# 实验十二--寄存器和寄存器传输设计实验报告

姓名：黄炯睿 学号：3170103455 专业：信息安全

课程名称：逻辑与计算机设计基础实验 同组学生姓名：无

实验时间：2018-12-6 实验地点：紫金港东 4-509 指导老师：洪奇军

# 一、实验目的和要求

1.1 掌握寄存器传输电路的工作原理
1.2 掌握寄存器传输电路的设计方法
1.3 掌握 ALU 和寄存器传输电路的综合应用

# 二、实验内容和原理

## 2.1 实验内容

2.1.1 基于 ALU 的数据传输应用设计
2.1.2 寄存器
2.1.3 寄存器传输
2.1.4 基于多路选择器总线的寄存器传输

## 2.2 实验原理

2.2.1 寄存器

一组二进制存储单元

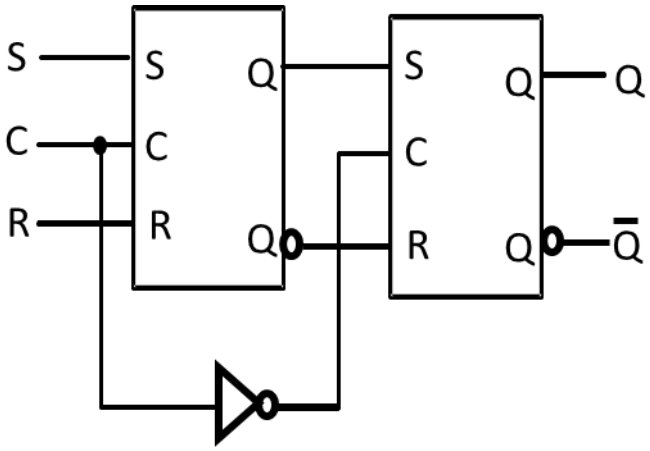
一个寄存器可以用于存储一系列二进制值，通常用于进行简单数据存储、移动和处理等操作

