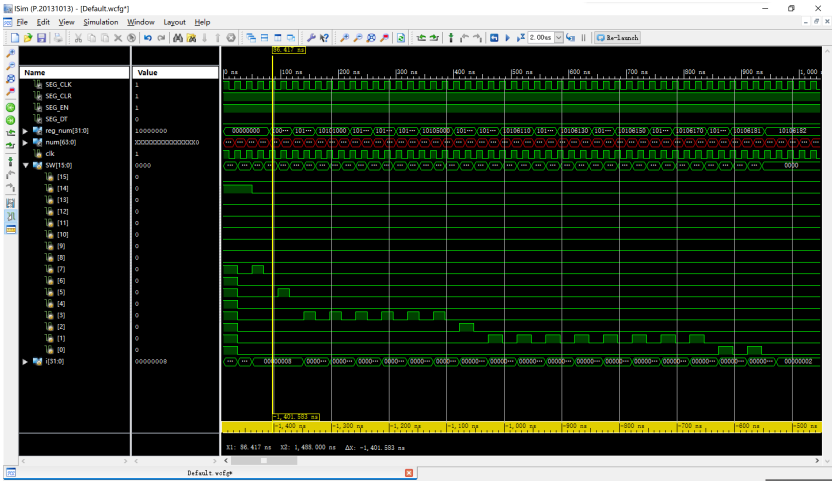


图 15: 仿真波形图 1

(b) 等待输出的 num 全零后，说明我们的移位寄存器 (segment, 仿真中间没有显示) 已经将所有 0 移出寄存器，现在移位寄存器内 [63:0] 的值为全 1, 因此finish=1.

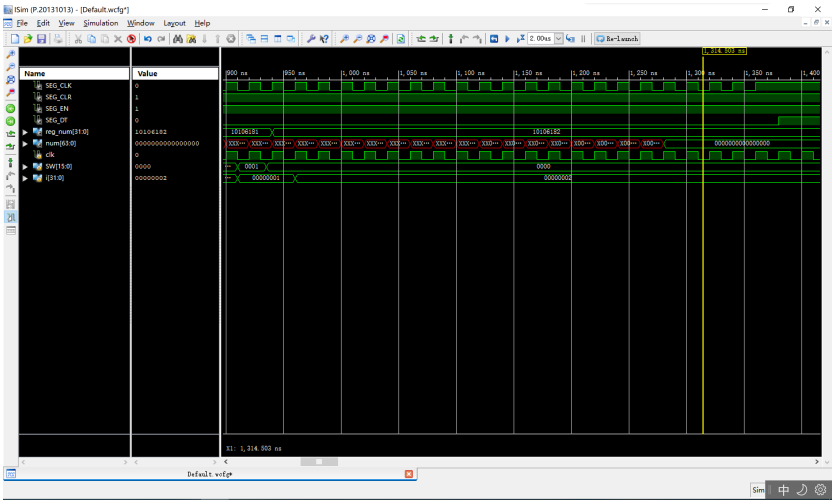


图 16: 仿真波形图 2

(c) 和 2 中一样拨动 SW[15] 之后等待两个周期后开始移位

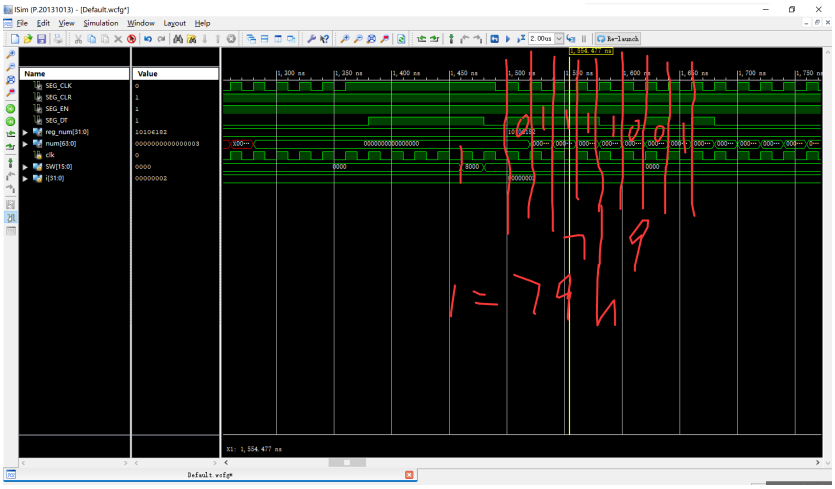


图 17: 仿真波形图 3

四、实验结果分析

相关结果都已经在前文写出。实验结果基本符合要求：仿真激励波形与真值表都相对应；仿真结果和 Verilog 代码在前文已经给出。

4.1 分析硬件描述代码

第二个实验，起初我们是将两个八位移位寄存器拼在一起形成十六位寄存器，但这样会一直移位，无法实现 num 读取外部并行输入后便不再移位的操作。

于是我们重新研究老师的 PPT, 发现真正要输出的 num 并非移位寄存器的输出，而是其最高位 ser_out, 并将 ser_out 作为另一个朴素的移位寄存器的移位输入进而显示我们并行输入的值。实际中是利用 LED_CLK 来作为朴素移位寄存器的时种。（若有实验板则我们只需要实现驱动模块，即将正确的 LED_CLK,LED_DO 输出即可，但受困于环境，我们需要自己模拟这样的功能）这样我们只要在移位结束后利用 finish 停止时钟即可保证显示的值不再改变。

