实验十二报告
（寄存器和寄存器传输设计）

姓名：蒋仕彪　学号：3170102587　专业：求是科学班（计算机）1701

课程名称：逻辑与计算机设计基础实验

实验时间：2018-12-06

一、实验目的

- 掌握寄存器传输电路的工作原理
- 掌握寄存器传输电路的设计方法
- 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：

- 任务：基于 ALU 的数据传输应用设计

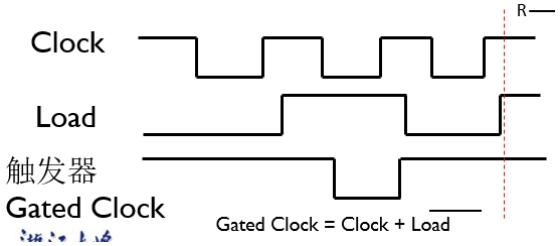
2.2 实验原理：

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

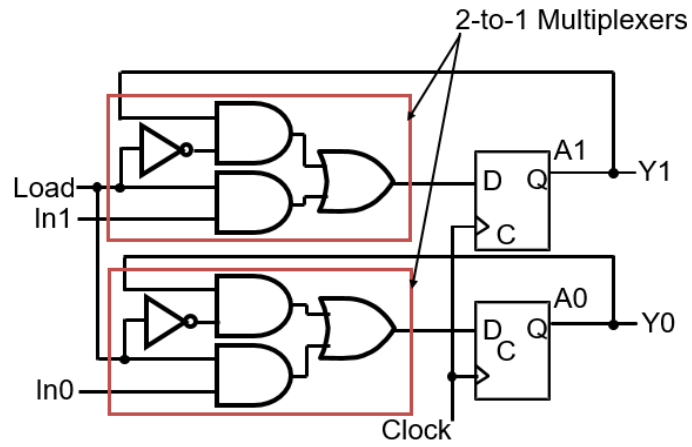
2.2.2 采用门控时钟的寄存器

- 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：



2.2.3 采用Load控制反馈的寄存器

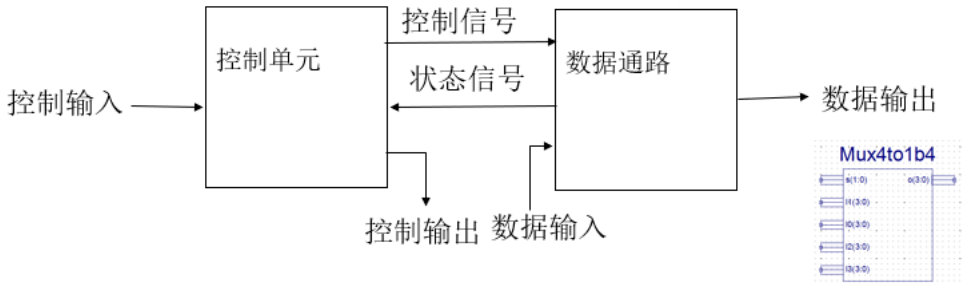
- 进行有选择地加载寄存器的更可靠方法是：
  - 保证时钟的连续性
  - 选择性地使用加载控制来改变寄存器的内容



2.2.4 寄存器传输

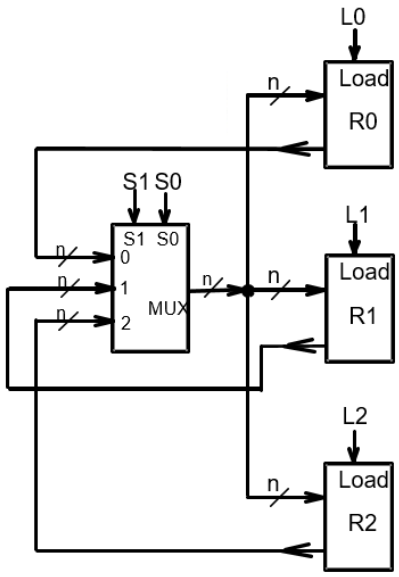
- 寄存器传输：寄存器中数据的传输和处理
- 三个基本单元：寄存器组、操作、操作控制