能存储信息并保存多个时钟周期，能用信号来控制“保存”或“加载”信息

2.2.2 采用门控时钟的寄存器

如果 Load 信号为 1，允许时钟信号通过，如果为 0 则阻止时钟信号通过

例如:对于上升沿触发的边沿触发器或负向脉冲触发的主从触发器：



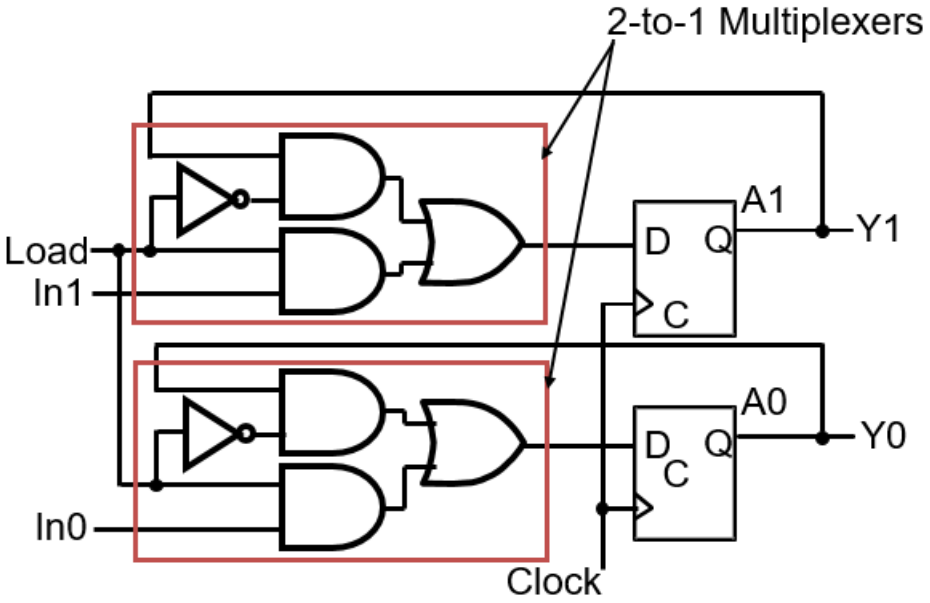

图表 1 上升沿触发器示意图

2.2.3 采用 Load 控制反馈的寄存器

进行有选择地加载寄存器的更可靠方法是：

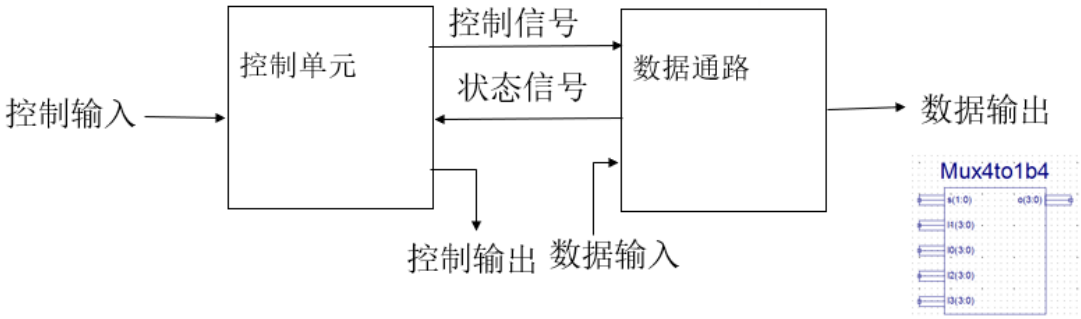
保证时钟的连续性

选择性地使用加载控制来改变寄存器的内容



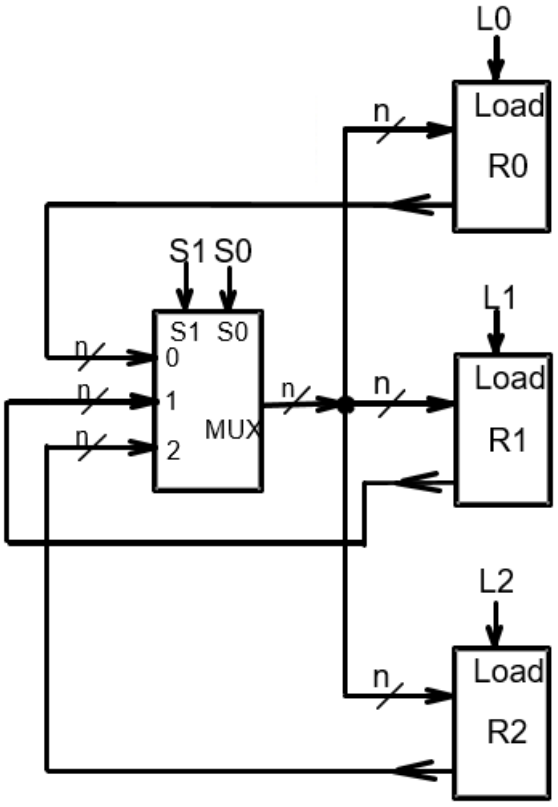

图表 2 采用 Load 控制反馈的寄存器的示意图

2.2.4 寄存器传输：寄存器中数据的传输和处理
三个基本单元：寄存器组、操作、操作控制
基本操作:加载、计数、移位、加法、按位操作等




图表 3 寄存器传输原理

2.2.5 基于多路选择器总线的寄存器传输
1. 由一个多路选择器驱动的总线可以降低硬件开销。
2. 这个结构不能实现多个寄存器相互之间的并行传输操作。

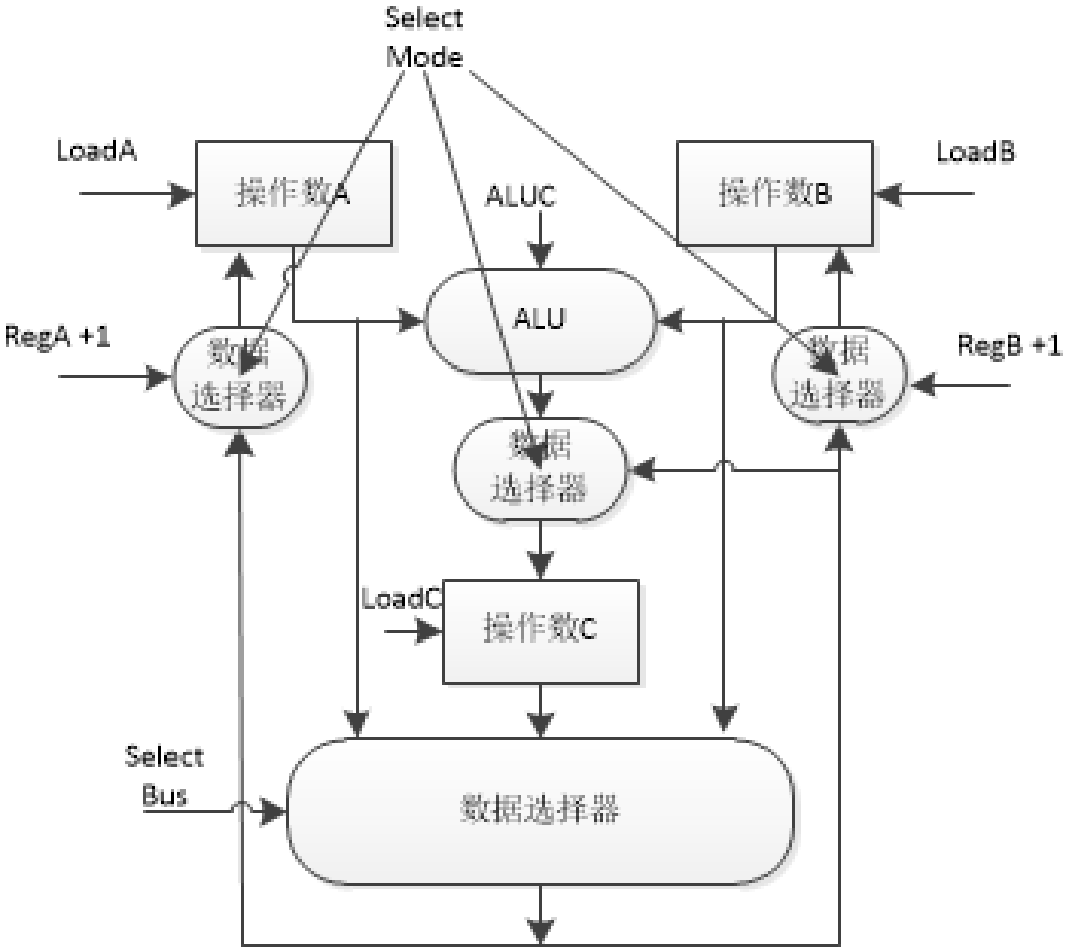



图表 4 基于多路选择器总线的寄存器传输的示意图

2.2.5 寄存器传输应用设计

Mode1 ： ALU 运算输出控制

Mode2 ： 数据传输控制




图表 5 寄存器传输应用设计原理图

# 三、主要仪器设备

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

# 四、操作方法与实验步骤

4.1 基于 ALU 的数据传输应用设计

4.1.1 新建工程，工程名称用 MyALUTrans。

4.1.2 添加如下模块： ALU 模块、4 位 4 选 1 模块、防抖动模块、显示模块 。

4.1.3 新建源文件，类型是 Verilog，文件名称用 Top，右键设为“Set as Top Module”其源代码如下：

```
module Top(
    input wire clk,
    input wire [1:0]SW,//两个操作数的加减
    input wire [1:0]SW2,//determine the ALU operator
    input wire [1:0]busSelect,//选择哪个赋值
    input wire mode,
    input wire BTN_Y[2:0],
    output wire BTN_X,
    output wire [3:0]AN,
    output wire [7:0]SEGMENT
    );
    wire [3:0] Result;
    reg [15:0] num;
    wire [2:0] btn_out;
    wire [3:0] C;
    wire C0;
    wire [31:0] clk_div;
    wire [3:0]A2,B2,C2,A1,B1;
    assign BTN_X=0;
    AddSub4b m0(.A(num[3:0]),.B(4'b0001),.Ctrl(SW[0]),.S(A1));
    AddSub4b m1(.A(num[7:4]),.B(4'b0001),.Ctrl(SW[1]),.S(B1));
    Mux4to14b m2(.I0(num[3:0]),.I1(num[7:4]),.I2(num[11:8]),
                 .I3(4'b0),.s(busSelect),.o(Result));

    assign A2 = (mode==1'b0)?A1:Result;
    assign B2 = (mode==1'b0)?B1:Result;
    assign C2 = (mode==1'b0)?C:Result;

    clkdiv m3(clk,1'b0,clk_div);
    pbdebounce m4(clk_div[17],BTN_Y[0],btn_out[0]);
    pbdebounce m5(clk_div[17],BTN_Y[1],btn_out[1]);
    pbdebounce m6(clk_div[17],BTN_Y[2],btn_out[2]);

    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;



    ALU m7(.A(num[3:0]),.B(num[7:4]),.S(SW2[1:0]),.C(C),.C0(C0));
    disp_num  m8(clk,  {num[3:0],num[7:4],C,num[11:8]},  4'b0,  4'b0,
1'b0, AN,SEGMENT);

endmodule
```

4.1.4 下载验证 UCF 引脚定义：

输入用 6 个开关和 3 个按钮

引脚约束代码如下：

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18 ;
net "mode" loc = AF10      | IOSTANDARD = LVCMOS15;

net "busSelect[0]" loc = AE13      | IOSTANDARD = LVCMOS15;
net "busSelect[1]" loc = AF13    | IOSTANDARD = LVCMOS15;

NET "SW[0]" LOC = AA10      | IOSTANDARD = LVCMOS15;
NET "SW[1]" LOC = AB10      | IOSTANDARD = LVCMOS15;
net "SW[0]" clock_dedicated_route = false;
net "SW[1]" clock_dedicated_route = false;

NET "SW2[0]" LOC = AA13      | IOSTANDARD = LVCMOS15;
NET "SW2[1]" LOC = AA12      | IOSTANDARD = LVCMOS15;

NET "BTN_Y[0]" LOC = V18      | IOSTANDARD = LVCMOS18;
NET "BTN_Y[1]" LOC = V19      | IOSTANDARD = LVCMOS18;
NET "BTN_Y[2]" LOC = V14      | IOSTANDARD = LVCMOS18;
NET "BTN_X" LOC = W16      | IOSTANDARD = LVCMOS18;

NET "SEGMENT[0]" LOC = AB22      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[1]" LOC = AD24      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[2]" LOC = AD23      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[3]" LOC = Y21      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[4]" LOC = W20      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[5]" LOC = AC24      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[6]" LOC = AC23      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[7]" LOC = AA22      | IOSTANDARD = LVCMOS33;

