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：	龙永奇
学　　院：	计算机科学与技术学院
系：	本系
专　　业：	计算机科学与技术
学　　号：	3220105907
指导教师：	董亚波

2023 年 12 月 27 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：龙永奇 专业：计算机科学与技术 学号：3220105907

同组学生姓名：周楠 指导老师：董亚波

实验地点：东 4-509 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. 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 16 位 LED 灯驱动模块
3. 任务 3：设计主板 8 位数码管驱动模块

原理：

1. 移位寄存器

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. 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：

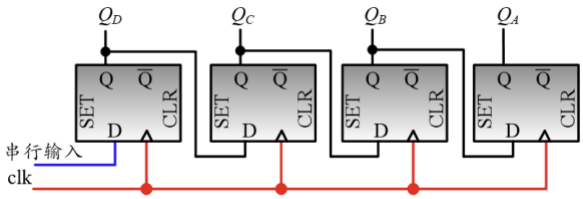

图 2.1 串行输入右移移位寄存器

3. 循环右移移位寄存器

将 D_A 的输出和 D_D 输入相连，原理图如下：

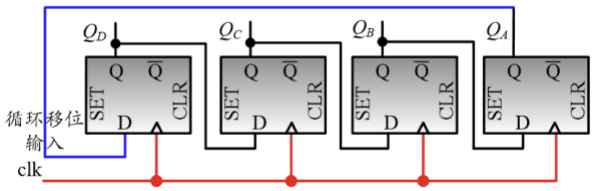

图 2.2 循环右移移位寄存器

4. 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：

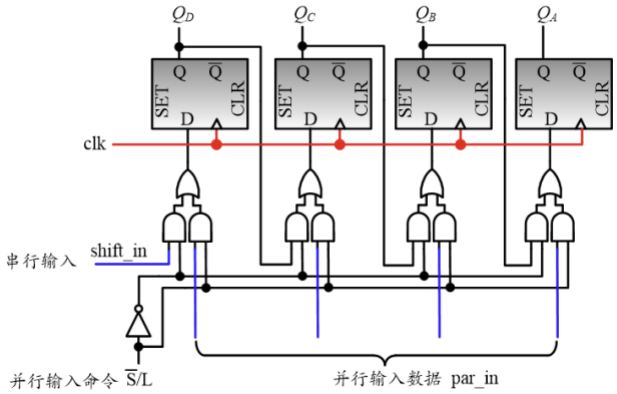

图 2.3 带并行输入的右移移位寄存器

其 Verilog 代码为：

```
`timescale 1ns / 1ps
module shift_reg(
        input wire clk,
        input wire S_L,
        input wire 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
```