基本操作:加载、计数、移位、加法、按位操作等



2.2.5 基于多路选择器总线的寄存器传输

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



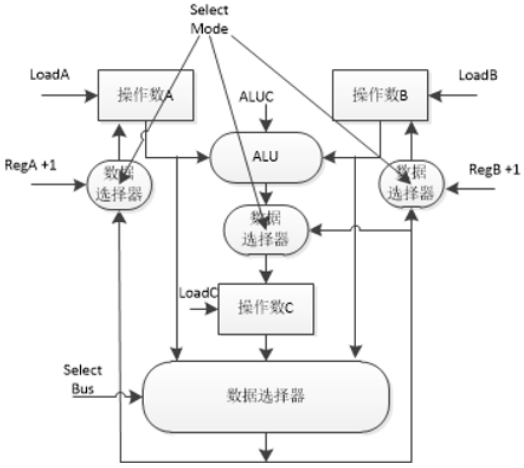
2.2.6 寄存器传输应用设计

Mode1

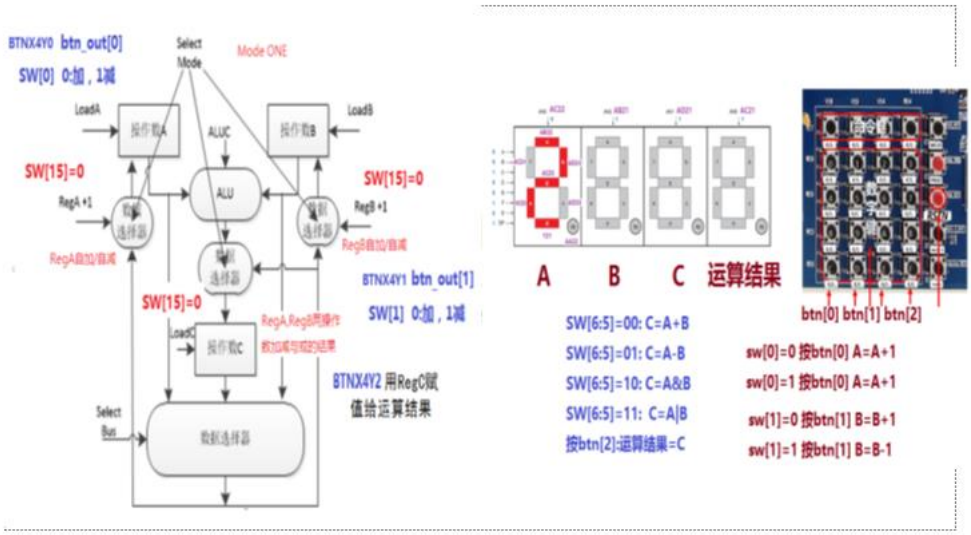
ALU运算输出控制

Mode2

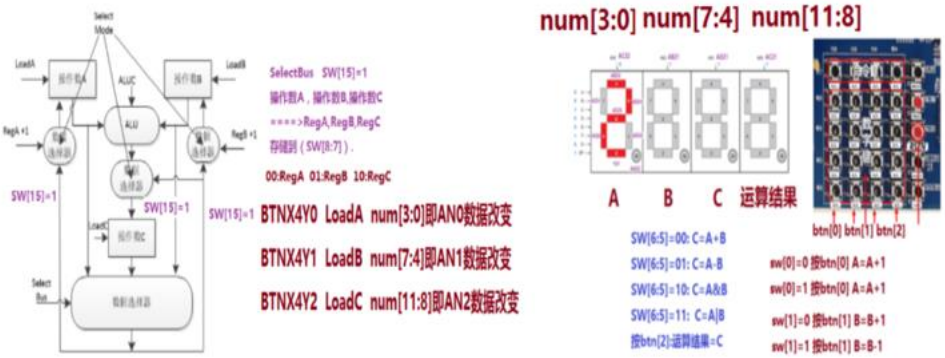
数据传输控制



2.2.7 Mode1　ALU运算输出控制



2.2.8 数据传输控制



sw[8:7]对应SelectBus即总线上放的是什么数据：
00-总线上的数据选择A，按BTX4Y0存储到num[3:0]
01-总线数据上的选择B，按BTX4Y1存储到num[7:4]
10-总线数据上的选择C。按BTX4Y2存储到num[11:8]