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

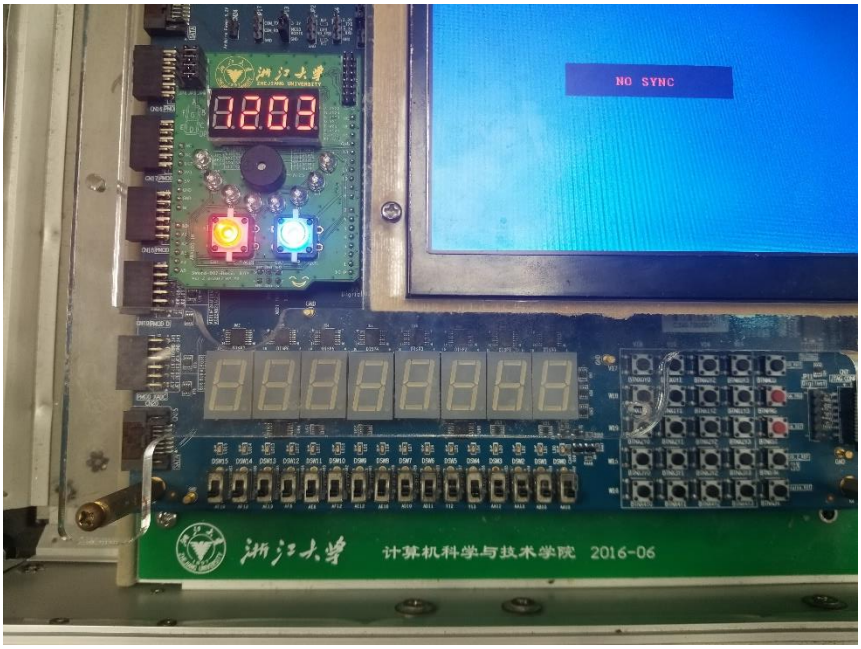
# 五、实验结果与分析

5.1 实验结果（此处的 ALU 计算已在上一个实验报告中演示过了，故本次演示只展示加法）

Mode0：


图中从左到右分别是寄存器 ABC 以及 AB 经过 ALU 算出的答案。

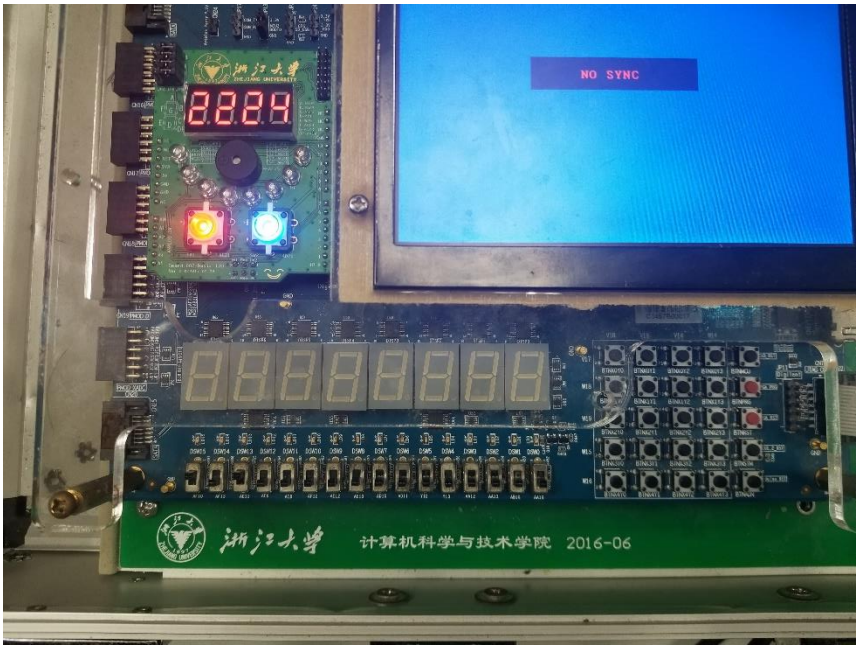
上图中的 ALU 用的是加法，可以看到 1+2 = 3 的结果，而寄存器 C 则还是一个值。



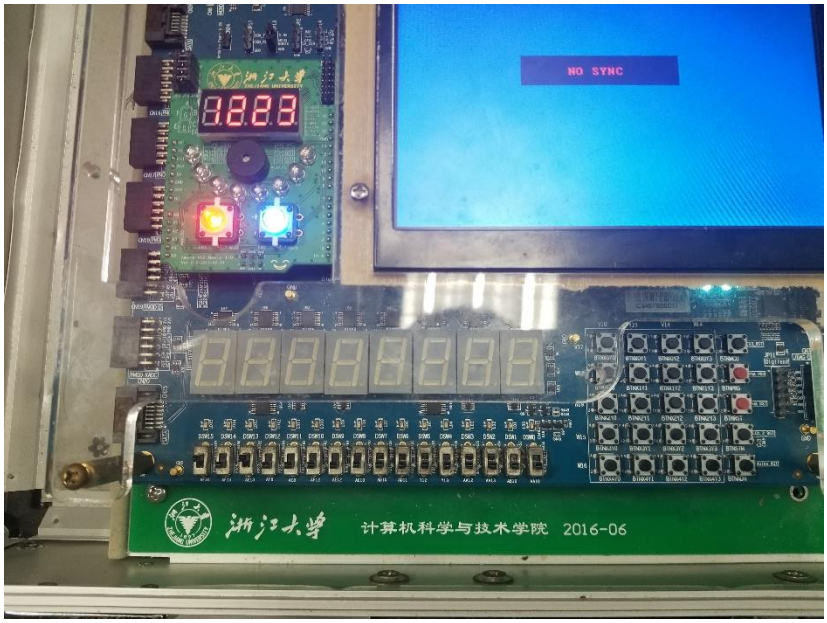

利用对应于 C 寄存器的 button，在开关处选择递减，按下三次按钮后，C 寄存器的值变为 0，因为 AB 都没有变，所以 AB 经过 ALU 算出的结果也没有变，仍为 3.

Mode1：


此时，将最左边对应于 mode 的开关拨上去，并且对应于选择信号的按钮（左数第 2,3 个）都拨为 1，此时按下对应于 ABC 的按钮，可以将 ABC 寄存器置为 0. 因为 AB 寄存器均为 0，此时经过 ALU 算出的答案也为 0.




此时 mode1 仍然是 1，选择开关置为选择 B 寄存器，此时按下对应于 ABC 的按钮，就会将 B 寄存器中的值赋给对应寄存器。如图将 A 寄存器赋值为 2 后，结果数字也发生变化，变为 4.




利用 mode 为 0 时修改寄存器的数据，在改变选择器的开关，这次选择 C 寄存器赋值，上图可看出 B 寄存器拿到了 C 的值，并由此改变了答案的值。

# 六、讨论、心得

这次实验的提示很少，因为我是提前完成实验的，ppt 的信息很混乱，我花了很长时间才知道自己要干什么。这次实验的大量时间花在了了解题意。在写代码本身上，由于一直不明白 createnumber 模块的作用，没有把它删去，因而走了很长的弯路。所幸最后还是在 ppt 的线索中完成了实验。感觉完全由代码写出来的程序真是很棒的体验！