5. **接口说明：16 位 LED 灯**

本次实验使用 74LV164A 芯片：

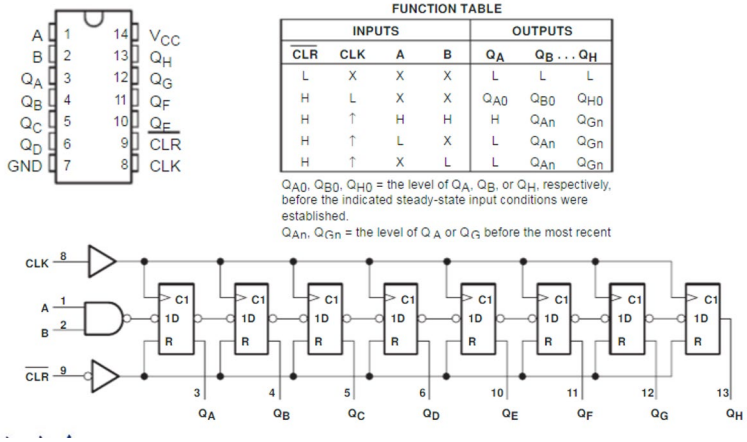

图 2.4 74LV164A 芯片

实验板上，采用 2 个 8 位移位寄存器 74LV164A 构成 16 位串行左移移位寄存器，其在实验板上的电路图如下：

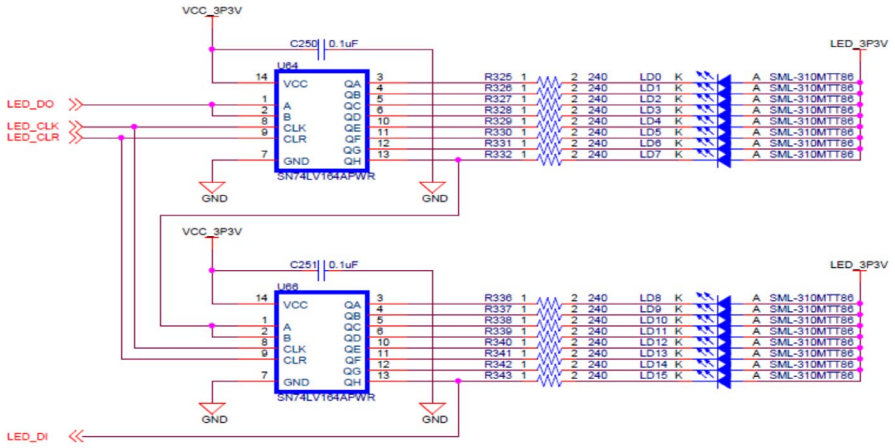

图 2.5 实验板电路图

寄存器的并行输出控制 16 个 LED 灯 LED0-LED15：

图 2.6 16 位 LED 灯

其引脚约束为：

```
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 灯模块的时钟，上升沿触发移位
- LED_CLR：清零，使所有 LED 亮，低电平有效
- LED_DO：串行移位数据输入，0 使 LED 亮
- LED_EN：LED 模块总控开关，1 为使能
- 用 LED_CLK 和 LED_DO 按顺序 LED15，LED14，……，LED1，LED0 串行移入 16 位数据

具体实现效果例如：

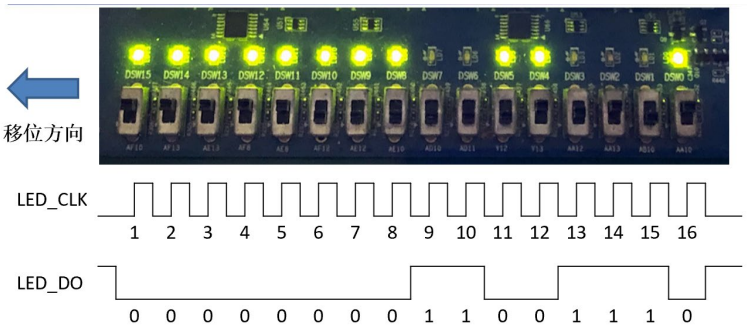

图 2.7 实验效果

采用 Verilog 语言设计 16 位 LED 驱动模块，模块里实现 16 位并行-串行转换模块，将需要显示的 16 位二进制数 num[15:0]求反串行左移输出到实验板上的 16 位 LED 灯模块，控制各位 LED 的亮暗。

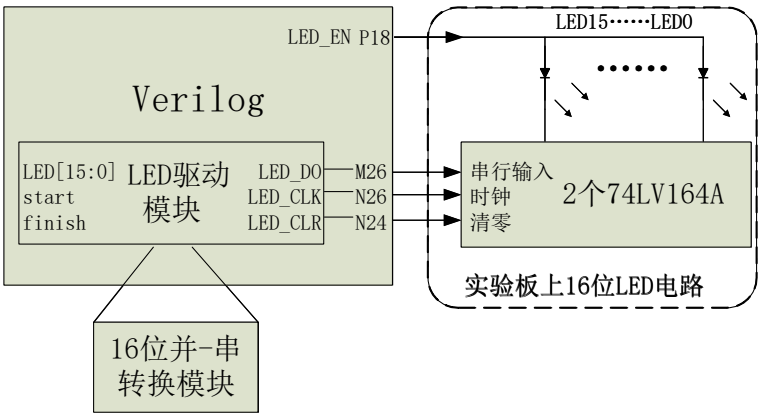

图 2.8 LED 逻辑电路图

因此 LED_15 到 LED_0 为： 亮暗亮暗亮亮亮亮暗暗亮亮暗亮暗亮暗

6. 并行－串行转换器设计

- 目标：

  将需要显示的 16 位二进制数 num[15:0]从 LED_DT 引脚左移输出到 16 位 LED 灯模块，同时在 LED_CLK 引脚上提供 16 个周期的时钟

- 关键：

  16 位数据移位完成后要停止时钟，避免把之前的数据移出 16 位 LED 灯模块

因此我们可以采用门控时钟方式给 LED_CLK 提供时钟：

