

# 浙江大学

## 本科实验报告

课程名称:	数字逻辑设计
姓 名:	NoughtQ
学 院:	计算机科学与技术学院
专 业:	计算机科学与技术
邮 箱:	
QQ 号:	
电 话:	
指导教师:	洪奇军
报告日期:	2024 年 5 月 30 日

# 浙江大学实验报告

课程名称：\_\_\_\_数字逻辑设计\_\_\_\_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：\_\_\_\_钱梓洋\_\_\_\_学号：\_\_\_\_3230103502\_\_\_\_同组学生姓名：\_\_\_\_官欣\_\_\_\_

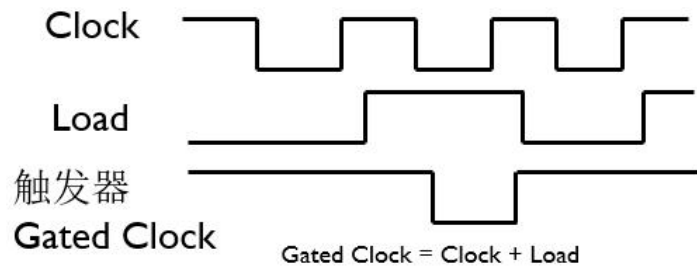
实验地点：\_\_\_\_紫金港东四 511 室\_\_\_\_实验日期：\_\_\_\_2024\_\_\_\_年\_\_\_\_5\_\_\_\_月\_\_\_\_16\_\_\_\_日

## 一、操作方法与实验步骤

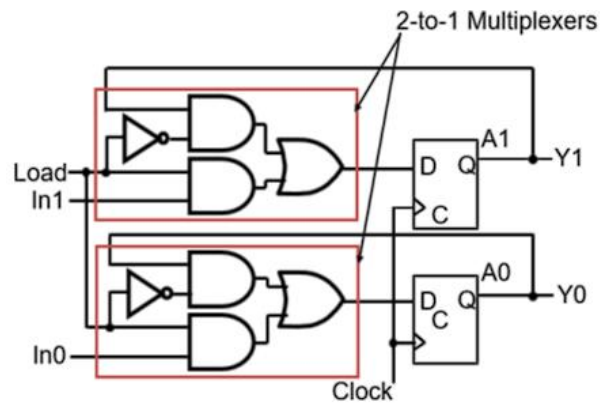
### 1.1 实验原理

**寄存器**是一组二进制存储单元，一个寄存器可以用于存储多位二进制值，通常用于进行简单数据存储、移动和处理等操作。最基本的操作为**读和写**，其中写操作通常在时钟边沿时将接受的输入值存储在触发器中。

采用**门控时钟的寄存器**指通过一个或多个控制信号控制门电路的打开和关闭，只有在门控打开时才能对寄存器的值进行修改，一个简单的实现是使用一个信号 Load 作为门控信号，Load 与时钟信号的与为门控时钟信号，如下图：

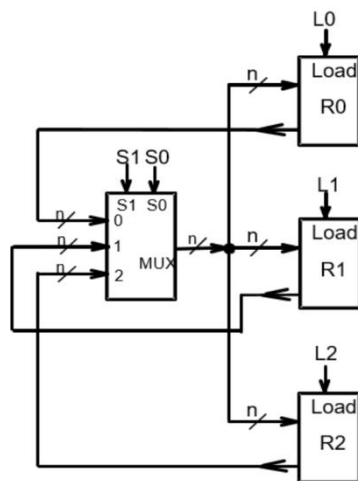


一个更稳定的方式是“采用 Load 控制反馈的寄存器”，它不对时钟信号进行处理，而是在触发器的输入端使用 2-1 多路选择器从**触发器的值**与**输入的值**之间进行选择，原理图如下（存储 2 位数据的寄存器）：

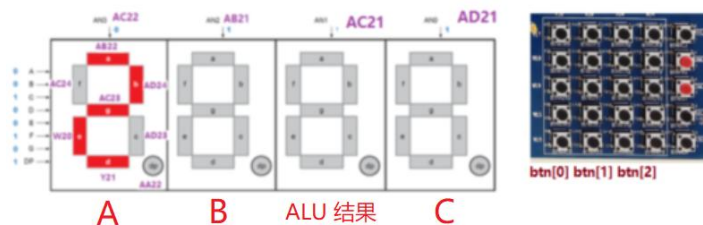


## 数据传输应用

本次实验使用多路选择器总线进行寄存器传输，其模式图如下，寄存器的值“汇总”到一个多路选择器并通过选择信号选择一个值作为三个寄存器的输入，当 Load 信号升起时对对应寄存器的值进行修改：



使用 Arduino 板上的七段数码管显示结果，显示顺序如下：



需要实现的功能如下：

- $SW[15] = 0$  ALU 运算输出模式
- $SW[0]$  控制 A 的增/减； $SW[1]$  控制 B 的增/减

- SW[3:2] 控制 ALU 运算类型
- 按下 btn[0] 用 A 自增/自减的值更新 A
- 按下 btn[1] 用 B 自增/自减的值更新 B
- 按下 btn[2] 用 ALU 运算结果更新 C
- SW[15] = 1 数据传输控制
  - SW[5:4] 传输选择信号, 00 选择 A, 01 选择 B, 10 选择 C, 11 选择常数 0
  - btn[0], btn[1], btn[2] 分别为三个寄存器 A, B, C 的 load 信号。如在按下 btn[0] 时用当前总线上数据对 A 的值进行更新

## 1.2 实验步骤

- (1) 在 Vivado 中新建工程 MyALUTrans。
- (2) 根据提供的顶层模块框架, 根据自身情况完善代码, 最终代码如下:

```
module Top(
    input clk,
    input [3:0] BTN_Y,
    input [15:0] SW,
    // output BTN_X,
    output [3:0] AN,
    output [7:0] SEGMENT,
    output wire seg_clk,
    output wire seg_clrn,
    output wire seg_sout,
    output wire SEG_PEN
);

    wire [31:0] my_clkdiv;
    wire [2:0] btn_out;
    reg [11:0] num;
    wire [3:0] A1, A2, B1, B2, C1, C2; // C1 maybe useless
    wire [3:0] mux_out;
    wire Co;
    wire [3:0] ALU_res;

    /* SW[1:0] to control if the counter for A or B is reversal */
    wire A_Ctrl = SW[0];
    wire B_Ctrl = SW[1];
    /* SW[3:2] to choose the mode of the ALU */
    wire [1:0] ALU_Ctrl = SW[3:2];
    /* SW[5:4] to choose from A B C and 0 */
```

```

/* 00 for A; 01 for B; 10 for C; 11 for 0 */
wire [1:0] Trans_select = SW[5:4];

wire [3:0] reg_A_val = num[ 3: 0];
wire [3:0] reg_B_val = num[ 7: 4];
wire [3:0] reg_C_val = num[11: 8];

// assign BTN_X = 1'b0;

clkdiv m0(.clk(clk), .rst(1'b0), .div_res(my_clkdiv));

pbdebounce
m1(.clk(my_clkdiv[17]), .button(BTN_Y[0]), .pbreg(btn_out[0]));
pbdebounce
m2(.clk(my_clkdiv[17]), .button(BTN_Y[1]), .pbreg(btn_out[1]));
pbdebounce
m3(.clk(my_clkdiv[17]), .button(BTN_Y[2]), .pbreg(btn_out[2]));

AddSub4b m4(.A(reg_A_val), .B(4'b0001), .Ctrl(A_Ctrl), .S(A1));
AddSub4b m5(.A(reg_B_val), .B(4'b0001), .Ctrl(B_Ctrl), .S(B1));

Mux4to1b4
m6(.I0(reg_A_val), .I1(reg_B_val), .I2(reg_C_val), .I3(4'b0000),
    .S(Trans_select), .O(mux_out));

/* ALU module implemented in Lab8 */
/* A/B      : operands */
/* S        : select the operation on ALU */
/* C        : result of ALU */
/* Co       : Carry bit */
ALU
m7(.A(reg_A_val), .B(reg_B_val), .res(ALU_res), .Cout(Co), .op(ALU_Ctrl));
// (Co) may be useless

DisplayNumber m8(.clk(clk), .hexs({reg_A_val, reg_B_val, ALU_res,
reg_C_val}),
    .LEs(4'b0000), .points(4'b0000), .rst(1'b0), .
AN(AN),
    .SEGMENT(SEGMENT));

/* Your code here */
// SW[15]: 0 for ALU mode, 1 for Trans mode.
assign A2 = (1'b0 == SW[15]) ? A1 : mux_out;
assign B2 = (1'b0 == SW[15]) ? B1 : mux_out;

```

```
assign C2 = (1'b0 == SW[15]) ? ALU_res : mux_out;
```

```
always@(posedge btn_out[0]) num[3:0] = A2;  
always@(posedge btn_out[1]) num[7:4] = B2;  
always@(posedge btn_out[2]) num[11:8] = C2;  
/*****/
```

### SSeg\_Dev

```
m9(.clk(clk),.rst(1'b0),.Start(my_clkdiv[20]),.flash(my_clkdiv[25]),  
    .Hexs({12'h3FF, mux_out, reg_A_val, reg_B_val, ALU_res, reg_C_val}),  
    .Point(8'b00000000), .LES(8'b00000000),.seg_clk(seg_clk),.seg_clrn(s  
eg_clrn),.seg_sout(seg_sout),.SEG_PEN(SEG_PEN));
```

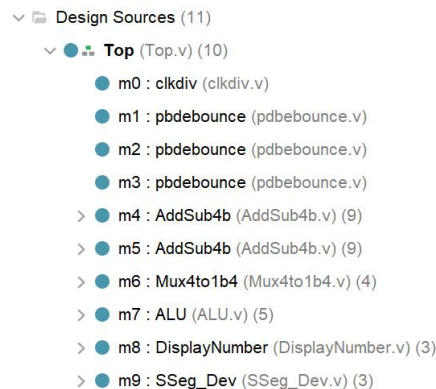
```
endmodule
```

(3) 补充 Top 模块的子模块，包括：

- lab7: 4 位 4-1 多路选择器模块 Mux4to1b4，显示模块 DisplayNumber
- lab8: ALU 模块，去抖动模块 pbdebounce，七段数码管显示模块 SSeg\_Dev