# 实验十三--计数器、定时器设计与应用

姓名：黄炯睿 学号：3170103455 专业：信息安全

课程名称：逻辑与计算机设计基础实验 同组学生姓名：无

实验时间：2018-12-6 实验地点：紫金港东 4-509 指导老师：洪奇军

# 三、实验目的和要求

1.1 掌握同步四位二进制计数器 74LS161 的工作原理和设计方法
1.2 掌握时钟/定时器的工作原理与设计方法

# 四、实验内容和原理

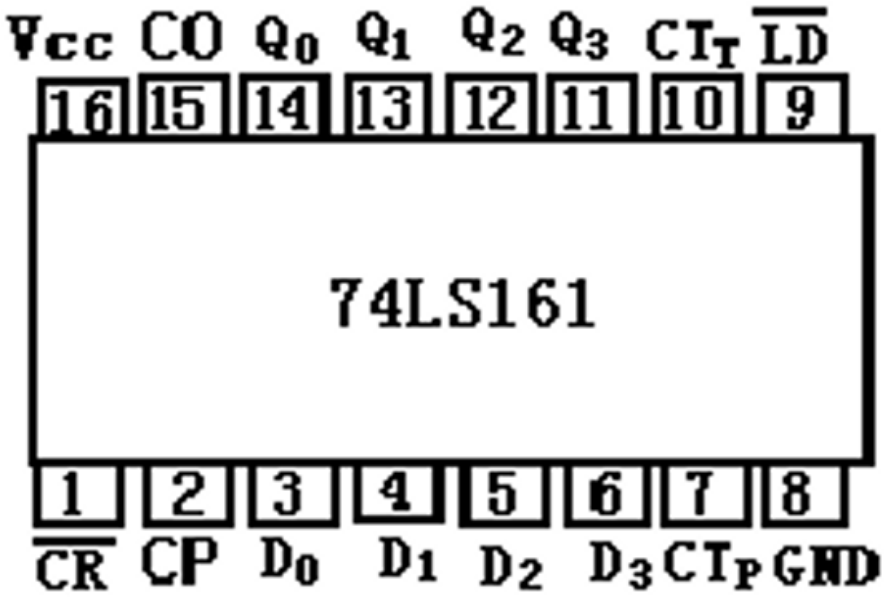
## 2.1 实验内容

2.1.1 采用行为描述设计同步四位二进制计数器 74LS161
2.1.2 基于 74LS161 设计时钟应用

## 2.2 实验原理

2.2.1 同步四位二进制计数器 74LS161

1. 74LS161 是常用的四位二进制可预置的同步加法计数器
2. 可灵活运用在各种数字电路，实现分频器等很多重要的功能




清零端 $\overline{CR}$ 置数端 $\overline{LD}$

使能端 $CT_P$、$CT_T$ 进位输出端 $CO$

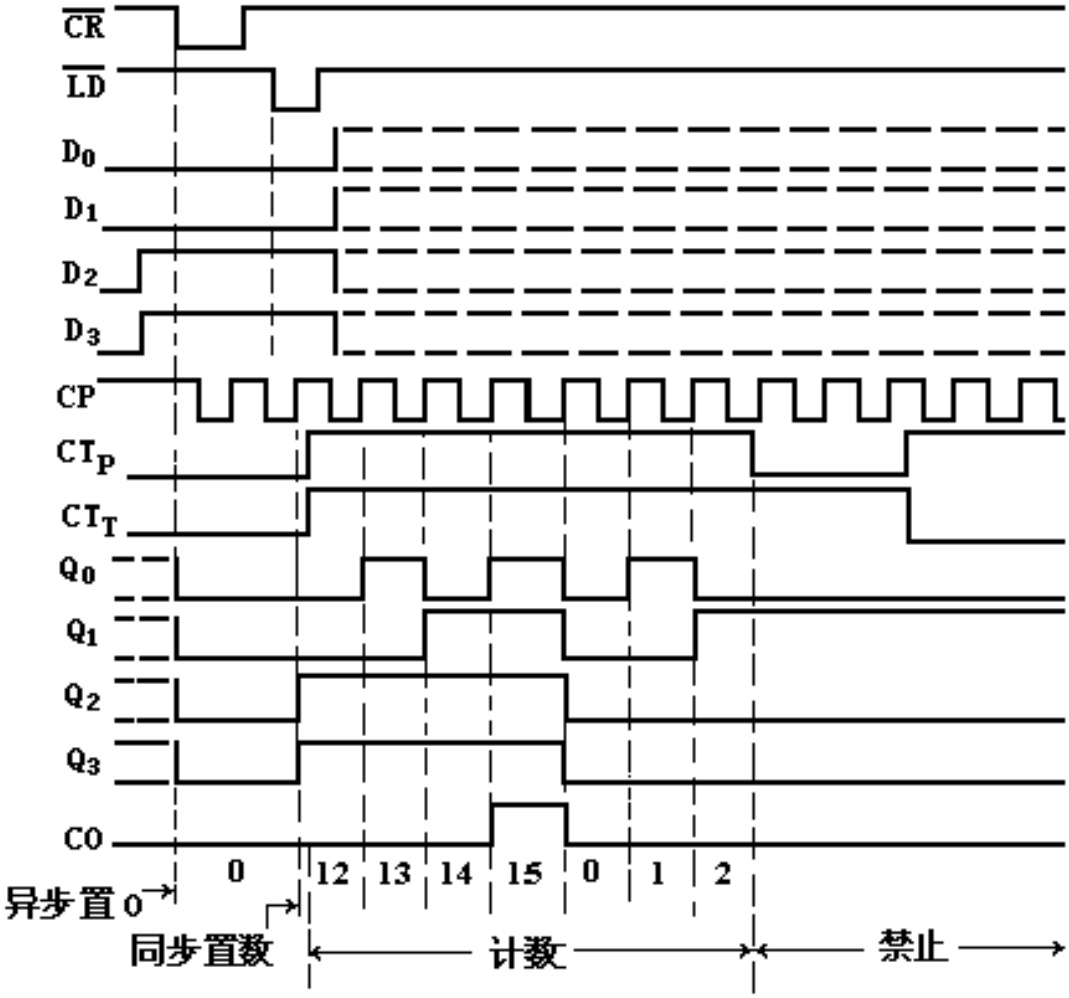
图表 1 74LS161 模块的功能描述图

3. 74LS161 功能表

| 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 | × | × | × | × | × × × × | 0 | 0 | 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 | | | |

图表 2 74LS161 模块的功能表

4. 74LS161 时序图



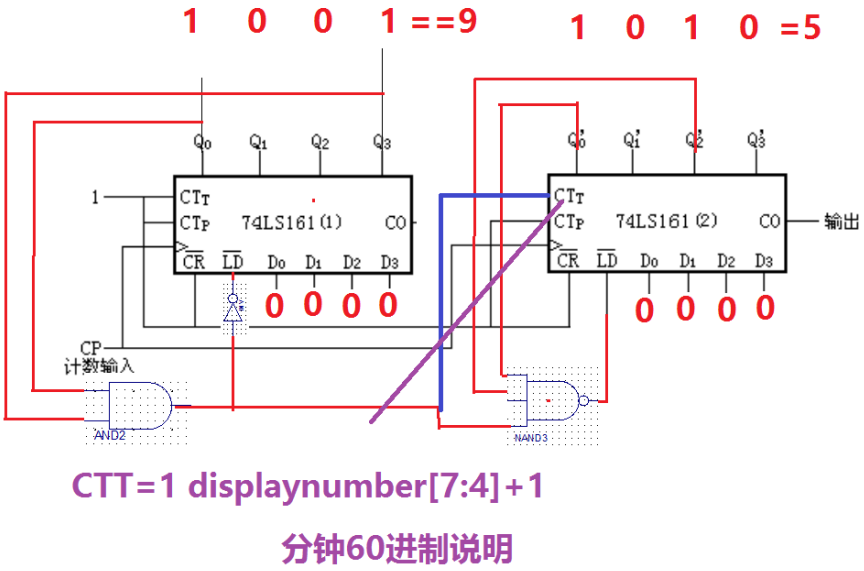

图表 3 74LS161 模块的时序图

2.2.2 数字时钟

1.设计一个数字钟，使用 60 进制和 24 进制计数器，实现 24 小时内时间的实时显示。

2. 数字钟的初值通过初始化语句来实现，用数码管前两位显示小时的十位和个位，后两位显示分钟的十位和个位。

3.用 74LS161 模块显示分钟的 60 进制




图表 3 74LS161 模块用于 60 进制说明

# 三、主要仪器设备

3. SWORD 开发板 1 套
4. 装有 Xilinx ISE 14.7 的计算机 1 台

# 四、操作方法与实验步骤

4.1 设计同步四位二进制计数器 74LS161

4.1.1 新建工程，工程名称用 My74LS161。

4.1.2 用行为描述，其代码如下：

```
module My74LS161(
    input wire CR,
    input wire LD,
    input wire [3:0]num_in,
    input wire [1:0]ctrl,
    input wire clk,
    output reg [3:0]num_o,
    output reg Co
    );

always @(negedge CR)begin
    num_o=4'b0;
end

always @(posedge clk) begin
    if (CR == 1'b1)begin
        if (LD == 1'b0)begin
            num_o = num_in;
        end else begin
            if(ctrl == 2'b11)begin
```

```
                num_o = num_o +1;
            end
        end
    end
    Co = num_o[0]&num_o[1]&num_o[2]&num_o[3]&ctrl[0];
end

endmodule
```

4.1.3 进行波形仿真，其仿真代码如下：

```
    always begin
        clk=0;#20;
        clk=1;#20 ;
    end
    initial begin
        CR = 0;
        LD = 0;
        num_in = 0;
        ctrl = 0;

        #100;
      CR = 1;
        LD = 1;
        num_in = 4'b1100;
        ctrl = 0;
        #30 CR = 0;
        #20 CR = 1;
        #10 LD = 0;
        #30 ctrl = 2'b11;
        #10 LD = 1;

        #510;
        CR = 0;
        #20 CR = 1;
        #500;
    end
```