```
assign LED_CLK = clk | finish
```

其中 finish 为转换结束标志，为 1 表示转换结束，可以通过计数+比较的方式实现 finish。

- 由于计数+比较的方式，硬件实现成本非常高，可以采用 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的方式：

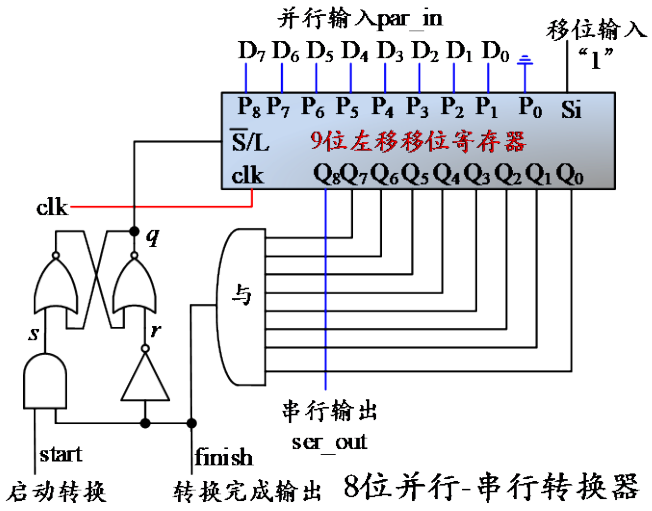

图 2.9 并行串行转换器

finish 输出为 0 表示当前正在进行左移，为 1 表示移位停止

- 没有启动命令 start = 0 时，移位寄存器处于左移状态，将“1”左移移入，Q 从 0 到 7 依次被置为 1

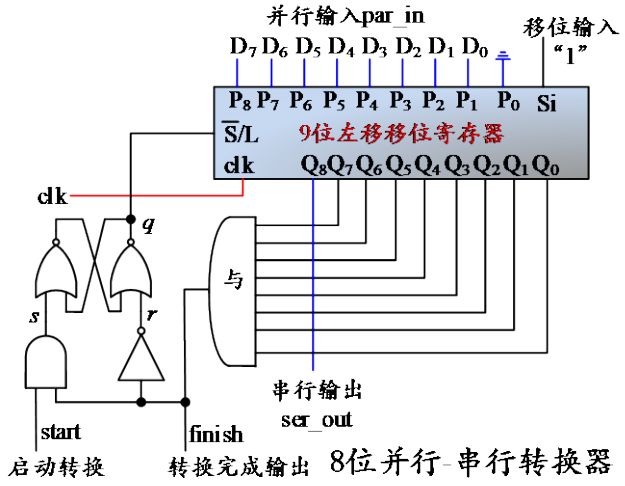

图 2.10 没有启动信号

- 有启动命令 start = 1 时，finish = S = 1，R = 0，加载并行输入数据，开始左移移位，finish 输出 0

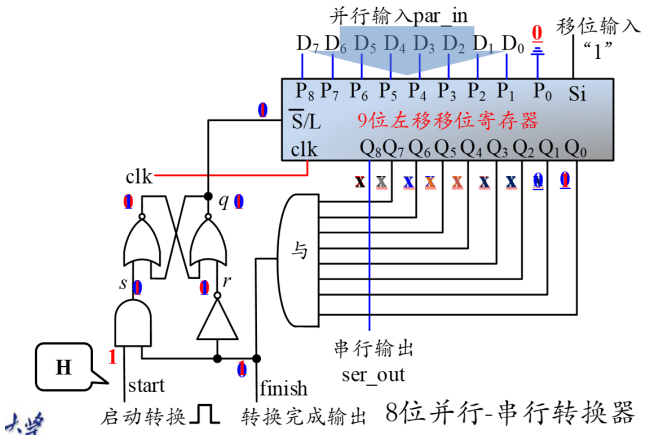

图 2.11 有启动信号

转换即将完成时，finish 依然输出 0

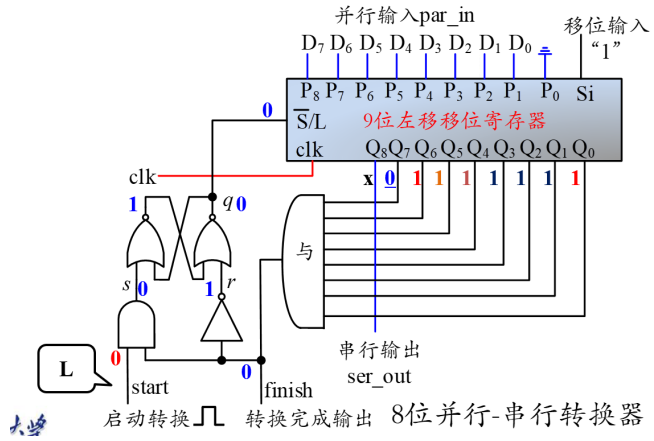

图 2.12 finish = 0

转换完成时，Q7-Q0 全部为 1，finish 输出 1，标志转换结束

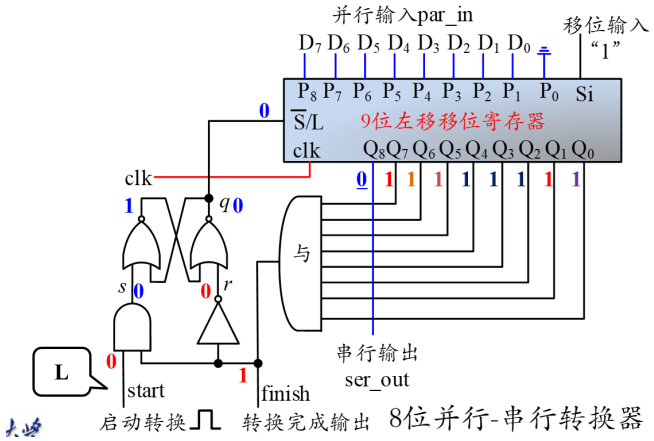

图 2.13 finish = 1

7. 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，注

意这里的七段数码管一共有 8 个，因此需要 64 位控制，需要将 8 个芯片串联实现一个 64 位移位寄存器

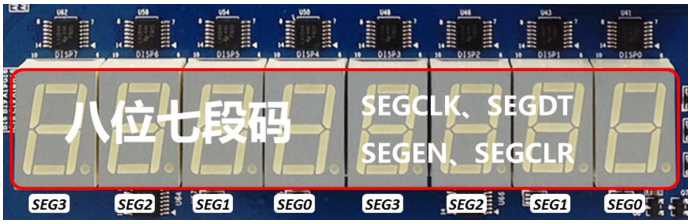

图 2.14 实验板上的 8 个七段数码管

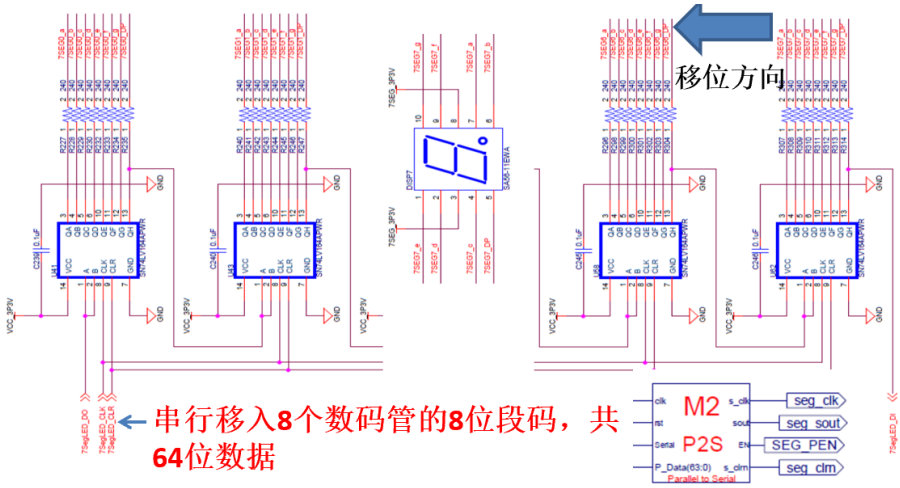

图 2.15 数码管电路图&逻辑符号图

其引脚约束为：