(注: clkdiv 模块被上述模块包含，因此不单独列出)

文件结构如下：



(4) 添加引脚约束文件，代码如下：

```
# Filename: constraints.xdc  
## Constraints file for LabB  
# Main clock  
set_property PACKAGE_PIN AC18 [get_ports clk]  
set_property IOSTANDARD LVCMOS18 [get_ports clk]  
create_clock -period 10.000 -name clk [get_ports "clk"]  
# Switches as inputs  
set_property PACKAGE_PIN AA10 [get_ports {SW[0]}]  
set_property PACKAGE_PIN AB10 [get_ports {SW[1]}]  
set_property PACKAGE_PIN AA13 [get_ports {SW[2]}]  
set_property PACKAGE_PIN AA12 [get_ports {SW[3]}]  
set_property PACKAGE_PIN Y13 [get_ports {SW[4]}]  
set_property PACKAGE_PIN Y12 [get_ports {SW[5]}]
```

```

set_property PACKAGE_PIN AD11 [get_ports {SW[6]}]
set_property PACKAGE_PIN AD10 [get_ports {SW[7]}]
set_property PACKAGE_PIN AE10 [get_ports {SW[8]}]
set_property PACKAGE_PIN AE12 [get_ports {SW[9]}]
set_property PACKAGE_PIN AF12 [get_ports {SW[10]}]
set_property PACKAGE_PIN AE8 [get_ports {SW[11]}]
set_property PACKAGE_PIN AF8 [get_ports {SW[12]}]
set_property PACKAGE_PIN AE13 [get_ports {SW[13]}]
set_property PACKAGE_PIN AF13 [get_ports {SW[14]}]
set_property PACKAGE_PIN AF10 [get_ports {SW[15]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[3]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[4]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[5]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[6]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[7]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[8]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[9]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[10]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[11]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[12]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[13]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[14]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[15]}]
# Key as inputs
set_property PACKAGE_PIN V18 [get_ports {BTN_Y[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[0]}]
set_property PACKAGE_PIN V19 [get_ports {BTN_Y[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[1]}]
set_property PACKAGE_PIN V14 [get_ports {BTN_Y[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[2]}]
# Arduino-Segment & AN
set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]

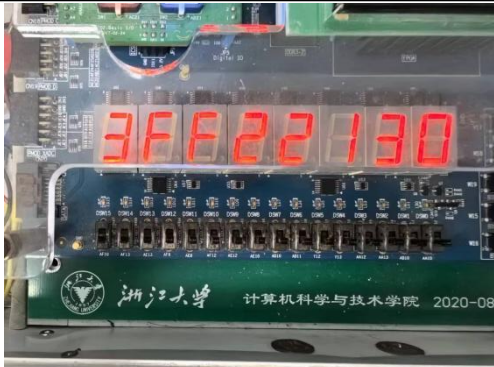
```

```
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]

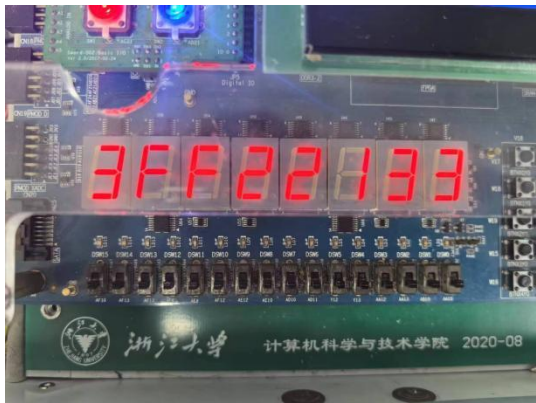

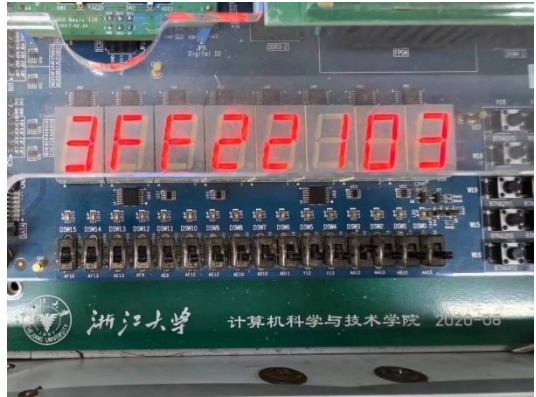

set_property PACKAGE_PIN M24 [get_ports {seg_clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_clk}]
set_property PACKAGE_PIN M20 [get_ports {seg_clrn}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_clrn}]
set_property PACKAGE_PIN L24 [get_ports {seg_sout}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_sout}]
set_property PACKAGE_PIN R18 [get_ports {SEG_PEN}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG_PEN}]
```

(5) 综合运行，烧录 bit 流文件，下板验证。

二、实验结果与分析

图片	说明
	SW[15] = 0, ALU 运算输出模式 SW[3:2] = 00, ALU 做加法 A = 2, B = 1, res = A + B = 3 C 初始为 0 (左边 3 位 3FF 为常数, 占位用)



	<p>SW[15] = 0, ALU 运算输出模式 按下 btn[2], C = res = 3</p>
	<p>SW[15] = 0, ALU 运算输出模式 SW[3:2] = 01, ALU 做减法 res = A - B = 2 - 1 = 1</p>
	<p>SW[15] = 0, ALU 运算输出模式 SW[3:2] = 10, ALU 做按位与 res = A &amp; B = 10 &amp; 01 = 00</p>
	<p>SW[15] = 0, ALU 运算输出模式 SW[3:2] = 11, ALU 做按位或 res = A   B = 10   01 = 11</p>



SW[15] = 0, ALU 运算输出模式  
 SW[3:2] = 01, ALU 做减法  
 SW[0] = 1, 控制 A 自减 1  
 $res = A - B = 1 - 1 = 0$



调整初始状态，方便后续实验结果的讲解



SW[15] = 1, ALU 数据传输控制  
 SW[5:4] = 00, 总线选择 A  
 按下 btn[1],  $B = A = 6$



SW[15] = 1, ALU 数据传输控制  
 SW[5:4] = 00, 总线选择 A  
 按下 btn[2],  $C = A = 6$

### 三、讨论、心得

如果从整个实验的步骤来看，实验并不是很难；然而，这次实验是我目前以来做的最坎坷的一次。由于 Top 框架已经给出，且要求我们填的空并不长，而且引入的子模块大都是以前实验做过的，因此前期工作还算比较顺利地完成了。

但是到了上板验证时，我们却频频受挫：首先，我们花了一些时间找 btn 按钮，结果发现需要同时按同一列上的两个按钮才能改变 A, B 等变量。老师告诉我们要在引脚约束中约束一下按钮位于哪一行，这样才能确定按钮的准确位置。其次，我们在原来顶层模块框架的基础上添加了一个七段数码管的显示子模块，然而它显示的内容竟然是 lab8 实验的内容。结果老师帮我找到了原因：传入板内的 bit 流文件是 lab8 的。虽然我后来改好了，但现在我还是想不通为什么 Vivado 会自动将 lab8 的 bit 流文件作为默认上传的文件，而不是我现在这个工程烧录的文件。最后，虽然 Vivado 没有报错，但是板上只有上面 4 位的 LED 灯亮，SWORD 板上的 8 个数码管都不亮。这里有两个原因：首先 Top.v 中的变量名没有写准确，其次 Top.v 的输出部分忘记补上七段数码管的输出——这些都是细节问题。

回过头来看，整个实验大部分时间都花在了调试上，改了错，错了再改，这种体验十分煎熬；而等到最后显示出正确结果的那一刻，我的内心自然是激动万分的——终于看到了希望！总之，这次实验给了我一个很大的教训：注重细节！

# 浙江大学实验报告

课程名称：\_\_\_\_数字逻辑设计\_\_\_\_实验类型：\_\_\_\_综合\_\_\_\_

实验项目名称：\_\_\_\_计数器、定时器设计与应用\_\_\_\_

学生姓名：\_\_\_\_钱梓洋\_\_\_\_学号：\_\_\_\_3230103502\_\_\_\_同组学生姓名：\_\_\_\_官欣\_\_\_\_

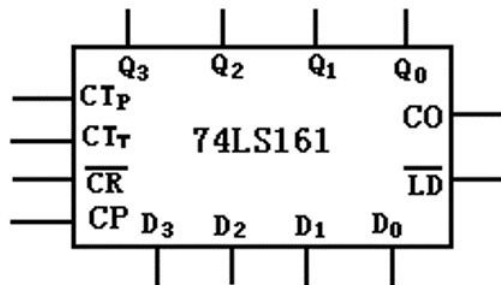
实验地点：\_\_\_\_紫金港东四 511 室\_\_\_\_实验日期：\_\_\_\_2024\_\_\_\_年\_\_\_\_5\_\_\_\_月\_\_\_\_23\_\_\_\_日

## 一、操作方法与实验步骤

### 1.1 实验原理

#### 1.1.1 74LS161

74LS161 芯片具有同步四位二进制计数器功能，其引脚如下：

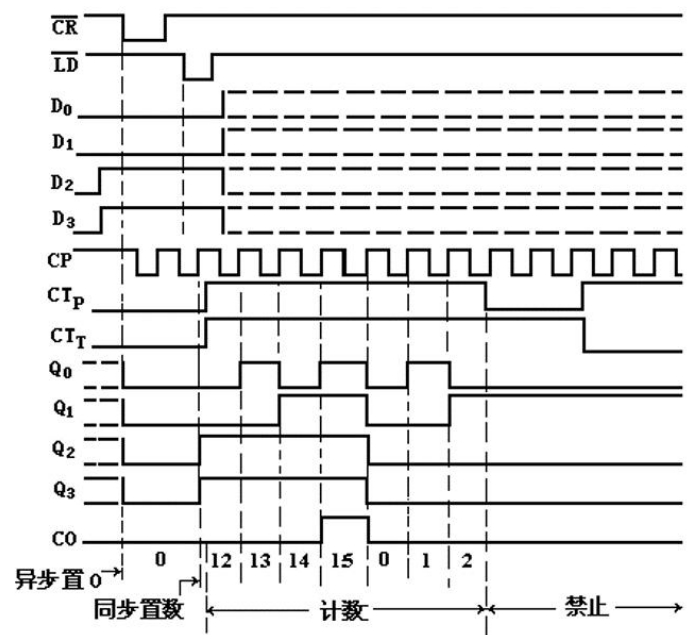


- CP: 接入时钟信号，上升沿触发
- CRn: 清零端，低电平有效，且为异步清零
- LDn: 置数控制端，低电平有效
- D3~D0: 置数数据端，当 LDn 有效时将数据写入
- CTT, CTP: 使能端，两脚均为高电平时启用计数功能，任意一脚为低电平时计数器保持原状态
- Q3~Q0: 数据输出端
- CO: 进位输出端，当输出位均为 1 时置 1