4.2 设计基于 74LS161 的数字时钟

4.2.1 新建工程，工程名称用 MyClock。

4.2.2 调用 My74LS161、分频模块、显示模块，用 100ms 作为分的驱动时钟，并用结构化描述设计，其代码如下：

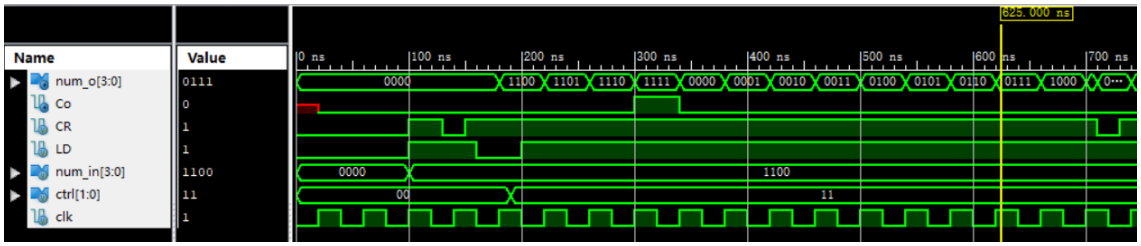
```
module Top(
      input wire clk,
      output wire [3:0]AN,
      output wire [7:0]SEGMENT
      );
       wire [15:0]displaynumber;
clk_100ms m0(clk,clk_100ms);

My74LS161 m1    (.CR(1'b1),
.LD(~(displaynumber[3] & displaynumber[0])),
.ctrl(2'b11), .clk(clk_100ms), .num_in(4'b0),
.num_o(displaynumber[3:0]));

My74LS161 m2(.CR(1'b1), .LD(~(displaynumber[6] & displaynumber[4] & displaynumber[3]
& displaynumber[0])),
.ctrl({1'b1,displaynumber[3]&displaynumber[0]}),
  .clk(clk_100ms), .num_in(4'b0), .num_o(displaynumber[7:4]));

My74LS161 m3(.CR(1'b1), .LD(~(    (displaynumber[11] & displaynumber[8] &
displaynumber[6] & displaynumber[4] & displaynumber[3] & displaynumber[0]) |
(displaynumber[13] & displaynumber[9] & displaynumber[8]& displaynumber[6] &
displaynumber[4] & displaynumber[3] & displaynumber[0]))),
.ctrl({1'b1,(displaynumber[6] & displaynumber[4] & displaynumber[3] & displaynumber[0])}),
  .clk(clk_100ms), .num_in(4'b0), .num_o(displaynumber[11:8]));

My74LS161 m4(.CR(1'b1), .LD(~((displaynumber[13] & displaynumber[9] &
displaynumber[8]& displaynumber[6] & displaynumber[4] & displaynumber[3] &
displaynumber[0])) ),
.ctrl({1'b1,displaynumber[11] & displaynumber[8]& displaynumber[6] & displaynumber[4] &
displaynumber[3] & displaynumber[0]}),
  .clk(clk_100ms), .num_in(4'b0), .num_o(displaynumber[15:12]));

disp_num m5 (clk,
{displaynumber[15:12],displaynumber[11:8],displaynumber[3:0],displaynumber[7:4]}, 4'b0,
4'b0, 1'b0, AN,SEGMENT);
endmodule
```

4.2.3 书写引脚约束并下载验证，引脚约束代码如下：

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

NET "SEGMENT[0]" LOC = AB22     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[1]" LOC = AD24     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[2]" LOC = AD23     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[3]" LOC = Y21      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[4]" LOC = W20      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[5]" LOC = AC24     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[6]" LOC = AC23     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[7]" LOC = AA22     | IOSTANDARD = LVCMOS33;

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

# 五、实验结果与分析

5.1 74LS161 的仿真波形分析



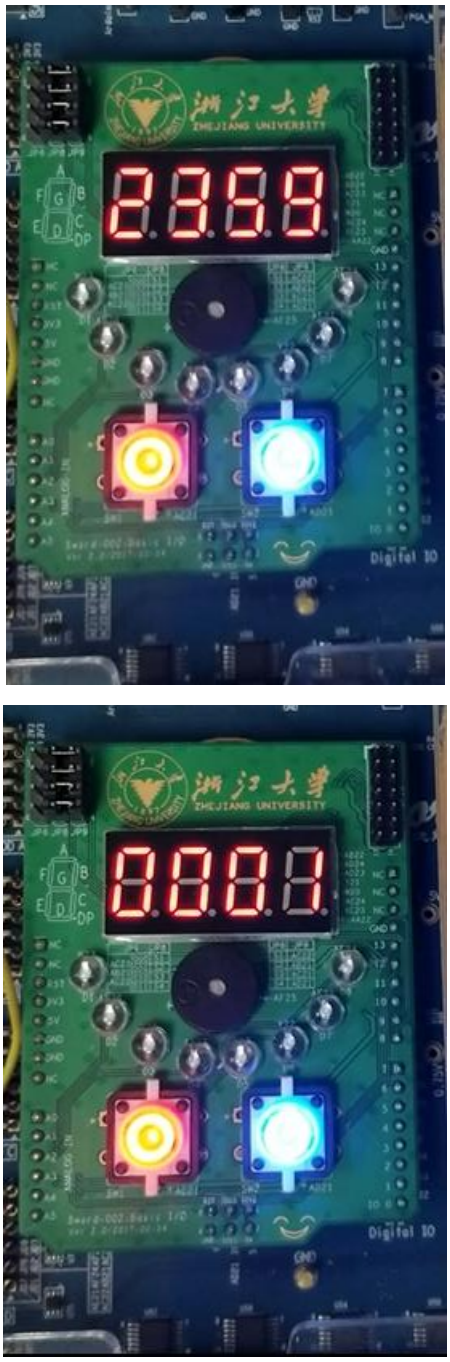

图表 4 74LS161 模块的波形仿真图

可见，当 CR=1，LD=0，并在时钟上升沿的时候，num_in 中的数字赋值给 num_o，此时 num_o=1100；

当 CR 和 LD 都为 1，ctrl 为 11 时，num_o 进行计数，可以看到每一个时钟的上升沿 num_o 都会加 1,并且在 num_o 为 1111 时再加一则为 0000.

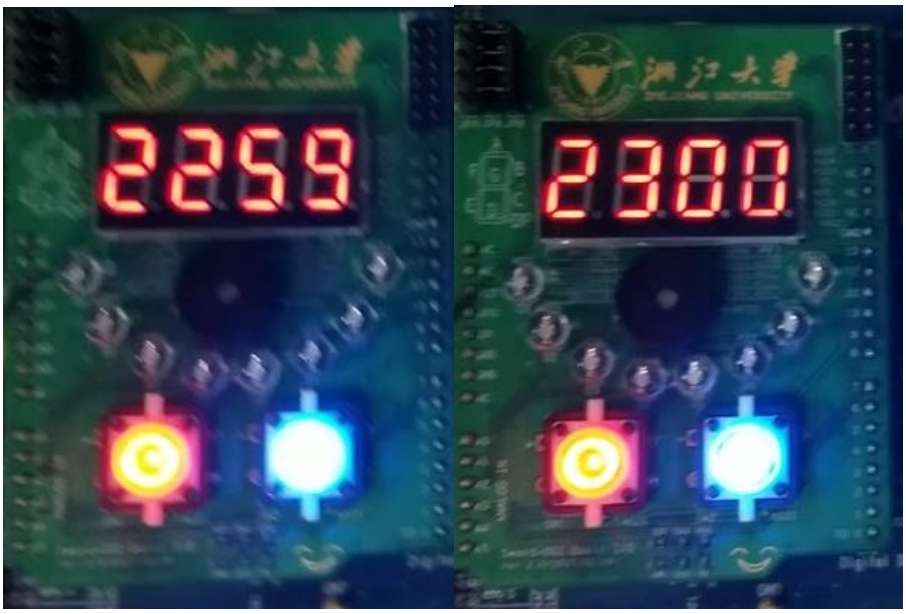
该模块符合所需要的逻辑。

5.2 基于 74LS161 的数字时钟（以下图片截取于一段视频里的不同时间点的图片）




图表 5 计时器由 23.59 进位为 00.00 的刹那

可以看到数字在 23:59 后直接跳回了 00：00，符合设计思想




图表 6 计时器由 22.59 进位为 23.00 的刹那

可以看到数字在 22:59 后直接跳到了 23：00，符合设计思想.