```
NET "SEGCLK"        LOC = M24    | IOSTANDARD = LVCMOS33 ;
NET "SEGCLR"        LOC = M20    | IOSTANDARD = LVCMOS33 ;
NET "SEGDT"      LOC = L24     | IOSTANDARD = LVCMOS33 ;
NET "SEGEN"      LOC = R18     | IOSTANDARD = LVCMOS33 ;
```

其中：

- SEGCLK：8 位七段数码管模块的时钟，上升沿触发移位
- SEGCLR：清零，所有段亮，低电平有效
- SEGDT：串行移位数据输入，0 亮
- SEGEN： 8 位七段数码管模块总控开关，1 为使能
- 用 SEGCLR 和 SEGDT 按顺序 SEG7_DP，SEG7_g，SEG7_f，……，SEG0_b，SEG0_a 串行移入 64 位数据

需要注意的是，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

可以参考 16 位 LED 驱动模块，扩展设计 8 位数码管驱动模块，需要注意发送完成后停止时钟

三、实验过程和数据记录

1. 设计 8 位带并行输入的右移移位寄存器

(a)新建工程 ShfitReg8b，Top Level Source 为 HDL

(b)新建 Verilog 类型源文件 shift_reg

(c)用 Verilog 代码方式设计，输入代码如下：

```
`timescale 1ns / 1ps
module shift_reg(
        input wire clk,
        input wire S_L,
        input wire 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
```

(d)建立仿真文件 shift_reg_sim.v，根据仿真要求：

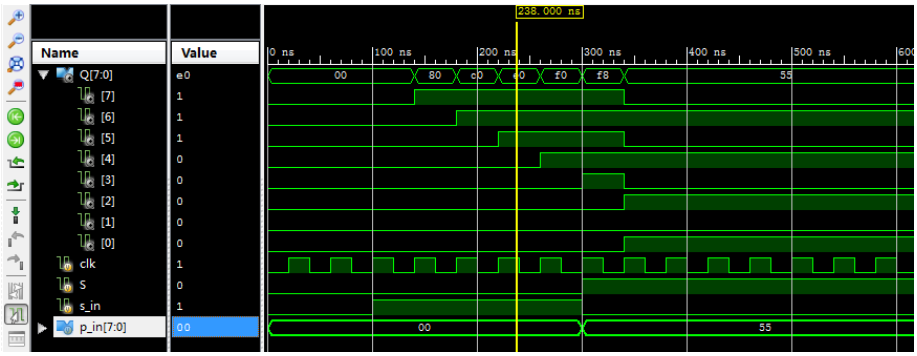

图 3.1 仿真要求

输入以下代码：

```
`timescale 1ns / 1ps
module shift_reg_sim;
    reg clk;
    reg S_L;
    reg s_in;
    reg [7:0] p_in;
    wire [7:0] Q;

    shift_reg uut (
        .clk(clk),
        .S_L(S_L),
        .s_in(s_in),
        .p_in(p_in),
        .Q(Q)
    );

    initial begin
        clk = 0;
        S_L = 0;
        s_in = 0;
        p_in = 0;
        #100;