三、主要仪器设备

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

四、操作方法与实验步骤

4.1 基于ALU的数据传输应用设计（1）

- 新建工程
  - 工程名称用MyALUTrans。
  - Top Level Source Type用HDL
- 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

4.2 基于ALU的数据传输应用设计（2）

- 新建源文件
  - 类型是Verilog
  - 文件名称用Top。
  - 右键设为“Set as Top Module”
- 实现基于ALU的数据传输应用设计

4.3 物理验证

- **UCF引脚定义**
  - **输入**
    - sw[15]=0 Mode0
      - 按键控制输入：sw[2]控制Reg A，sw[3]控制Reg B，sw[4]对RegC赋值
      - 按键加/减1控制：sw[0]对应btn[0]，sw[1]对应btn[1]
      - ALU运算控制：sw[6:5],00-加，01-减，10-与，11-或
      - **(RegC是Reg A，Reg B加减与或的结果)**
    - sw[15]=1 Mode1　　数据传输控制
      - sw[8:7]对应SelectBus：00-选择A，01-选择B，10-选择C
      - sw[2] LoadA(num[3:0]),sw[3] LoadB(num[7:4])，sw[4] LoadC(ALU结果)
      - (对SW[8:7]选择出来后的结果加载到哪个寄存器中)
  - **输出**
    - AN[0]：Reg B
    - AN[1]：Reg A
    - AN[2]：ALU结果
    - AN[3]：Reg C

浙江大学

sw[15]=0 Mode0

按键控制输入：

BTNX4Y0 作为按键信号控制RegA(自加或自减)。

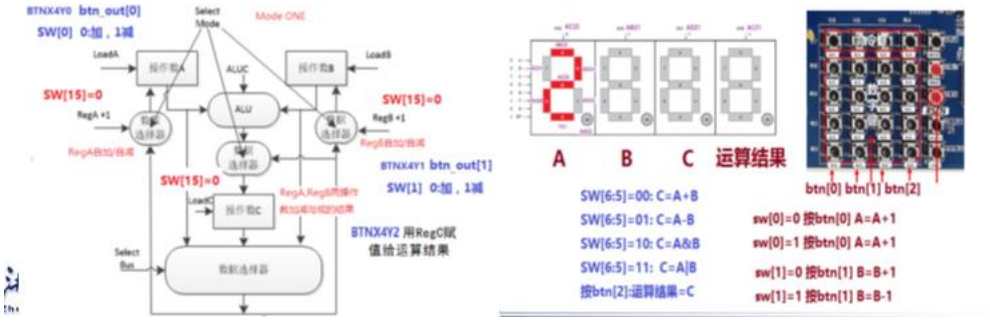
BTNX4Y1 作为按键信号控制RegB。BTNX4Y2 作为按键信号对RegC赋值。

按键加/减1控制：sw[0]=0加 /1减，对应btn_out[0]即去抖动后的BTNX4Y0按键。

sw[1]=0加/1减，对应btn_out[1] 即去抖动后的BTNX4Y1按键。

btn_out[2]即去抖动后的BTNX4Y2按键。

- ALU运算控制：sw[6:5],00-加，01-减，10-与，11-或
- (RegC是Reg A, Reg B加减与或的结果)



- sw[15]=1 Mode1 数据传输控制

sw[8:7]对应SelectBus：00-总线数据选择A，01-总线数据选择B，10-选择C

BTNX4Y0 LoadA(num[3:0])[总线数据存储到num[3:0]，位置在AN0]。

BTNX4Y1 LoadB(num[7:4])[总线数据存储到num[7:4]，位置在AN1]

BTNX4Y2 LoadC(ALU运算结果) [总线数据存储到num[11:8]，位置在AN3]

- (对SW[8:7]选择出来后的结果加载到哪个寄存器中)