# 六、讨论、心得

本次实验要求实现数字时钟的计时图片，所有的步骤都由 Verilog 代码结构化描述。在本次实验中我遇到最大的困难是对于小时数的个位的进位判定。由于它的进位，即变为 0 时的条件是多种的，我一开始忘记考虑了 2359 跳转 0000 时该位的变化，导致了错误。后来更改回来才正确实现了功能。

对于置 0 的模块使用，本实验有两种方法能够将数字置为 0：一种是用 74LS161 中的 RA 还有一种是该实验现在在用的方法。如果使用第一个方法，由于 RA 的键为 0 则必然会有一定的延迟，所以对于时间要求高的电路，该方法不合适。

本次实验加深我对 Verilog 语言的理解，也让我对时序电路有更好的了解。

# 实验十二--移位寄存器设计与应用实验报告

姓名：黄炯睿　学号：3170103455　专业：信息安全

课程名称：逻辑与计算机设计基础实验　同组学生姓名：无

实验时间：2018-12-20　实验地点：紫金港东 4-509　指导老师：洪奇军

# 五、实验目的和要求

1.1 掌握支持并行输入的移位寄存器的工作原理

1.2 掌握支持并行输入的移位寄存器的设计方法

## 2.1 实验内容

2.1.1 设计 8 位带并行输入的右移移位寄存器

2.1.2 设计跑马灯应用

## 2.2 实验原理

2.2.1 移位寄存器

1.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

2. 数据移动方式：左移、右移、循环移位

3. 数据输入输出方式

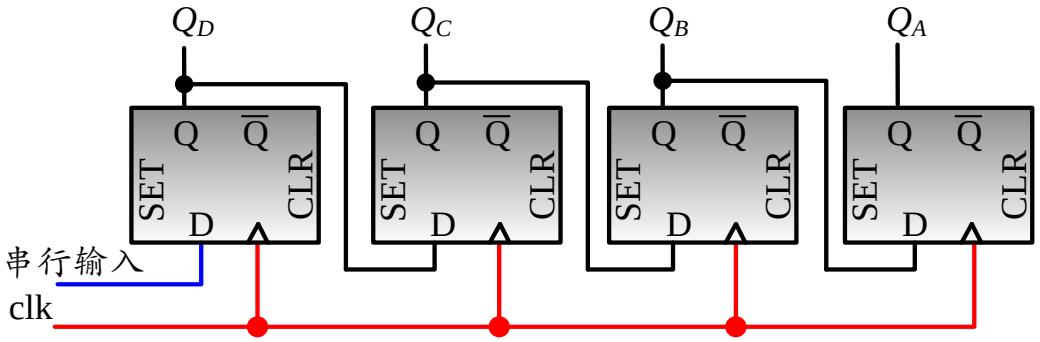
串行输入，串行输出

串行输入，并行输出

并行输入，串行输出

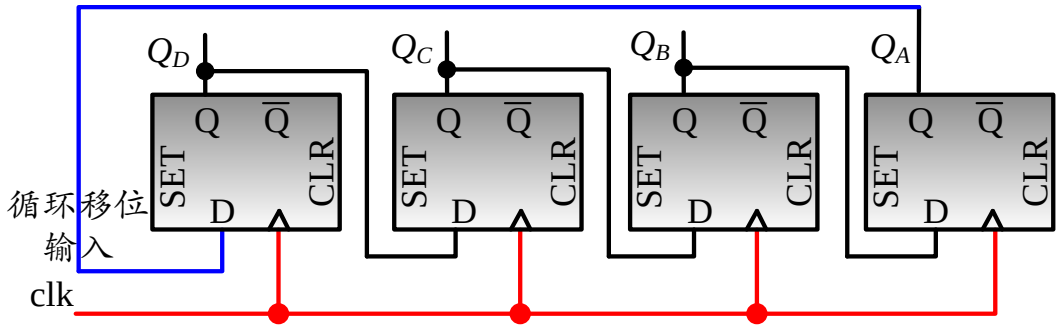
2.2.2 带并行输入的移位寄存器

1. 使用 D 触发器构成串行输入的右移移位寄存器



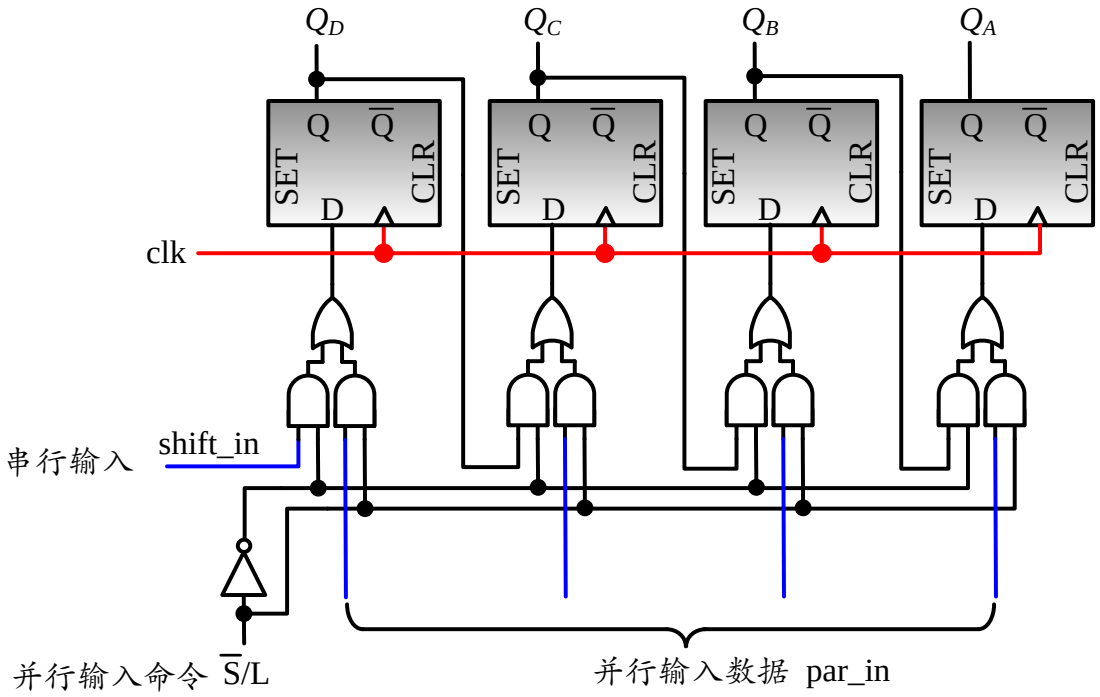

图表 1 串行输入的右移移位寄存器原理图

2. 循环右移移位寄存器




图表 2 循环右移移位寄存器的原理图

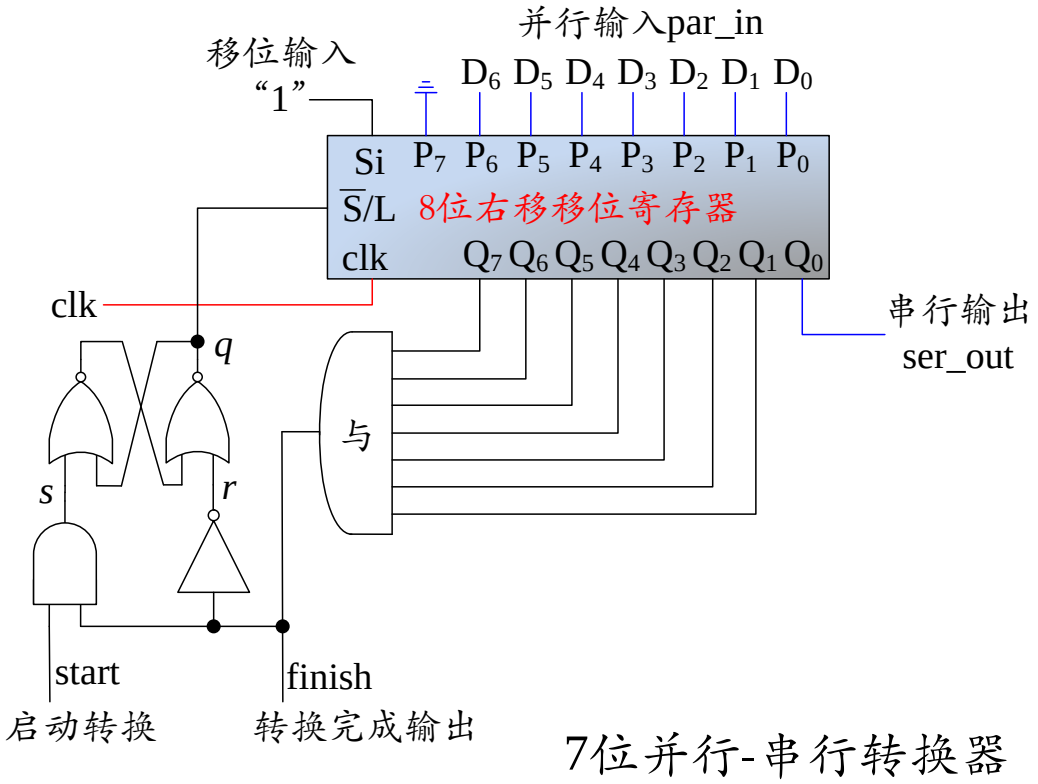
3. 带并行输入的右移移位寄存器




图表 3 带并行输入的右移移位寄存器的原理图

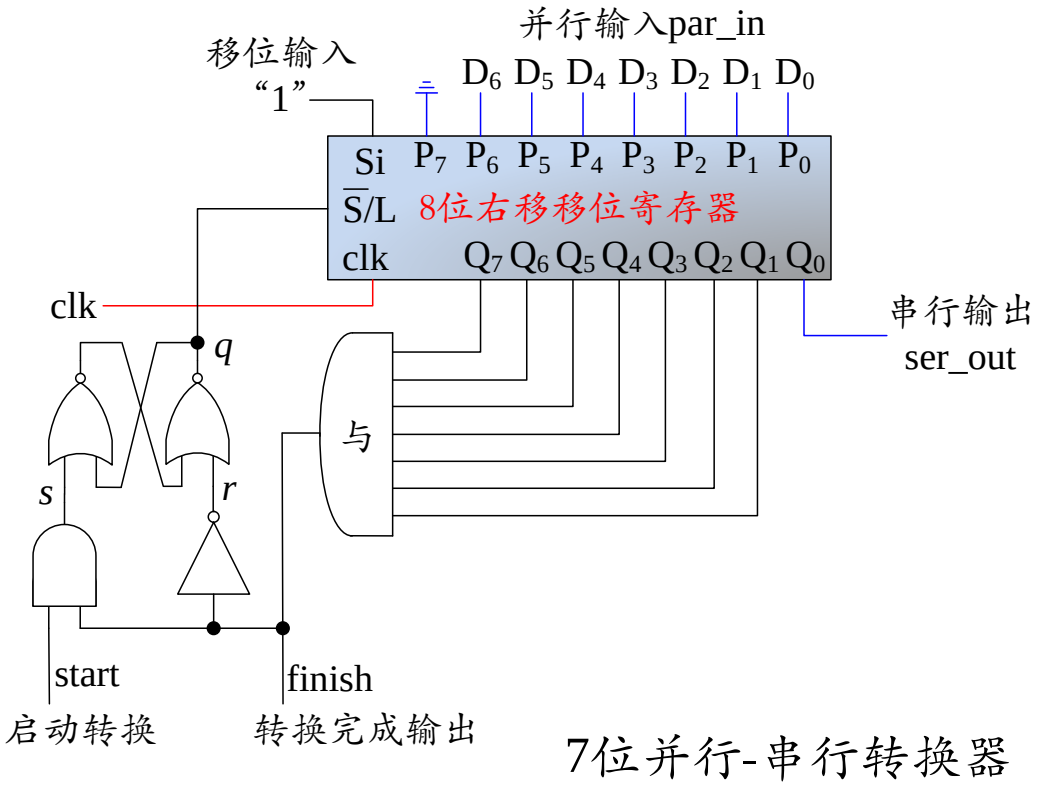
2.2.3 并行—串行转换器

1. **没有启动命令时**




图表 4 没有启动命令时的并行－串行转换器

1. **有启动命令时**




图表 5 有启动命令时的并行－串行转换器

# 三、主要仪器设备

5. SWORD 开发板 1 套
6. 装有 Xilinx ISE 14.7 的计算机 1 台

# 四、操作方法与实验步骤

4.1 设计 8 位带并行输入的右移移位寄存器

4.1.1 新建工程，工程名称用 ShiftReg8b。

4.1.2 用结构化描述设计

```
module ShfitReg8b(
    input wire clk,S_L,s_in,