        S_L = 0;
        p_in = 0; #20;
        s_in = 1; #400;
        S_L = 1;
        s_in = 0;
        p_in = 8'b01010101; #100;
        s_in = 1; #500;
    end
```

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

(e)得到波形图如下：

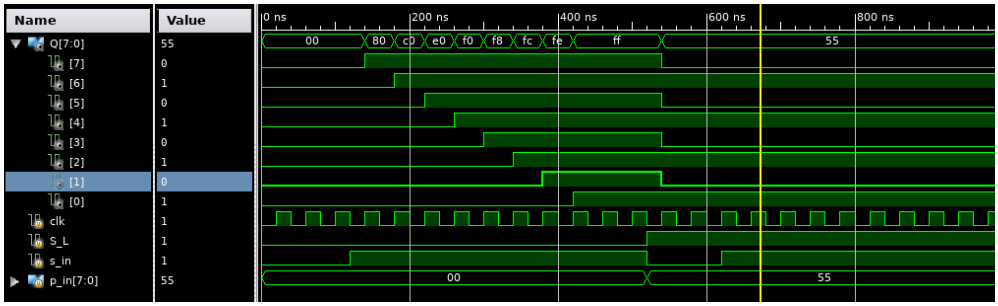

图 3.2 仿真波形图

- 初始的三个周期内（0 至 120ns）S_L 和 S_in 均为 0，Q = 0
- 此后 S_in = 1 从 Q7 直到 Q0 每隔 40ns 变为 1
- 当 S_L = 1 时并行输入 p_in = 01010101，上升沿时 Q = 01010101

仿真波形符合预期

2. 设计主板 16 位 LED 灯驱动模块

(a)新建工程 LEDP2S，Top Level Source 为 HDL

(b)使用行为描述设计，要求如下：

- 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

(c)本次实验采用将 4 个自增 4 位寄存器汇总得到总线 num[15:0]的方式来实现，最后显示在数码管上

(d)新建 Verilog 类型源文件 Regtrans4b，代码如下：

```
`timescale 1ns / 1ps
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), .clk_1ms(clk_div[17]),.btn_in(SW1), .Load_out(Lo
ad_A));
        clkdiv m1(clk, 1'b0, clk_div);
        AddSub4b m2(.A(A), .B(4'b0001), .Ctrl(1'b0), .S(A1));

        assign A_IN = (SW2 == 1'b0)? A1 : 4'b0000;

endmodule
```

(e)将之前设计的 8 位左移寄存器改为 9 位右移寄存器（把 s_in 变为 Q0 输入，8 到 0 改为 0 到 8 即可）：

```
`timescale 1ns / 1ps
module SLReg9b(
        input wire clk,
        input wire S_L,
        input wire s_in,
        input wire [8:0] p_in,
        output wire [8:0] Q
    );
        FD m8(.C(clk), .D((!S_L & Q[7]) | (S_L &
p_in[8])), .Q(Q[8]));
        FD m7(.C(clk), .D((!S_L & Q[6]) | (S_L &
p_in[7])), .Q(Q[7]));
        FD m6(.C(clk), .D((!S_L & Q[5]) | (S_L &
p_in[6])), .Q(Q[6]));
        FD m5(.C(clk), .D((!S_L & Q[4]) | (S_L &
p_in[5])), .Q(Q[5]));
        FD m4(.C(clk), .D((!S_L & Q[3]) | (S_L &
p_in[4])), .Q(Q[4]));
        FD m3(.C(clk), .D((!S_L & Q[2]) | (S_L &
p_in[3])), .Q(Q[3]));
        FD m2(.C(clk), .D((!S_L & Q[1]) | (S_L &
p_in[2])), .Q(Q[2]));
```

```
        FD m1(.C(clk), .D((!S_L & Q[0]) | (S_L &
p_in[1])), .Q(Q[1]));
        FD m0(.C(clk), .D((!S_L & s_in) | (S_L &
p_in[0])), .Q(Q[0]));
endmodule
```

(f)新建 Verilog 文件 LED，用 Verilog 代码方式设计，输入代码如下：

```
`timescale 1ns / 1ps
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
```

由于我们需要对 LED_DRV 进行仿真，因此需要在虚拟一个 LED 模块，用 01 代表现实中的 LED 灯亮灭

(g)注意到我们这里还需要一个 S`R`锁存器来生成并行信号

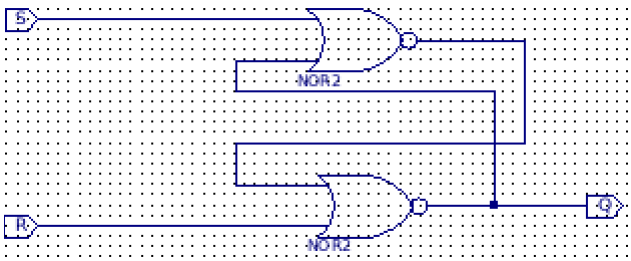

图 3.3 S`R`锁存器

(h)新建 Verilog 文件 LED_DRV，即我们的 top 文件，输入代码如下：

```
`timescale 1ns / 1ps
module LED_DRV(
        input wire clk,
        input wire [15:0] SW,
        output LED_CLK,
        output LED_CLR,
```