特别关注输出改变时机：**异步清零**意味着不论时钟信号为何当清零端有效时**立即**改变输出为 0；**清零外**的所有输出改变都发生在**时钟上升沿**（包括置数与计数）。其功能表如下：

输 入					输 出				
$\overline{CR}$	$\overline{LD}$	$CT_P$	$CT_T$	$CP$	$D_3 D_2 D_1 D_0$	$Q_3 Q_2 Q_1 Q_0$			
0	×	×	×	×	×	×	×	×	×
1	0	×	×	↑	$d_3 d_2 d_1 d_0$	$d_3 d_2 d_1 d_0$			
1	1	0	1	×	×	×	×	×	保 持
1	1	×	0	×	×	×	×	×	保 持
1	1	1	1	↑	×	×	×	×	计 数

时序图如下：



1.1.2 x 进制计数器

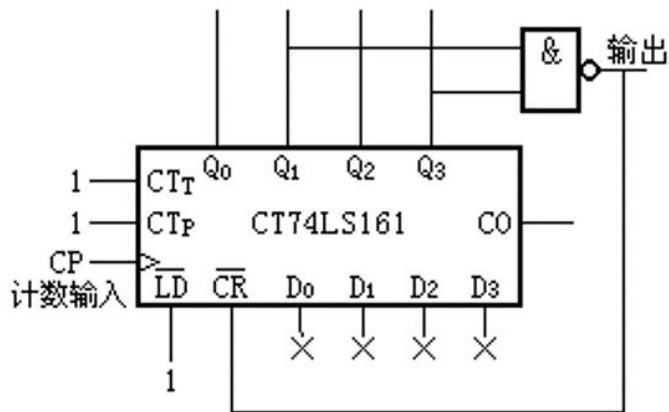
74LS161 提供了十六进制计数器的功能，我们可以通过反馈清零和反馈置位等方法得到任意进制的计数器，本实验中只使用反馈清零的设计方法。

所谓“反馈清零”指的是用输出  $Q_3 \sim Q_0$  的值决定何时清零，即清零端  $\overline{CR}_n$  是关于  $Q_3 \sim Q_0$  的函数。以十进制计数器的设计为例，我们期望输出值为  $0 \sim 9$  十个状态，也就需要在将输出  $10$  时进行清零。十进制数  $10$  的二进制表示

是  $b1010$ ，我们得到清零端为  $\overline{CR} = \overline{Q_3 Q_2 Q_1 Q_0}$ ，注意到输出从  $0$  开始自增，到  $b1010$  时是第一次  $Q_3, Q_1$  同时为  $1$  的状态

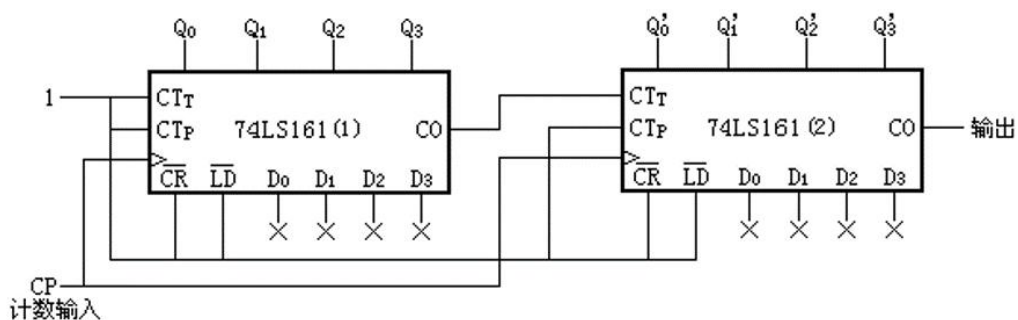
( $b0000 \rightarrow b0001 \rightarrow \dots \rightarrow b1010$ )，因此我们可以简化清零端为  $\overline{CR} = \overline{Q_3 Q_1}$ ，得到了如下电路，其功能为十进制计数器：



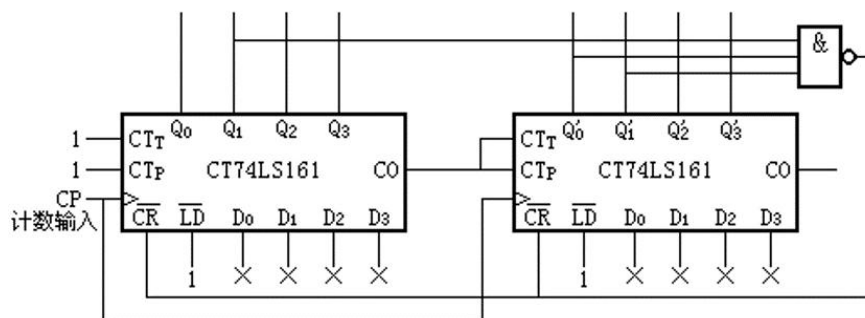


在这里可能有一个疑问“为什么不在输出值为 9 时(b1001)进行清零”，之前提到芯片的复位是**异步**的，与时钟信号无关，在输出变为 10 的时钟上升沿后很短的时间内，复位信号变为有效，进而将输出清零，我们并不去考虑这段很短的时间可能造成的影响。而如果在输出值为 9 时进行清零，则实际得到的是“九进制”计数器。

如果希望拓展计数器的位宽，则需要多个计数器相连，以 **256 进制计数器** 为例，使用两片 74LS161，高 4 位的一个使能端接低 4 位计数器的进位端，在低 4 位输出为 b1111 时进位信号升起，使高位片在下个时钟上升沿时自增。



结合刚刚介绍的两个计数器的设计特点，我们可以设计出任意进制的计数器（在不考虑具体时延影响的前提下），比如一个**五十进制计数器**，其输出状态应为 0~49，在输出应为 50 时(b0011\_0010)进行清零。容易得到其清零端应为高位片的  $Q_1$ ， $Q_0$  与低位片的  $Q_1$  的与非，即  $\overline{CR} = \overline{Q_1' Q_0' Q_1}$ ，同时需要将低位片的进位信号连接到高位片的使能端。



## 1.2 实验步骤

### 1.2.1 采用行为描述设计同步四位二进制计数器 74LS161

(1) 在 Vivado 中新建工程 My74LS161。

(2) 根据提供的代码框架完善代码，最终代码如下：

```

module My74LS161(
    input CP,
    input CRn,
    input LDn,
    input [3:0] D,
    input CTT,
    input CTP,
    output [3:0] Q,
    output CO
);
    reg [3:0] Q_reg = 4'b0;

    always @(posedge CP or negedge CRn) begin
        if(!CRn) begin // reset
            Q_reg <= 4'b0000;
        end
        else if (!LDn) begin
            Q_reg <= D;
        end
        else if(CTT & CTP & LDn) begin //
            Q_reg <= Q_reg + 4'b0001;
        end
    end

    assign Q = Q_reg;
    assign CO = (Q == 4'hF);
endmodule

```

(3) 添加仿真代码，代码如下：

```
`timescale 1ns / 1ps
module sim_74LS161();
    // Inputs
    reg CP;
    reg CRn;
    reg LDn;
    reg [3:0] D;
    reg CTT;
    reg CTP;
    // Output
    wire [3:0] Q;
    wire CO;
    // Instantiate the UUT
    My74LS161 My74LS161_inst (
        .CP(CP),
        .CRn(CRn),
        .LDn(LDn),
        .D(D),
        .CTT(CTT),
        .CTP(CTP),
        .Q(Q),
        .CO(CO)
    );
    integer i;
    initial begin
        assign LDn = 1;
        assign D = 4'b0000;
        CRn = 1;
        for(i = 0; i < 30; i = i + 1)begin
            CTT = 1'b1;
            CTP = 1'b1;
            CP = 1; #10;
            CP = 0; #10;
        end
        CRn = 0;#2;
        CRn = 1;
        for(i = 0; i < 2; i = i + 1)begin
            CTT = 1'b1;
            CTP = 1'b1;
            CP = 1; #10;
            CP = 0; #10;
        end
        for(i = 0; i < 2; i = i + 1)begin
```



```

        CTT = 1'b1;
        CTP = 1'b0;
        CP = 1; #10;
        CP = 0; #10;
    end
    for(i = 0; i < 2; i = i + 1)begin
        CTT = 1'b0;
        CTP = 1'b0;
        CP = 1; #10;
        CP = 0; #10;
    end
end
endmodule

```

(4) 仿真运行，观察生成的波形图是否符合预期。

### 1.2.2 基于 74LS161 设计时钟应用

(1) 在 Vivado 中新建工程 MyClock。

(2) 根据提供的顶层模块框架以及自身情况完善代码，最终代码如下：

```

module top(
    input clk,
    input [1:0] SW,
    output [3:0] AN,
    output [7:0] SEGMENT
);

    wire clk_10ms;
    wire clk_100ms;
    // clk_1s used in LabA
    clk_10ms clk_div_10ms (.clk(clk), .clk_10ms(clk_10ms));
    clk_100ms clk_div_100ms (.clk(clk), .clk_100ms(clk_100ms));

    wire clk_counter = (SW[0] == 1'b0) ? clk_10ms : clk_100ms; // Connect
    this clk_counter to CP-port of 74LS161

    wire [3:0] hour_tens;
    wire [3:0] hour_ones;
    wire [3:0] min_tens;
    wire [3:0] min_ones;

    // Your code here to get the correct HOUR and MINUTE
    wire [3:0] ht,ho,mt,mo;
    My74LS161 m1