    input wire [7:0] p_in,
    output reg [7:0] Q
    );
initial begin
    Q = 0;
end
always@(posedge clk)
begin
    if(S_L == 1)begin
        Q <= p_in;
    end else if(S_L ==0)
    begin
        Q[7] <= s_in;
        Q[6] <= Q[7];
        Q[5] <= Q[6];
        Q[4] <= Q[5];
        Q[3] <= Q[4];
        Q[2] <= Q[3];
        Q[1] <= Q[2];
        Q[0] <= Q[1];
    end
end

endmodule
```

4.1.3 波形仿真，仿真代码如下：

```
initial begin
        // Initialize Inputs
        clk = 0;
        S_L = 0;
        s  in = 0;
        p_in = 0;
        #100;
        S_L = 0;
        s  in = 1;
        p  in =0;
        #200;
        S_L = 1;
        s_in = 0;
        p  in = 8'b0101  0101;
        #500;
    end

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

4.2 设计跑马灯应用

4.1.1 新建工程，工程名称用 **MyMarquee。**

4.1.2 用结构化描述设计

调用 ShfitReg8b

调用分频模块，用 1s 作为移位寄存器驱动时钟

调用显示模块

调用 CreateNumber 模块

```
module Top(
    input wire clk,
    input wire [4:0]SW,
    output wire [3:0]AN,
    output wire [7:0]SEGMENT,
    output wire [7:0]LED
    );
      wire [15:0]num;
      wire clk_1s;
      wire s_in;

    clk_1s m1(clk,clk_1s);
    CreateNumber m2(.btn({2'b0,SW[1:0]}),.sw(4'b1100),.num(num));
    assign s_in = (SW[4]==1)? LED[0]:SW[3];
    ShfitReg8b m3(clk_1s, SW[2], s_in, num[7:0], LED);
    disp_num m4(clk, {num[3:0],num[7:4],LED}, 4'b0,4'b0,1'b0,AN,SEGMENT);