- **输出 {num[7:0],C,num[11:8]}**
  - AN[0]：RegA,AN[1]：RegB,AN[2]:ALU结果, AN[3]:Reg C


```
增加输入：input wire BTN_Y[3:0],output wire BTN_X
 assign BTN_X=0;
AddSub4b m4(.A(num[3:0]),.B(4'b0001),.Ctrl(SW[0]),.S(A1));
AddSub4b m5(.A(num[7:4]),.B(4'b0001),.Ctrl(SW[1]),.S(B1));
Mux4to1b4 m6(.I0(num[3:0]),.I1(num[7:4]),.I2(num[11:8]),
             .I3(4'b0),.s(SW[8:7]),.o(Result));
assign A2 = (SW[15]==1'b0)?A1:Result;
------   B2 = (SW[15]==1'b0) ....
         C2 = (SW[15]==1'b0).....
always@(posedge btn_out[0]) num[3:0] = A2;
------                      num[7:4] =?
                             num[11:8]=?
 myALUm7(.A(num[3:0]),.B(num[7:4]),.S(SW[6:5]),.C(C),.Co(Co[0]));
DispNum m8(clk, {num[3:0],num[7:4],C,num[11:8]}, 4'b0, 4'b0, 1'b0,
AN,SEGMENT);
```

五、实验结果与分析

5.1 基于ALU的数据传输应用设计

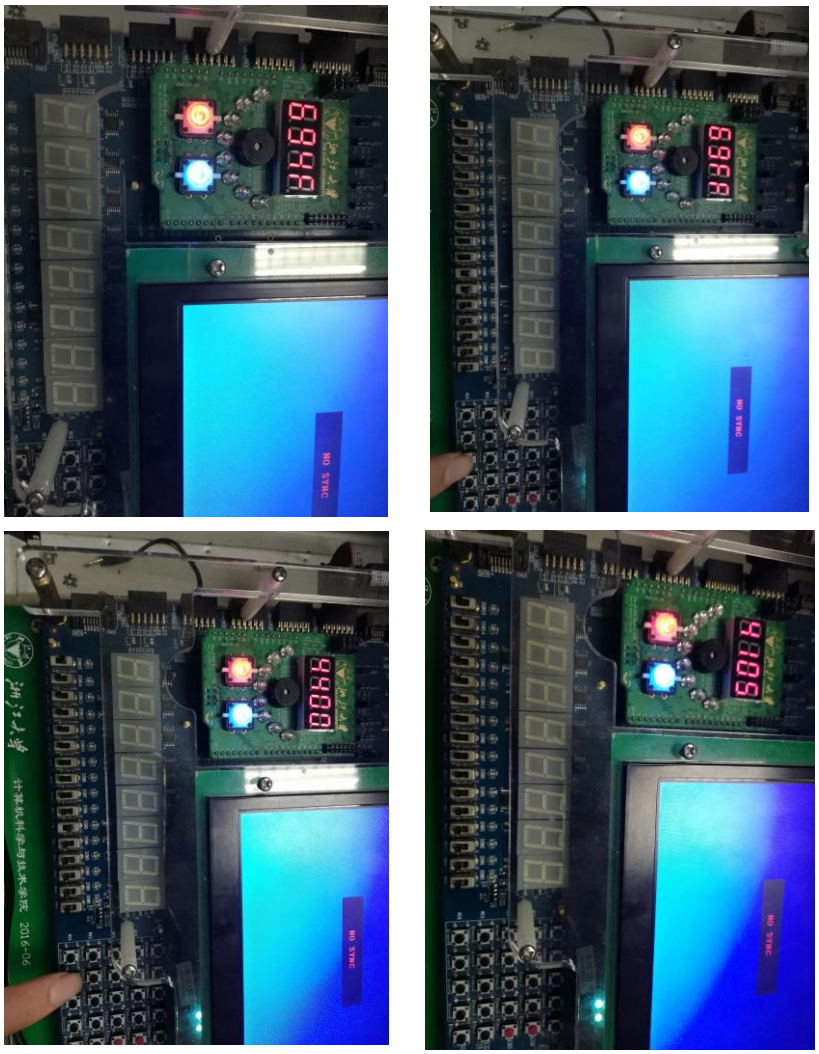
```
module Top(
    input wire clk,
    input wire [15:0]SW,
    input wire BTN_Y[3:0],
    output wire [3:0]AN,
    output wire [7:0]SEGMENT,
    output wire BTN_X
    //output wire BTN_X
);
    reg [15:0] num;
    wire [3:0] btn_out;
    wire Co;
    wire [31:0] clk_div;
    wire [3:0] A1,B1,C1,A2,B2,C2;
    wire [3:0] Result;
    pbdebounce m0(clk_div[17],BTN_Y[0],btn_out[0]);
    pbdebounce m1(clk_div[17],BTN_Y[1],btn_out[1]);
    pbdebounce m2(clk_div[17],BTN_Y[2],btn_out[2]);
    clkdiv m3 (clk, 0, clk_div);
    assign BTN_X=0;
    AddSuber4b m4(.A(num[3:0]),.B(4'b0001),.Ctrl(SW[0]),.S(A1));
    AddSuber4b m5(.A(num[7:4]),.B(4'b0001),.Ctrl(SW[1]),.S(B1));
    Mux4to1b4
m6(.I0(num[3:0]),.I1(num[7:4]),.I2(num[11:8]),.I3(4'b0),.s(SW[8:7]),.o(
Result));
    assign A2 = (SW[15]==1'b0)?A1:Result;
    assign B2 = (SW[15]==1'b0)?B1:Result;
    assign C2 = (SW[15]==1'b0)?C1:Result;

    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;

    ALU4b m7(.A(num[3:0]),.B(num[7:4]),.S(SW[6:5]),.C(C1),.Co(Co));
    disnum m8(clk, {num[3:0],num[7:4],C1,num[11:8]}, 4'b0, 4'b0, 1'b0,
AN,SEGMENT);
endmodule
```