```

```

        (.CP(clk_counter),.CRn(1'b1),.LDn(~(min_ones[0]&min_ones[3])),.D(4'b
0),.CTT(1'b1),.CTP(1'b
1),.Q(min_ones));
        My74LS161 m2
        (.CP(clk_counter),.CRn(1'b1),.LDn(~(min_tens[2]&min_tens[0]&min_ones
[0]&min_ones[3])),
        .D(4'b0),.CTT(min_ones[0]&min_ones[3]),.CTP(1'b1),.Q(min_tens));
        My74LS161 m3
        (.CP(clk_counter),.CRn(1'b1),.LDn(~(hour_tens[1]&hour_ones[2])),.D(4
'b0),.CTT(min_tens[2]
&min_tens[0]&min_ones[0]&min_ones[3]),.CTP(1'b1),.Q(hour_ones));
        My74LS161 m4
        (.CP(clk_counter),.CRn(1'b1),.LDn(~(hour_tens[1]&hour_ones[2])),.D(4
'b0),.CTT(hour_ones[0]&hour_ones[3]),
        .CTP(1'b1),.Q(hour_tens));

        assign mo = (1'b1 == SW[1]) ? 4'b0 : min_ones;
        assign mt = (1'b1 == SW[1]) ? 4'b0 : min_tens;
        assign ho = (1'b1 == SW[1]) ? 4'b0011 : hour_ones;
        assign ht = (1'b1 == SW[1]) ? 4'b0010 : hour_tens;

        // Module written in Lab 7
        DisplayNumber display_inst(.clk(clk), .hexs({hour_tens, hour_ones,
min_tens,
min_ones}), .points(4'b0100), .rst(1'b0), .LEs(4'b0000), .AN(AN), .SEGMENT
(SEGMENT));

endmodule

```

(3) 补充顶层模块所需的子模块:

① `clk_10ms`, `clk_100ms` 两个子模块在 lab10 中提供的 `clk_1s` 模块的基础上进行适当修改得到, 代码如下:

• `clk_10ms.v`:

```

`timescale 1ns / 1ps
module clk_10ms(
input clk,
output reg clk_10ms
);
    reg [31:0] cnt;
    initial begin
        cnt = 32'b0;
    end
    wire[31:0] cnt_next;

```

```

    assign cnt_next = cnt + 1'b1;
    always @(posedge clk) begin
        if(cnt<500_000)begin
            cnt <= cnt_next;
        end
        else begin
            cnt <= 0;
            clk_10ms <= ~clk_10ms;
        end
    end
end
endmodule

```

• clk\_100ms.v:

```

`timescale 1ns / 1ps
module clk_100ms(
    input clk,
    output reg clk_100ms
);
    reg [31:0] cnt;
    initial begin
        cnt = 32'b0;
    end
    wire[31:0] cnt_next;
    assign cnt_next = cnt + 1'b1;
    always @(posedge clk) begin
        if(cnt<5_000_000)begin
            cnt <= cnt_next;
        end
        else begin
            cnt <= 0;
            clk_100ms <= ~clk_100ms;
        end
    end
end
endmodule

```

② My74LS161 子模块在本实验第一个任务中已详细介绍，这里就不再赘述。

③ DisplayNumber 子模块在 lab7 中已实现过，且在之前的实验中多次使用过，因此只需将其导入到该工程即可，具体代码略。

(4) 添加引脚约束文件，代码如下：

```

# Filename: constraints.xdc
## Constraints file for LabB
# Main clock

```

```

set_property PACKAGE_PIN AC18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
create_clock -period 10.000 -name clk [get_ports "clk"]
# Switches as inputs
set_property PACKAGE_PIN AA10 [get_ports {SW[0]}]
set_property PACKAGE_PIN AB10 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[1]}]
# Arduino-Segment & AN
set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]

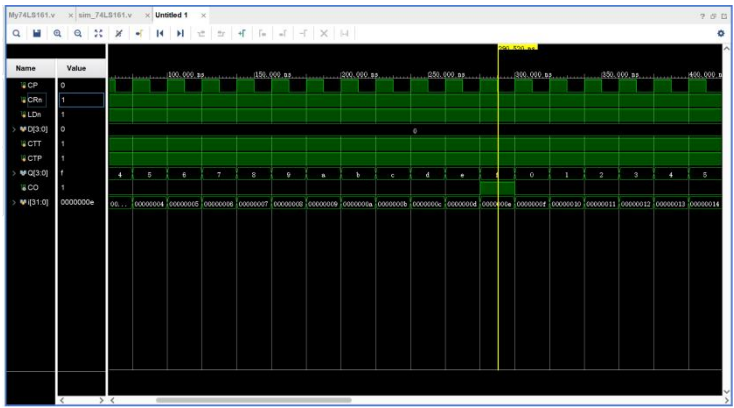
```

(5) 烧录 bit 流文件，下班验证，观察结果是否符合预期。

## 二、实验结果与分析

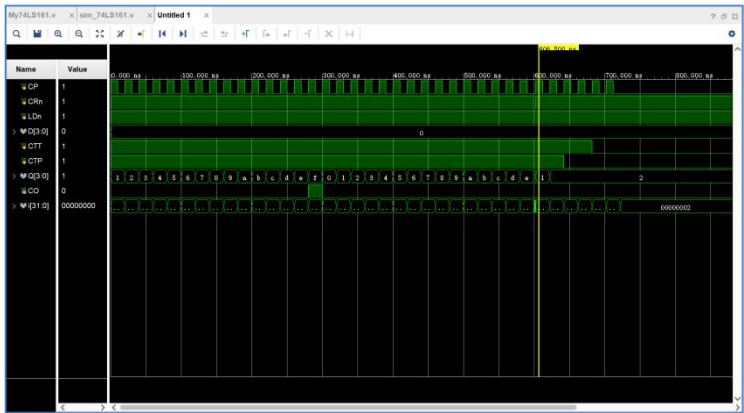
### 1. 采用行为描述设计同步四位二进制计数器 74LS161

(1) 如图黄色竖线处所示，该计数器成功实现进位功能( $CO = 1$ )。



波形图 1

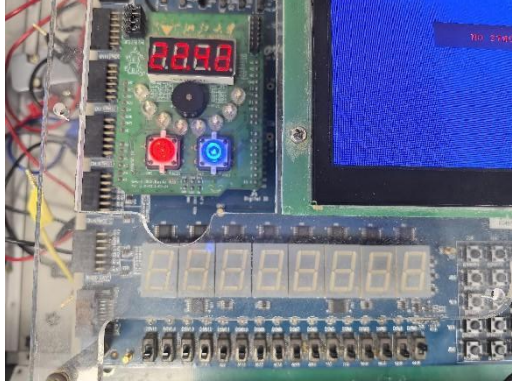
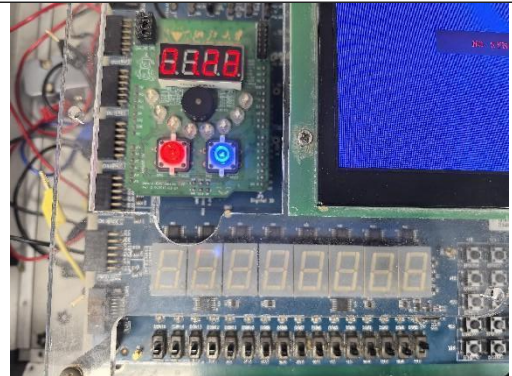
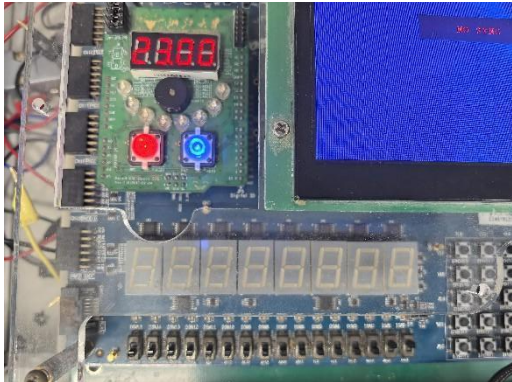
(2) 如图黄色竖线处所示，该计数器成功实现**清零**功能。



波形图 2

## 2. 基于 74LS161 设计时钟应用

图片	说明
	<p>说明</p> <p><math>SW[0] = 0, SW[1] = 0</math></p> <p>此时时钟流速为 10ms，速度较快</p>

	<p><math>SW[0] = 1, SW[1] = 0</math> 此时时钟流速为 <math>100ms</math>，速度较慢</p>
	<p><math>SW[0] = 1, SW[1] = 0</math> 与上一张图对照看，时间在经过 <math>24:00</math> 后能够回到 <math>0:00</math>，并继续计时</p>
	<p><math>SW[0] = 1, SW[1] = 1</math> 时钟重置为 <math>23:00</math></p>

### 三、讨论、心得

相较于上一次接踵而至的磕磕绊绊，这次实验总体上还算比较顺利，尽管稍微遇到了一点小麻烦。个人认为本次实验对我们提出的要求还算比较高的，因为除了要补充 74LS161 模块中的空缺代码外，还要运用这个模块实现时钟应用——这也是最麻烦的部分，因为子模块的参数稍有差错，结果就与预期大径相庭了，而这也正是我们这次实验中遇到的那个“麻烦”。好在试了半天后，终于被我们找出正确的参数了，从而成功实现时钟应用。

现在就快要开始进行大程的设计了，我得吸取这几次实验的经验和教训，尽量避开不该踩的坑；同时要对理论知识和 Verilog 代码具备更深入的了解，为将来更顺利地完完成大程打下坚实基础。