随后我们又遇到了问题：num 还没有达到我们的并联输入就停止。一个反例是如果我们的 $D_0 = 1$, 那么只需要十五个周期整个 16 位移位寄存器就会全是 1, 随即停止移位。因此需要在原来的移位寄存器上增加一位并假设其并行输入为 0. 这样在我们可以保证整个寄存器移动 16 个周期。第三个实验也有类似的操作，从 64 位移位变为了 65 位移位。

4.2 分析仿真结果

在进行仿真验证结果时，我们参考实验 12 修改代码，去掉去抖、扫描显示等存在大延迟的模块，去掉 AN、SEGMENT 输出端口。

我们认为之所以在拨动 SW[15] 后需要等待两个周期才能开始显示结果，是因为第一个周期 Load_Gen 产生信号，第二个周期移位寄存器的 ser_out 传到 LED 上进行输出。但是因为 4321h 第一位是 0 所以两个周期后 num 显示还是 0000

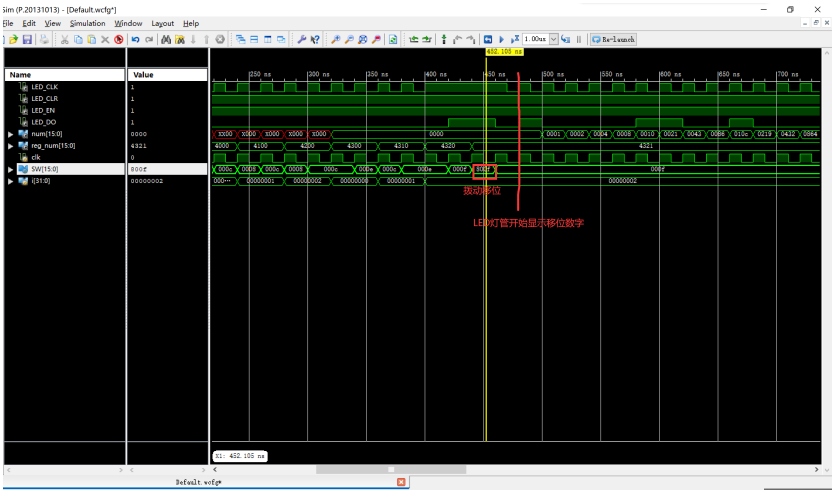
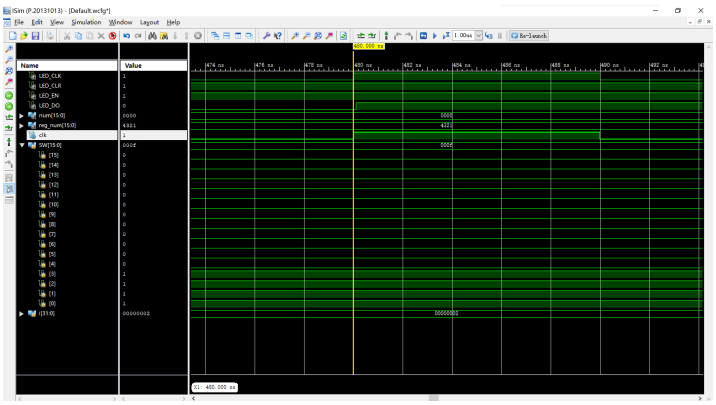
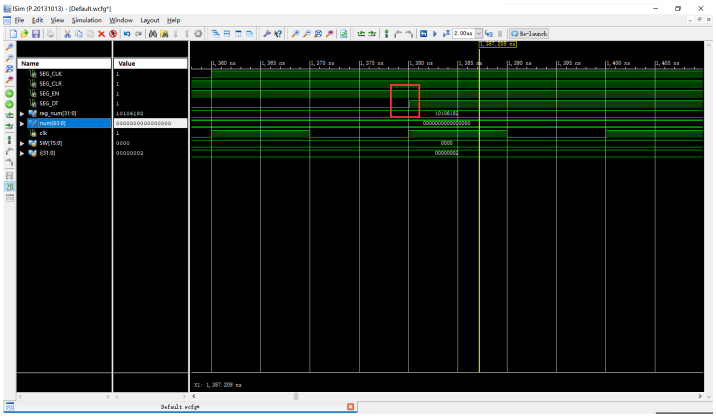


图 18: 分析仿真

放大来看，第一个信号是在 LED_CLK 上升沿的时候 LED_DO 有一个延迟所以第一次就把 0 传到 LED 上，第三个实验中同理。



(a) 任务 2 延迟



(b) 任务 3 延迟

图 19: 分析仿真

五、讨论与心得

这次的实验仿真难上了很多，我也反复和室友、助教哥哥探讨才得以顺利完成。好在终于要结束了！

再接再厉，继续努力！