```
        output LED_EN,
        output LED_D0,
        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; // reset signal
        assign LED_D0 = tmp[16];
        assign LED_EN = 1'b1;
        // finish is 0 to move left otherwise stop, same as
PPT
        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];


        // Switch 1 is used to control the rising edge, and
switch 2 is used to control addition and subtraction
        Regtrans4b
m0(.clk(clk), .SW1(SW[0]), .SW2(SW[14]), .num(reg_num[3:0]));
        Regtrans4b
m1(.clk(clk), .SW1(SW[1]), .SW2(SW[14]), .num(reg_num[7:4]));
        Regtrans4b
m2(.clk(clk), .SW1(SW[2]), .SW2(SW[14]), .num(reg_num[11:8]));
        Regtrans4b
m3(.clk(clk), .SW1(SW[3]), .SW2(SW[14]), .num(reg_num[15:12]));

        SLReg9b
m4(.clk(clk), .S_L(SL), .s_in(1'b1), .p_in({reg_num[7:0],
1'b0}), .Q(tmp[8:0]));
        SLReg9b
m5(.clk(clk), .S_L(SL), .s_in(tmp[8]), .p_in({1'b0,
reg_num[15:8]}), .Q(tmp[17:9]));
        SR_LATCH m6(.S(start & finish), .R(~finish),.Q(SL));
        Load_Gen
m7(.clk(clk), .btn_in(SW[15]), .Load_out(start));

        // LEDs are only used for simulation, and are not
needed for the upper board
        LED m8(.clk(LED_CLK), .s_in(LED_D0), .num(num));
endmodule
```

(i)建立仿真文件 LED_DRV_sim.v，根据仿真要求：

- 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

输入以下代码：

```
`timescale 1ns / 1ps
module LED_DRV_sim;
    reg clk;
    reg [15:0] SW;

    wire LED_CLK;
    wire LED_CLR;
    wire LED_EN;
    wire LED_D0;
    wire [15:0] num;
    wire [15:0] reg_num;

    LED_DRV uut (
        .clk(clk),
        .SW(SW),
        .LED_CLK(LED_CLK),
        .LED_CLR(LED_CLR),
        .LED_EN(LED_EN),
        .LED_D0(LED_D0),
        .num(num),
        .reg_num(reg_num)
    );
    integer i;
    initial begin
        clk = 0;
        SW = 0;
        #100;

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

```
        end
        #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; // Dial it up and down
        SW[0] = 1; #20;

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

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

endmodule
```

(j)仿真波形如下：

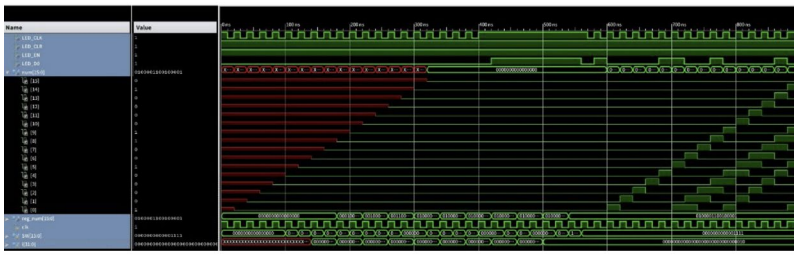

图 3.4 仿真波形

仿真开始时 SW[14] = 1，num 值不变，但是位移寄存器处于左移状

态，SW[15] = 0，从而所有位都置为 0。当 LED_DO 为 1，此时串行输入为 1，由于 LED_CLK 无上升沿，num 不受 tmp 影响。之后 SW[14] = 0，我们上下拨动开关让数码管显示 4321h，然后用 LED_DO 移入 0100，0011，0010，0001

(k)建立 K7.ucf 文件，输入代码如下：

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;

NET "LED_CLK"       LOC = N26   | IOSTANDARD = LVCMOS33 ;
NET "LED_CLR"       LOC = N24   | IOSTANDARD = LVCMOS33 ;
NET "LED_D0"        LOC = M26   | IOSTANDARD = LVCMOS33 ;
NET "LED_EN"        LOC = P18    | IOSTANDARD = LVCMOS33 ;

NET "btn[0]" LOC = W14     | IOSTANDARD = LVCMOS18;
NET "btn[0]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "btn[1]" LOC = V14    | IOSTANDARD = LVCMOS18;
NET "btn[1]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "btn[2]" LOC = V19    | IOSTANDARD = LVCMOS18;
NET "btn[2]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "btn[3]" LOC = V18     | IOSTANDARD = LVCMOS18;
NET "btn[3]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "BTNX4" LOC = W16 | IOSTANDARD = LVCMOS18;