# 浙江大学实验报告

课程名称： 数字逻辑设计 实验类型： 综合

实验项目名称： 计数器、定时器设计与应用

学生姓名： 钱梓洋 学号： 3230103502 同组学生姓名： 官欣

实验地点： 紫金港东四 511 室 实验日期： 2024 年 5 月 30 日

## 一、操作方法与实验步骤

### 1.1 实验原理

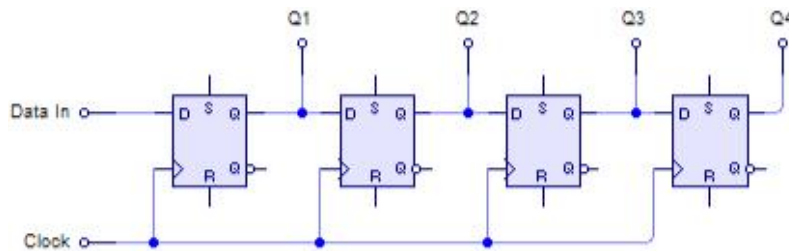
#### 1.1.1 移位寄存器

**移位寄存器(shift register)**是一种在若干相同时钟脉冲下工作的以触发器级联为基础的器件，每个触发器的输出接在触发器链的下一级触发器的“数据”输入端，使得电路在每个时钟脉冲内依次向左或向右移动一个比特，并在输出端进行输出。

根据输入与输出特点，可有破坏性读出的串入串出、串入并出、并入串出等不同功能的移位寄存器。

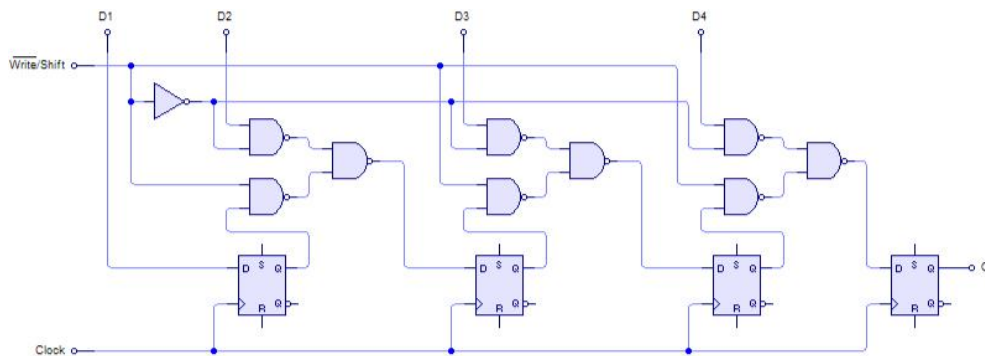
**串入串出**，以 8 位右移移位寄存器为例，每次移位时，数据输入端移入最左边一位触发器的输出中，同时最右边一位触发器的输出移出并丢失。破坏性读出的含义是所有数据在被移位到最右边的位后就会丢失。

**串入并出**，可以将输入的串行数据以并行格式输出。在串行通信要求的几位数据完成输入后就可以在输出端的各位同时读出并行数据。

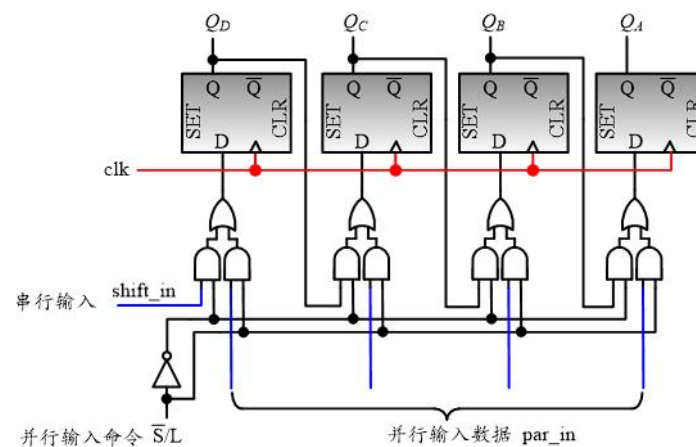




并入串出，可以接收外部并行数据，以最高级位触发器输出作为串行输出位，使用一位控制信号来选择并行读入或串行输出。在写/移位控制线保持低电平时，将并行输入的数据写入寄存器中；在写/移位控制线保持高电平时，进行移位。



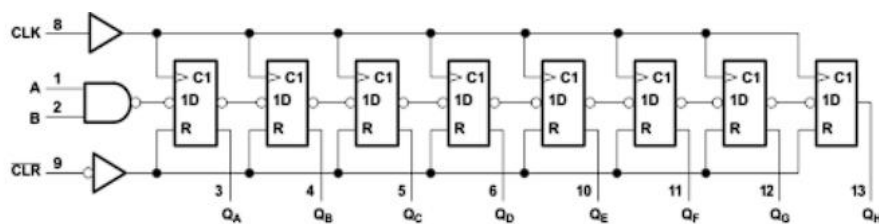
本实验要求的右移移位寄存器功能为：能够并行输入与串行输入功能，使用一个控制信号选择并行输入或串入并出。串行/并行输入信号  $\overline{Shift/Load}$  为低位时进行串入串出，在每个时钟上升沿进行右移同时将串行输入的 `shift_in` 信号移入最左边的触发器中；串行/并行输入信号为高位时进行并行写入，将并行输入端口数据 `par_in` 写入触发器中。



### 1.1.2 74LV164A 芯片功能

（仅作了解即可）74LV164A 功能为 8 位串入并出移位寄存器。清零端 **CLR** 低电平有效，异步重置将触发器的值修改为 0；串入数据端 **A**，**B** 均为高位时输入 1，其他输入下表示输入 0。使用的触发器为正边沿触发。



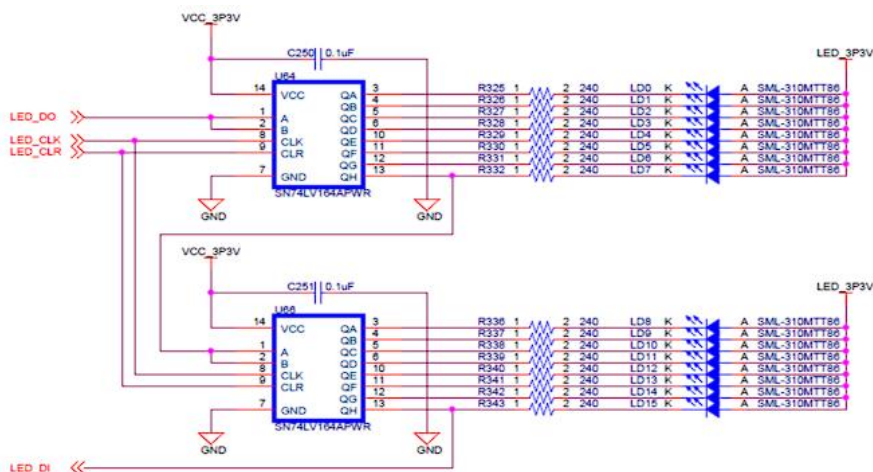


74LS164 芯片功能与 74LV164A 基本相同，使用的技术不同导致电气特性不同，我们不需要关注。

### 1.1.3 主板 LED 与七段数码管

SWORD 板上 LED 与七段数码管均是由若干个 74LV164A 构成的串转并（串行输入、并行输出）模块控制，可以回想在使用板上 LED 与七段数码管时，我们会约束 `s_out`, `s_clk`, `s_clr` 等信号，这其实就是串转并模块需要的串行通信信号。

以 LED 为例，板上共 16 个 LED，需要 16 位信号进行控制，可以使用两个 74LV164A 组成 16 位右移移位寄存器来控制，在使用时，我们需要对 `LED_CLK`, `LED_DO` 等信号进行控制，在 `LED_CLK` 上升沿会发生移位并将 `LED_DO` 信号的值传入寄存器最低位。



七段数码管与 LED 构成相似，区别在于，每个七段数码管都由 8 位信号（七段 + 小数点 = 八个信号）控制，板上共 8 个七段数码管，因此需要 8 个 74LV164A 相连形成 64 位串行输入转并行输出。

## 1.2 实验步骤

### 1.2.1 设计 8 位带并行输入的右移移位寄存器

- (1) 在 Vivado 中新建工程 ShiftReg8。
- (2) 根据提供的接口和要求功能，书写移位寄存器的代码，代码如下：

```
module ShiftReg8b(
    input      clk,
    input      shiftn_loadp,
    input      shift_in,
    input [7:0] par_in,
    output[7:0] Q
);
    wire cons;
    wire [7:0] Qn;
    wire [7:0] D;
    genvar k; // Used in generate block
    generate
        for(k = 7; k < 8; k = k + 1) begin
            Inputgate
Inputgate_else(.Dat(shift_in), .par_in(par_in[k]), .SL(shiftn_loadp), .D(
D[k]));
            end
        endgenerate
    genvar i; // Used in generate block
    generate
        for(i = 0; i < 7; i = i + 1) begin
            Inputgate Inputgate_else(.Dat(Q[i +
1]), .par_in(par_in[i]), .SL(shiftn_loadp), .D(D[i]));
            end
        endgenerate
    genvar j;
    generate
        for(j = 0; j < 8; j = j + 1) begin
            FD FD1(.clk(clk), .D(D[j]), .Q(Q[j]), .Qn(Qn[j]));
            end
        endgenerate
endmodule
```

- (3) 补充上述模块中使用到的子模块：

① Inputgate 模块的代码如下：

```
module Inputgate(
    input SL,
    input Dat,
    input par_in,
    output D
);
    assign D = (~SL & Dat) | (SL & par_in);
```

```
endmodule
```

② FD 模块的代码在 labA 中已给出，代码如下：

```
module FD(  
    input clk,  
    input D,  
    output Q,  
    output Qn  
);  
  
    reg Q_reg = 1'b0;  
    always @(posedge clk) begin  
        Q_reg <= D;  
    end  
  
    assign Q = Q_reg;  
    assign Qn = ~Q_reg;  
  
endmodule
```

(3) 添加仿真文件，代码如下：

```
`timescale 1ns / 1ps  
module sim();  
    // Inputs  
    reg clk;  
    reg shiftn_loadp;  
    reg shift_in;  
    reg [7:0] par_in;  
    // Output  
    wire [7:0] Q;  
    // Instantiate the UUT  
    ShiftReg8b ShiftReg8b_tb_inst(  
        .clk(clk),  
        .shiftn_loadp(shiftn_loadp),  
        .shift_in(shift_in),  
        .par_in(par_in),  
        .Q(Q)  
    );  
  
    initial begin  
        clk = 0;  
        assign shift_in = 1;  
        par_in = 8'b10101011;  
        assign shift_in = 1;  
        shiftn_loadp = 1;  
    end  
endmodule
```

```

        #20;
        shiftn_loadp = 0;
        #20;
        par_in = 8'b00000010;
        shiftn_loadp = 1;
        shift_in = 0;
        #20;
        shiftn_loadp = 0;
        #90;
    end
    always begin
        clk <= ~clk;
        #10;
    end
endmodule