endmodule
```

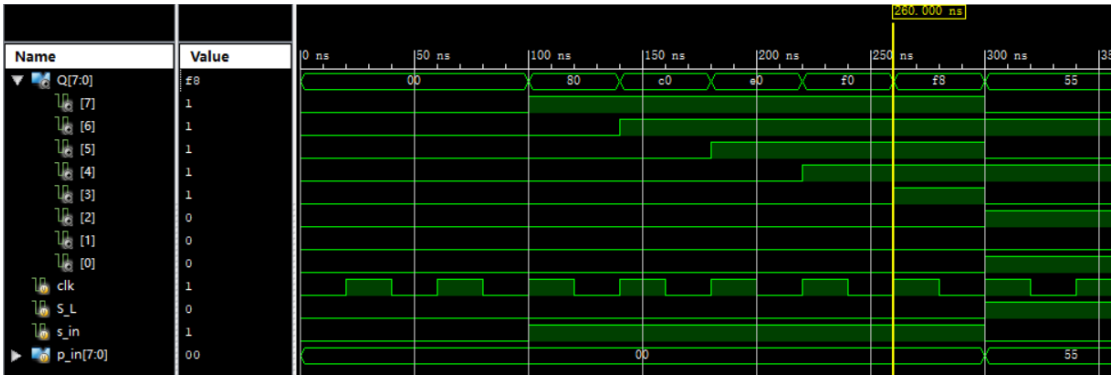
4.1.3 添加引脚约束文件，代码如下：

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

NET "SW[0]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "SW[1]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "SW[2]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "SW[3]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "SW[4]" CLOCK_DEDICATED_ROUTE = FALSE;

NET "SEGMENT[0]" LOC = AB22     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[1]" LOC = AD24     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[2]" LOC = AD23     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[3]" LOC = Y21      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[4]" LOC = W20      | IOSTANDARD = LVCMOS33;
NET "SEGMENT[5]" LOC = AC24     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[6]" LOC = AC23     | IOSTANDARD = LVCMOS33;
NET "SEGMENT[7]" LOC = AA22     | IOSTANDARD = LVCMOS33;

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

NET "LED[0]" LOC=W23 | IOSTANDARD = LVCMOS33;
NET "LED[1]" LOC=AB26 | IOSTANDARD = LVCMOS33;
NET "LED[2]" LOC=Y25 | IOSTANDARD = LVCMOS33;
NET "LED[3]" LOC=AA23 | IOSTANDARD = LVCMOS33;
NET "LED[4]" LOC=Y23 | IOSTANDARD = LVCMOS33;
NET "LED[5]" LOC=Y22 | IOSTANDARD = LVCMOS33;
NET "LED[6]" LOC=AE21 | IOSTANDARD = LVCMOS33;
NET "LED[7]" LOC=AF24 | IOSTANDARD = LVCMOS33;
```

# 五、实验结果与分析

### 5.1. 设计 8 位带并行输入的右移移位寄存器

对该模块进行仿真，得到的仿真图片如下：




图表 6　8 位带并行输入的右移移位寄存器的仿真图

当 S_L=0 时，寄存器进行串行右移，在 0-100ns 时由于 s_in 为 0，所以 Q 始终是 16 进制的 00；当 100-300ns 时，s_in 为 1，则每经过一个时钟上升沿，Q 右移一位，在 100ns 时就由 0000 0000 变化为 1000 0000（80）。以此类推，1000 0000 右移为 1100 0000（C0）…

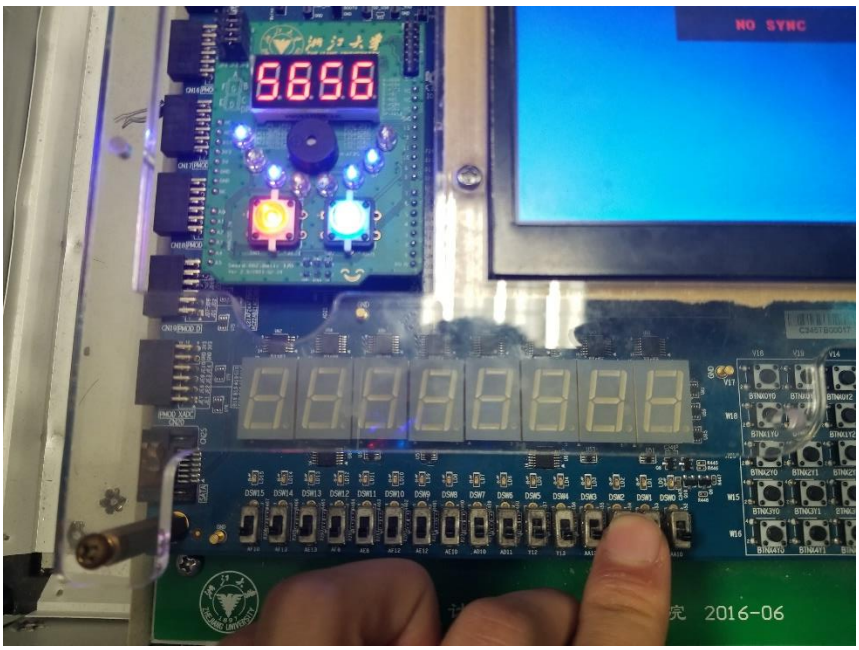
当 S_L=1 时， 寄存器进行并行赋值，将 p_in 值赋值给 Q，即在 300ns 时，Q 获得 p_in 的值，变为 55.

### 5.2. 设计跑马灯应用

5.2.1. 按键自增控制输入

通过 SW[0] 和 SW[1]，可以将左边的两个寄存器以不断递增的方法赋值为任意的值，图中我将它赋值为 5,6.

5.2.2. 并行输入，将{RegA,RegB}赋给移位寄存器




将 SW[2]拨为 1， 则将左边控制的寄存器的值赋值给右边的移位寄存器的值，并且可以看到 LED 灯的亮灭发生变化。

5.2.3 循环右移移位


| 当前显示值 | 下一个显示值 | 当前显示对应的 16 进制数 |
|---|---|---|
| 0101 0110 | 0010 1011 | 56 |
| 0010 1011 | 1001 0101 | 2B |
| 1001 0101 | 1100 1010 | 95 |
| 1100 1010 | 0110 0101 | CA |
| 0110 0101 | 1011 0010 | 65 |
| 1011 0010 | 0101 1001 | B2 |
| 0101 1001 | 1010 1100 | 59 |
| 1010 1100 | 0101 0110 | AC |
| 0101 0110 | | 56 |

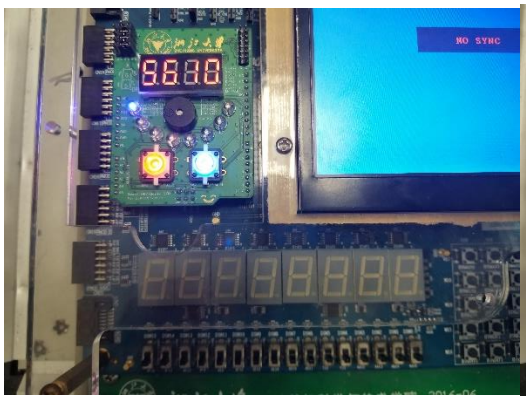
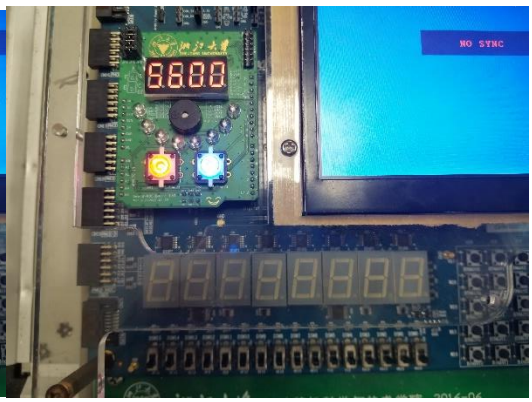
由此可见，上图的两个显示数字均符合要求。

5.2.4. 串行右移 s_in 为 1（**最右边的数字是移位寄存器的高位**）




如图，在移位寄存器为 FF 时进行 s_in 为 1 的的实验，可以看到 0F 的值逐渐变大直至为 FF. 数字的变化或许不够敏感，可以注意观察 LED 灯，LED 灯从右往左依次亮起，意味着从右往左 1 的加入。

5.2.4. 串行右移 s_in 为 0（**最右边的数字是移位寄存器的高位**）

0000 0001 向右移一位变为 0000 0000. 即为 01 变为 00. 观察到 LED 等从右往左依次熄灭，代表 0 在从高位向低位的加入。

# 六、讨论、心得

做到了最后一个实验总算有了这个实验还蛮简单的感觉，在实验过程中并没有什么大的问题。就是在 Top 模块写 if-else 语句时发现该语句不能在 Always 模块外写，但可以用三目运算符？：来代替，也算是进一步认识了 Verilog 语法吧

在最后的显示时，由于我的疏忽，我把一位寄存器的高低位显示反了，也就是现在看到的移位寄存器的高位在右，低位在左。

感谢这一个学期的学习，让我从无到有，认识了 Verilog 语言。