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[3]" LOC = AC22 | IOSTANDARD = LVCMOS33 ;
NET "AN[2]" LOC = AB21 | IOSTANDARD = LVCMOS33 ;
NET "AN[1]" LOC = AC21 | IOSTANDARD = LVCMOS33 ;
NET "AN[0]" LOC = AD21 | IOSTANDARD = LVCMOS33 ;
```

其中 BTNX4Y0、Y1、Y2、Y3 均为自增键盘

(l)生成 bit 文件，上板验证

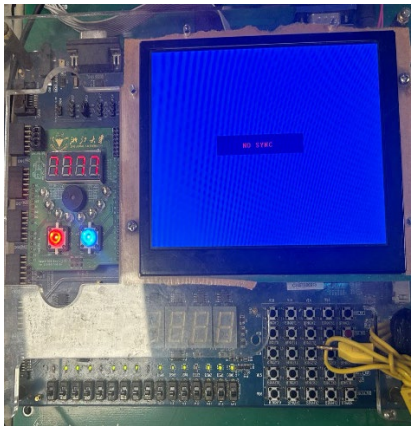

图 3.5 7777

图 3.6 3343

- 初始值 7777，转换为二进制为：0111011101110111，1 亮 0 灭
- 拨动开关得到了 3343，转换为二进制为：0011001101000011

3. 设计主板 8 位数码管驱动模块

(a)新建工程 SEGP2S，Top Level Source Type 使用 HDL

(b)用行为描述设计，要求为：

- 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
- 自行设计激励代码，对驱动模块进行仿真

(c)新建 Verilog 文件 Decoder，用 Verilog 代码方式设计，输入代码如下：

```
`timescale 1ns / 1ps
module Decoder(
    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
```

Decoder 用来将数字转换为数码管亮暗

(d)新建 Verilog 文件 SEGP2S, Top Level Source Type 用 HDL

```
`timescale 1ns / 1ps
module SEGP2S(
        input wire clk,
        input wire [15:0] SW,
        output SEG_CLK,
        output SEG_CLR,
        output SEG_EN,
        output SEG_DT,
        output [63:0] num,
        output [31:0] reg_num
    );

        wire [18:0] tmp;
        wire finish, start, SL;
        wire [64:0] Segment;
        wire [63:0] disp_num;
        wire [31:0] clk_div;

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

        assign finish = Segment[0] & Segment[1] & Segment[2] &
Segment[3] & Segment[4] & Segment[5] &Segment[6] & Segment[7]
```

```
                            & Segment[8] & Segment[9] &
Segment[10] & Segment[11] & Segment[12] & Segment[13] &
Segment[14] & Segment[15]
                            &Segment[16] & Segment[17]
&Segment[18] & Segment[19] & Segment[20] & Segment[21] &
Segment[22] & Segment[23]
                            & Segment[24] &Segment[25] &
Segment[26] &Segment[27] & Segment[28] & Segment[29] &
Segment[30] & Segment[31]
                            & Segment[32] & Segment[33]
&Segment[34] & Segment[35] &Segment[36] & Segment[37] &
Segment[38] & Segment[39]
                            & Segment[40] & Segment[41] &
Segment[42] &Segment[43] & Segment[44] &Segment[45] &
Segment[46] & Segment[47]
                            & Segment[48] & Segment[49] &
Segment[50] & Segment[51] &Segment[52] & Segment[53]
&Segment[54] & Segment[55]
                            & Segment[56] & Segment[57] &
Segment[58] & Segment[59] & Segment[60] &Segment[61] &
Segment[62] &Segment[63];

        clkdiv d0(.clk(clk), .rst(1'b0), .clkdiv(clk_div));

        Regtrans4b
m0(.clk(clk), .SW1(SW[0]), .SW2(SW[14]),.num(reg_num[3:0]));
        Regtrans4b
m1(.clk(clk), .SW1(SW[1]), .SW2(SW[14]),.num(reg_num[7:4]));
        Regtrans4b
m2(.clk(clk), .SW1(SW[2]), .SW2(SW[14]),.num(reg_num[11:8]));
        Regtrans4b
m3(.clk(clk), .SW1(SW[3]), .SW2(SW[14]),.num(reg_num[15:12]));
        Regtrans4b
m4(.clk(clk), .SW1(SW[4]), .SW2(SW[14]),.num(reg_num[19:16]));
        Regtrans4b
m5(.clk(clk), .SW1(SW[5]), .SW2(SW[14]),.num(reg_num[23:20]));
        Regtrans4b
m6(.clk(clk), .SW1(SW[6]), .SW2(SW[14]),.num(reg_num[27:24]));
        Regtrans4b
m7(.clk(clk), .SW1(SW[7]), .SW2(SW[14]),.num(reg_num[31:28]));

        SLReg9b
m8(.clk(clk), .S_L(SL) ,.s_in(1'b1),.p_in({disp_num[7:0] ,
1'b0}), .Q(Segment[8:0]));
```