```

(4) 进行仿真运行，观察生成的波形图是否符合预期。

### 1.2.2 设计跑马灯应用

- (1) 在 Vivado 中新建工程 MyMarquee。
- (2) 根据要求自行设计顶层模块，代码如下：

```

module Top(
    input wire clk,
    input wire [4:0]SW,
    input wire [1:0]BTN_Y,
    output wire [3:0]AN,
    output wire [7:0]LED,
    output wire BTN_X,
    output wire [7:0]SEGMENT
);

    wire clk_1,S_IN;
    wire [7:0]num;
    wire [31:0]clk_div;
    wire [1:0]btn_out;
    assign S_IN=(SW[4]==0)?SW[3]:LED[0];
    assign BTN_X=0;
    clk_1s m0(clk,clk_1);
    clkdiv m1(clk,0,clk_div);
    CreateNumber m2(btn_out[1:0],num);
    ShiftReg8b
m3(.clk(clk_1),.shiftn_loadp(SW[2]),.shift_in(S_IN),.par_in(num),.Q(LED));
endmodule

```

```

        DisplayNumber
m4(.clk(clk), .hexs({num[7:0],LED}), .points(4'b1), .LEs(4'b0), .rst(1'b0),
  .AN(AN),.SEGMENT(SEGMENT));
    pbdebounce m5(clk_div[17],BTN_Y[0],btn_out[0]);
    pbdebounce m6(clk_div[17],BTN_Y[1],btn_out[1]);

endmodule

```

(3) 补充顶层模块中用到的子模块：

① 之前实验中给出的模块，这里就不再引入其详细代码。

- `clkdiv`: 时钟分频模块, lab7
- `CreateNumber`: 计分模块, lab7
- `DisplayNumber`: 动态扫描模块（自己实现的），lab7
- `pbdebounce`: 去抖动模块, lab8
- `clk_1s`: lab10

② `ShiftReg8b` 模块在本实验的第一个任务中已详细介绍，故这里不再赘述。

(4) 添加引脚约束文件，代码如下：

```

# Filename: constraints.xdc
## Constraints file for LabD
# Main clock
set_property PACKAGE_PIN AC18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
create_clock -period 10.000 -name clk [get_ports "clk"]
# Switches as inputs
set_property PACKAGE_PIN AA13 [get_ports {SW[2]}]
set_property PACKAGE_PIN AA12 [get_ports {SW[3]}]
set_property PACKAGE_PIN Y13 [get_ports {SW[4]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[3]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[4]}]
# Arduino-Segment & AN
set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]

```

```

set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]

```

```

set_property PACKAGE_PIN AF24 [get_ports {LED[0]}]
set_property PACKAGE_PIN AE21 [get_ports {LED[1]}]
set_property PACKAGE_PIN Y22 [get_ports {LED[2]}]
set_property PACKAGE_PIN Y23 [get_ports {LED[3]}]
set_property PACKAGE_PIN AA23 [get_ports {LED[4]}]
set_property PACKAGE_PIN Y25 [get_ports {LED[5]}]
set_property PACKAGE_PIN AB26 [get_ports {LED[6]}]
set_property PACKAGE_PIN W23 [get_ports {LED[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]

```

```

set_property PACKAGE_PIN V18 [get_ports {BTN_Y[0]}]
set_property PACKAGE_PIN V19 [get_ports {BTN_Y[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[1]}]
set_property PACKAGE_PIN W16 [get_ports {BTN_X}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_X}]

```

(5) 烧录 bit 流文件，下板验证，观察结果是否符合预期。

### 1.2.3 设计主板 LED 灯驱动模块

- (1) 在 Vivado 中新建工程 MyMarquee。
- (2) 根据提供的代码框架补充 P2S 模块，代码如下：

```

module P2S
#(parameter BIT_WIDTH = 7)(
    input clk,
    input start,
    input[BIT_WIDTH-1:0] par_in,
    output sclk,
    output sclrn,
    output sout,
    output EN
);
    wire[BIT_WIDTH:0] Q;
    wire Qn;
    wire S;
    wire q;
    wire Qn;
    wire consone;
    wire finish;

    assign S = start & finish;
    assign R = ~finish;
    assign consone = 1'b1;

    SR_Latch SR_Latch_1(.S(S), .R(R), .Q(q), .Qn(Qn));
    ShiftReg8b
ShiftReg_1(.clk(clk), .shiftn_loadp(q), .shift_in(consone), .par_in({1'b0
,
    par_in[6:0]}), .Q(Q[7:0]));
    assign finish = Q[7] & Q[1] & Q[2] & Q[3] & Q[4] & Q[5] & Q[6];
    assign EN = !start && finish;
    assign sclk = finish | clk;
    assign sclrn = 1'b1;
    assign sout = Q[0];
endmodule

```

(3) 补充 P2S 模块中用到的子模块:

① SR\_Latch: 模仿 SR 锁存器行为的模块, 代码如下:

```

module SR_Latch(
    input S,
    input R,
    output Q,
    output Qn
);

    reg Q_reg = 1'b0;

```

```

always @(*) begin
    if(!S && R) Q_reg = 1'b0;
    else if(S && !R) Q_reg = 1'b1;
end

assign Q = Q_reg;
assign Qn = ~Q_reg;

endmodule

```

② **ShiftReg8b** 模块在本实验的第一个任务中已详细介绍，故这里不再赘述。

(4) 添加仿真文件，代码如下：

```

`timescale 1ns / 1ps
module sim();
    // Inputs
    reg clk;
    reg start;
    reg [6:0] par_in;
    // Output
    wire sclk;
    wire sclrn;
    wire sout;
    wire EN;
    // Instantiate the UUT
    P2S P2S_inst(
        .clk(clk),
        .start(start),
        .par_in(par_in),
        .sclk(sclk),
        .sclrn(sclrn),
        .sout(sout),
        .EN(EN)
    );
    initial begin
        start = 0;
        clk = 0;
        par_in = 7'b0101011;
        #250
        start = 1;
        #1;
        start = 0;
        #1;
    end
endmodule

```



```

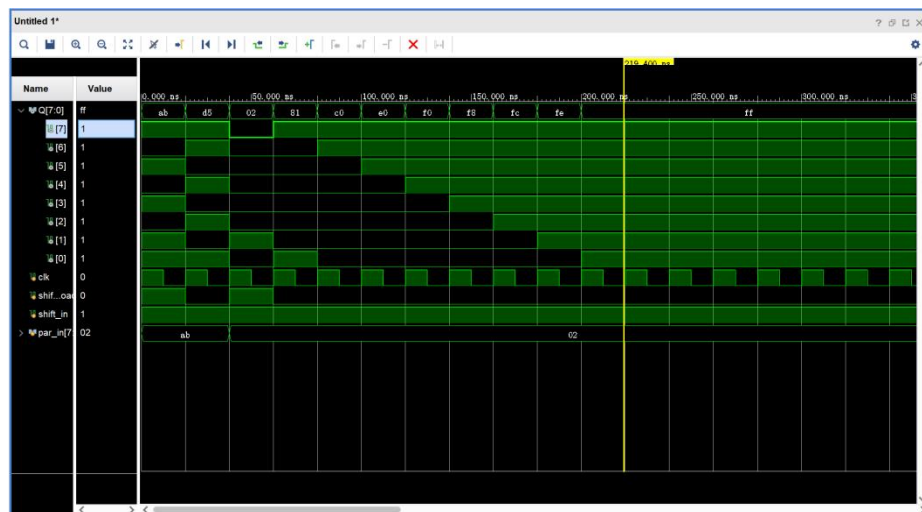
#125;
#125;
par_in = 7'b1010101;
#250;
start = 1;
#1;
start = 0;
#1;
end
always begin
    clk <= ~clk;
    #5;
end
endmodule

```

(5) 仿真运行，观察生成的波形图是否符合预期。

## 二、实验结果与分析

### 1. 设计 8 位带并行输入的右移移位寄存器

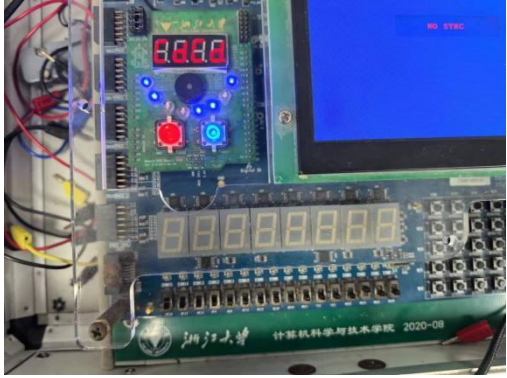
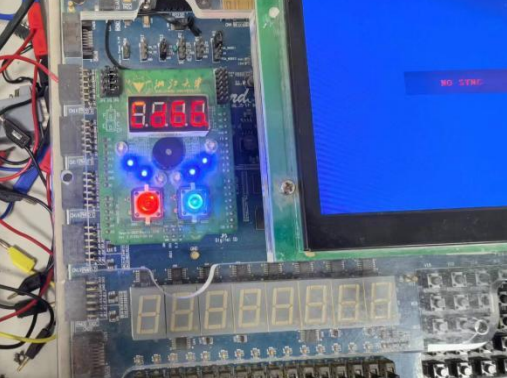
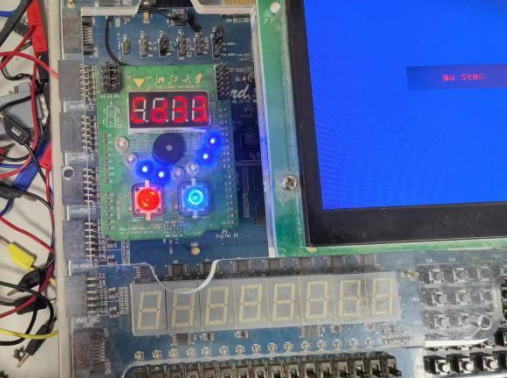