```
        SLReg8b
m9(.clk(clk), .S_L(SL) ,.s_in(Segment[8]),.p_in({disp_num[15:8
]}), .Q(Segment[16:9]));
        SLReg8b
m10(.clk(clk), .S_L(SL) ,.s_in(Segment[16]),.p_in({disp_num[23
:16]}), .Q(Segment[24:17]));
        SLReg8b
m11(.clk(clk), .S_L(SL) ,.s_in(Segment[24]),.p_in({disp_num[31
:24]}), .Q(Segment[32:25]));
        SLReg8b
m12(.clk(clk), .S_L(SL) ,.s_in(Segment[32]),.p_in({disp_num[39
:32]}), .Q(Segment[40:33]));
        SLReg8b
m13(.clk(clk), .S_L(SL) ,.s_in(Segment[40]),.p_in({disp_num[47
:40]}), .Q(Segment[48:41]));
        SLReg8b
m14(.clk(clk), .S_L(SL) ,.s_in(Segment[48]),.p_in({disp_num[55
:48]}), .Q(Segment[56:49]));
        SLReg8b
m15(.clk(clk), .S_L(SL) ,.s_in(Segment[56]),.p_in({disp_num[63
:56]}), .Q(Segment[64:57]));

        Decoder
m16(.hex(reg_num[3:0]), .Segment(disp_num[7:0]));
        Decoder
m17(.hex(reg_num[7:4]), .Segment(disp_num[15:8]));
        Decoder
m18(.hex(reg_num[11:8]), .Segment(disp_num[23:16]));
        Decoder
m19(.hex(reg_num[15:12]), .Segment(disp_num[31:24]));
        Decoder
m20(.hex(reg_num[19:16]), .Segment(disp_num[39:32]));
        Decoder
m21(.hex(reg_num[23:20]), .Segment(disp_num[47:40]));
        Decoder
m22(.hex(reg_num[27:24]), .Segment(disp_num[55:48]));
        Decoder
m23(.hex(reg_num[31:28]), .Segment(disp_num[63:56]));

        SR_LATCH m24(.S(start & finish), .R(~finish),.Q(SL));
        Load_Gen
m25(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[15]), .Load_o
ut(start));
endmodule
```

(e)建立 Top_sim 仿真文件，根据仿真要求：

- 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

输入代码如下：

```
`timescale 1ns / 1ps
module SEGP2S_sim2;
    reg clk;
    reg [15:0] SW;

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

    SEGP2S uut (
        .clk(clk),
        .SW(SW),
        .SEG_CLK(SEG_CLK),
        .SEG_CLR(SEG_CLR),
        .SEG_EN(SEG_EN),
        .SEG_DT(SEG_DT),
        .num(num),
        .reg_num(reg_num)
    );

    integer i;
    initial begin
        clk = 0;
        SW = 0;
        #100;

        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;
        SW[7] = 1; #25;
        SW[7] = 0; #25;
        SW[7] = 1; #25;
        SW[7] = 0; #25;
        SW[5] = 1; #25;
        SW[5] = 0; #25;

        for(i = 0; i < 2; i = i + 1)
        begin
            SW[3] = 1; #25;
            SW[3] = 0; #25;
        end
        for(i = 0; i < 5; i = i + 1)begin
            SW[2] = 1; #25;
            SW[2] = 0; #25;
        end
        for(i = 0; i < 3; i = i + 1)
        begin
            SW[1] = 1; #25;
            SW[1] = 0; #25;
        end
        for(i = 0; i < 5; 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
```

(f)得到仿真波形图如下：

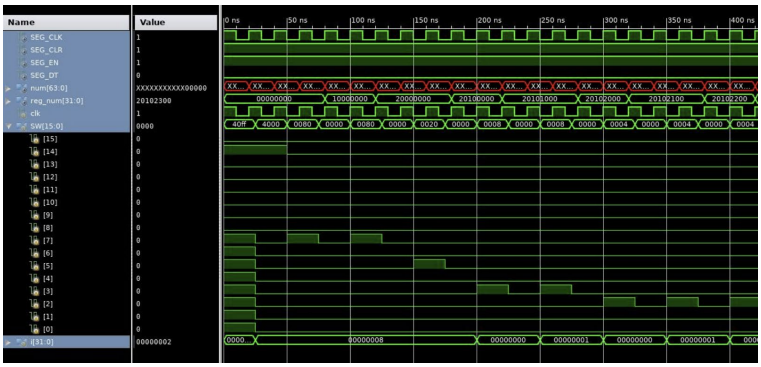

图 3.7 仿真波形图

(g)建立 K7.ucf 引脚约束文件，输入代码如下：

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
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 ;

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[14]" LOC = AF13 | IOSTANDARD = LVCMOS15;
NET "SW[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
```

(h)生成 bit 文件，下载到实验板上进行验证：

上下拨动开关显示学号后八位：

图 3.8 学号后 8 位

四、实验结果分析

本次实验通过仿真并使用 SWORD 板进行验证，实验结果符合预期。

在任务二实验过程中，我们组首次写出来完整的模块以及 K7 文件并上板验证，但是发现没有添加防抖动，然后添加之后发现 Generate 过程中 Translate 发生 error，最后修改 clk 才得以修复。

五、讨论与心得

本次实验多亏了老师的讲解，在按键防抖的过程中需要去掉 1ms 的引脚，因为时间太短，扫描得太快，这个地方卡了好长时间。和同学们讨论之后发现需要改变 Load_Gen 模块中的 btn_out 改为 in，否则 num 就一直为 0（这个模块改了好多次啊），最后在仿真的时候波形才正确。

感谢老师、助教以及本组同学的帮助，本次实验顺利完成！