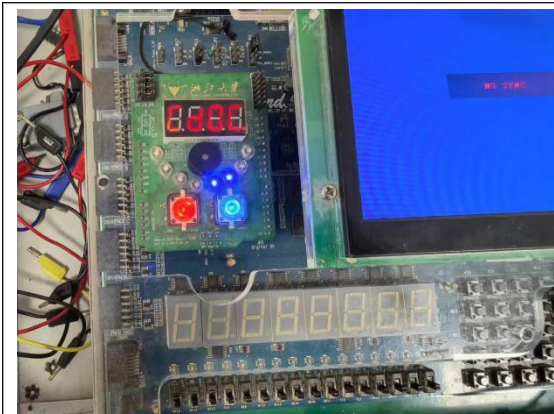
波形图 1

观察生成的波形图，不难看出该移位寄存器具备正常的移位和存储的功能，因而符合预期。

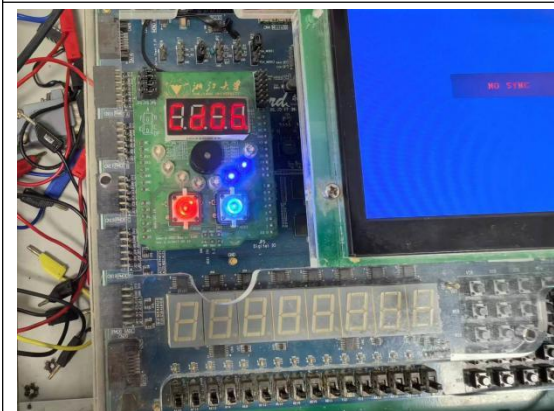
## 2. 设计跑马灯应用

### 2.1 非循环右移位

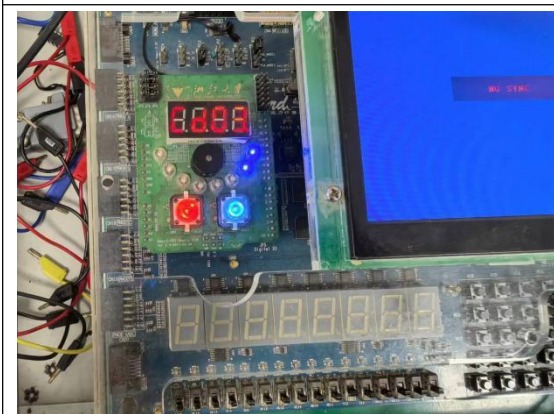
图片	说明
	<p><math>SW[2] = 1</math> 初始显示为: <code>cd(1100_1101)</code> (我们只关心右边 2 位) LED 灯亮表示 1, LED 灯灭表示 0</p>
	<p><code>66(0110_0110)</code> 可以看到最右边的 1 右移出去后就被抛弃, 后面以此类推</p>
	<p><code>33(0011_0011)</code></p>
	<p><code>19(0001_1001)</code></p>



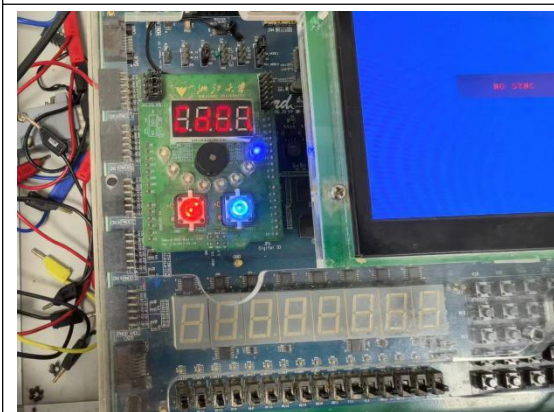
0c(0000\_1100)



06(0000\_0110)

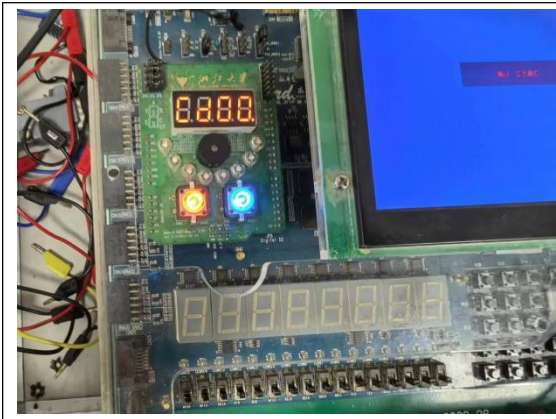


03(0000\_0011)



01(0000\_0001)

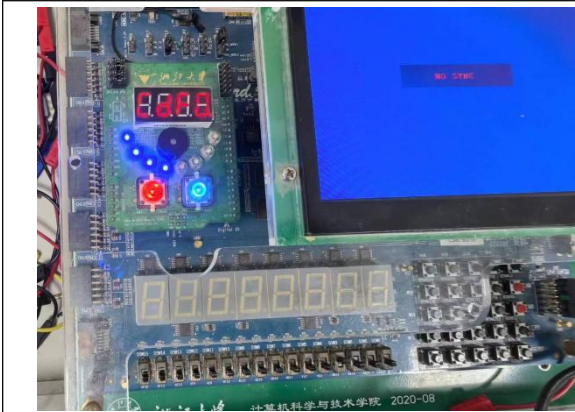




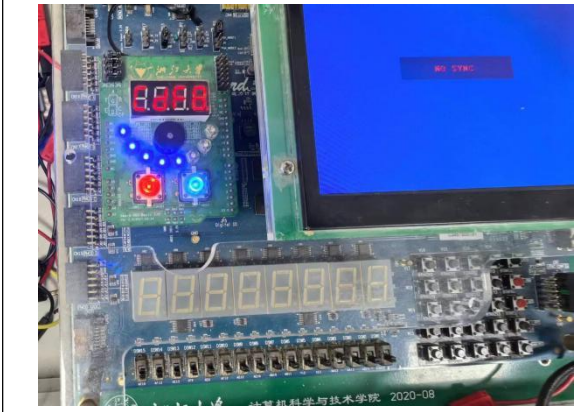
00(0000\_0000)

### 2.2 循环右移位

图片	说明
	<p>SW[3] = 1            初始数据为: 80(1000_0000)            数据将 1 位位向右移, 同时保留前面的位</p>
	<p>c0(1100_0000)</p>
	<p>e0(1110_0000)</p>



f0(1111\_0000)



f8(1111\_1000)



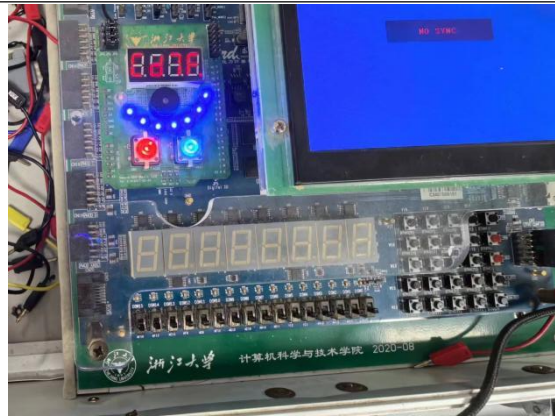
fc(1111\_1100)



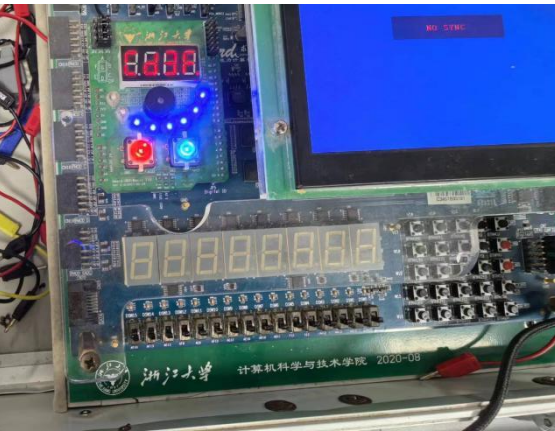
fe(1111\_1110)



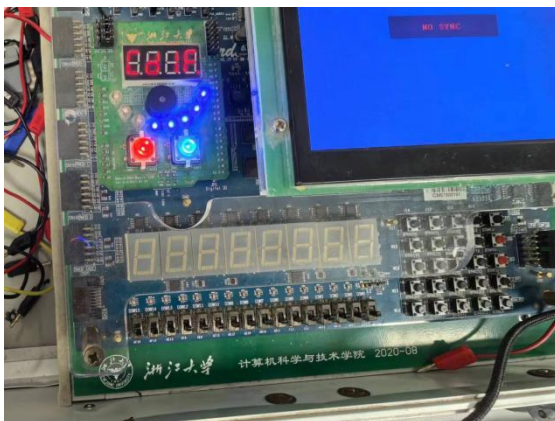
ff(1111\_1111)



SW[3] = 0,  
整个数开始右移  
数据: 7f(0111\_1111)

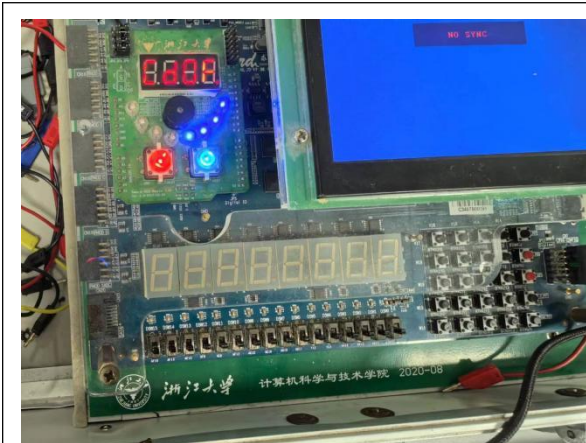


3f(0011\_1111)

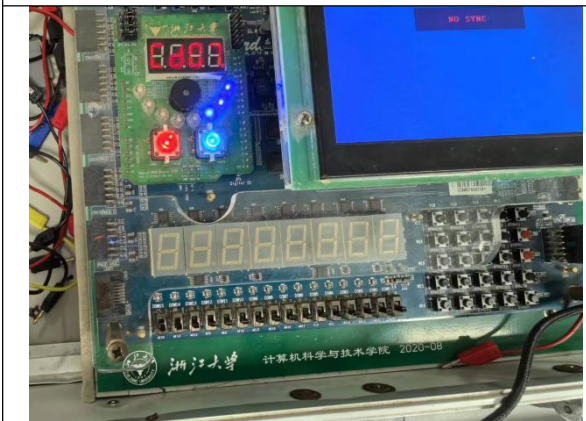


1f(0001\_1111)

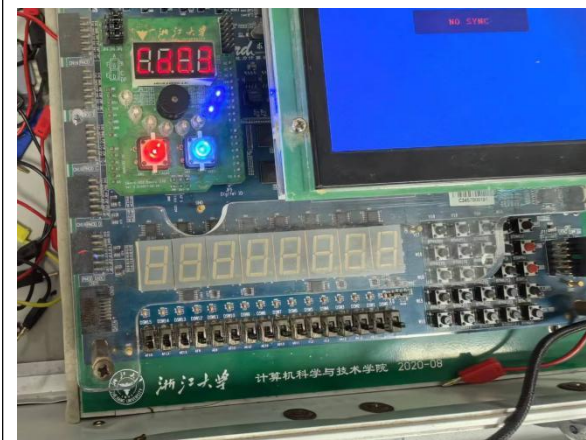




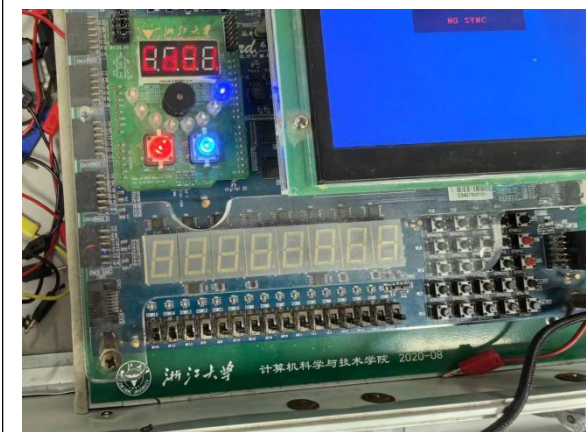
0f(0000\_1111)



07(0000\_0111)

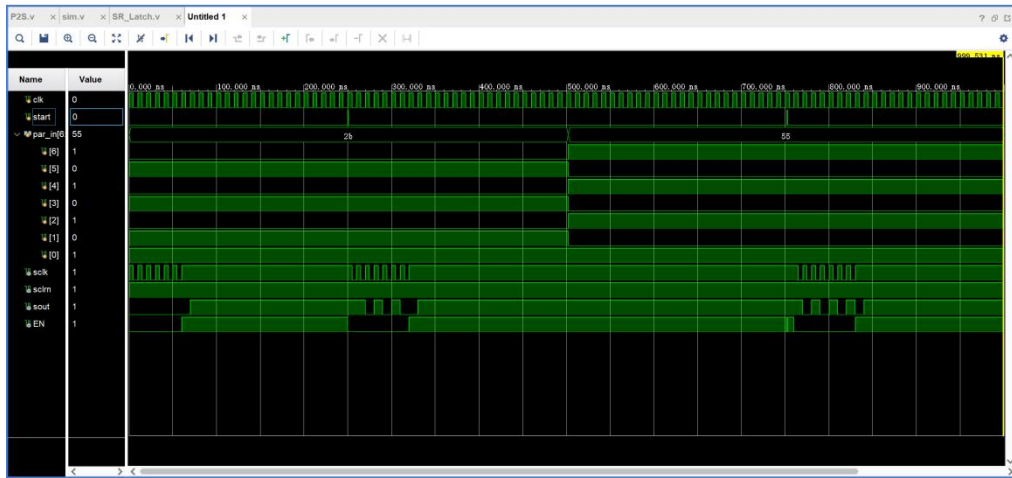


03(0000\_0011)



01(0000\_0001)

### 3. 设计主板 LED 灯驱动模块



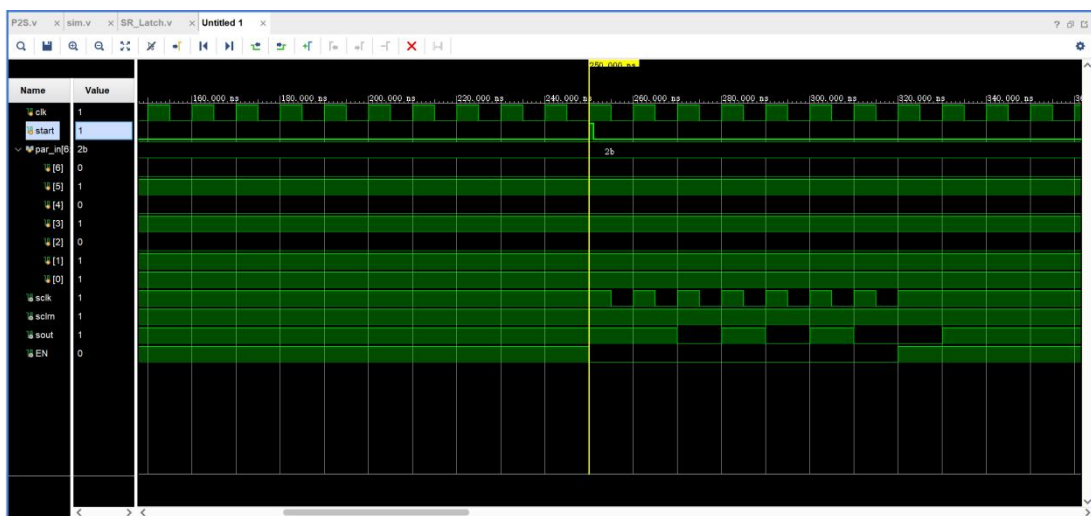
波形图 2

如波形图 2 所示，初始时 `start` 置 0，此时 S-R 锁存器的 `set` 信号一定为 0，根据锁存器当前存储信号 `q` 的值分类讨论：

- `q=0` 表示进行串行输入，即每一个时钟周期移位并补 1，若干周期后 `Q[7]~Q[1]` 均为 1，此时 `finish` 信号置 1，`reset` 信号置 0，锁存器状态保持为 0。

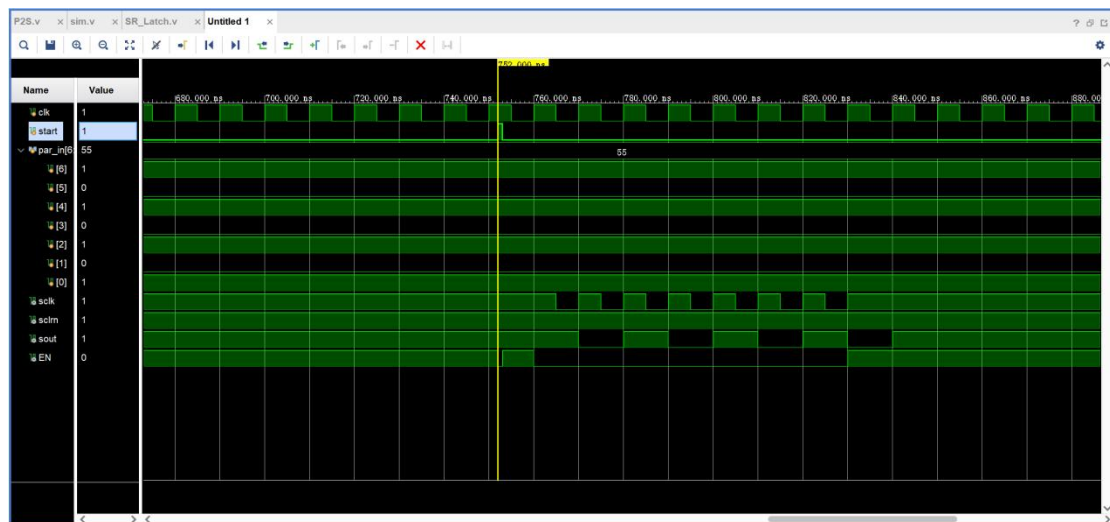
- `q=1` 表示进行并行输入，此时并行输入 7 位脏值，但由于最高位接地一定为 0，`finish` 信号一定为 0，`reset` 信号置 1，锁存器状态改变 `q=0`，后经过一段时间后锁存器状态为 0，`finish` 为 1

初始状态开始一段时间以后，`finish` 信号一定为 1，表示并未进行串行输出，此时模块状态稳定，等待 `start` 信号。



波形图 3





波形图 4

如波形图 3、4 所示：在并行输入的 7 位数据准备好后，start 信号进行一次脉冲（0-1-0），（因为初始状态下 finish 置 1）在 start 置 1 时，S-R 锁存器进行一次 set， $q=1$ ，移位寄存器进行了并行输入  $Q[7:0] = \{1'b0, D[6:0]\}$ ，最高位存在一个 0，因此一定有  $finish=0$ ， $sclk = finish \mid clk$ ，此时 sclk 值和 clk 完全相同，即输出的时间点和移位寄存器进行一次右移输出的时间点相同，sout 每一个时钟周期输出最低位，右移一位，高位补 1，传输结束：传输过程中，高位始终补 1，当并行输入的 7 位全部输出后，当前的 Q 值为 8'b1111\_1110。

finish 置 1，表示串行输出结束，sclk 置 1，不再存在“上升沿”， $q=0$ ，且 set 和 reset 信号均为 0，保持。等待下一个 start 信号，重新传输。

### 三、讨论、心得

不知不觉中，我们完成了最后一次实验——这次实验还是有点难度的：首先，我们需要根据已知的接口和功能来完成整个 8 位移位寄存器的设计；其次，还得利用该模块实现 P2S 模块的设计；最后，需要理解清楚仿真波形图的含义。其实，移位寄存器的设计也不是特别难，因为该模块的主体只需循环调用 2 个子模块，只要想清楚了还是能够比较顺利地实现了。最头疼的部分是对 P2S 的仿真波形图的理解，我愣是看了半天才有了一点头绪。

希望在大程中，我能够利用前面实验中获得的经验，尽量顺利地实现我们的设计。