◘ 以上为top.v代码

5.2 物理验证

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "AN[1]" LOC = AD21 | IOSTANDARD = LVCMOS33;
NET "AN[0]" LOC = AC21 | IOSTANDARD = LVCMOS33;
NET "AN[2]" LOC = AB21 | IOSTANDARD = LVCMOS33;
NET "AN[3]" LOC = AC22 | IOSTANDARD = LVCMOS33;
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_Y[3]" LOC = W14 | IOSTANDARD = LVCMOS18;
NET "BTN_Y[0]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "BTN_Y[1]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "BTN_Y[2]" CLOCK_DEDICATED_ROUTE = FALSE;
NET "BTN_Y[3]" CLOCK_DEDICATED_ROUTE = FALSE;
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 "SW[0]" LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "SW[1]" LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "SW[2]" LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "SW[3]" LOC = AA12 | IOSTANDARD = LVCMOS15;
NET "SW[4]" LOC =  Y13 | IOSTANDARD = LVCMOS15;
NET "SW[5]" LOC =  Y12 | IOSTANDARD = LVCMOS15;
NET "SW[6]" LOC = AD11 | IOSTANDARD = LVCMOS15;
NET "SW[7]" LOC = AD10 | IOSTANDARD = LVCMOS15;
NET "SW[8]" LOC = AE10 | IOSTANDARD = LVCMOS15;
NET "SW[9]" LOC = AE12 | IOSTANDARD = LVCMOS15;
NET "SW[10]" LOC = AF12 | IOSTANDARD = LVCMOS15;
NET "SW[11]" LOC =  AE8 | IOSTANDARD = LVCMOS15;
NET "SW[12]" LOC =  AF8 | IOSTANDARD = LVCMOS15;
NET "SW[13]" LOC = AE13 | IOSTANDARD = LVCMOS15;
NET "SW[14]" LOC = AF13 | IOSTANDARD = LVCMOS15;
NET "SW[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
```

- 以上为引脚定义



◘ 以上为实际成果展示

六、实验心得

这次实验要新写的东西并不多（只需在top层上完成一个简单的存储器），但是对top.v的逻辑性要求很强。

一开始我没有完全理解“用户需求”，对SW按钮具体要操控什么束手无策。后来研究好了好久才弄明白：SW[15]是模块总体选择器，SW[0]表示执行ALU运算（此时SW[5]和SW[6]决定运算类型），SW[1]表示执行数据传输（此时SW[8]和SW[7]决定把数据放到哪个对应的总线上，三个按钮表示把数据总线的数据赋给A,B,C的哪个）。SW[0]和SW[1]是控制两个数字A和B是自增还是自减（两个按钮是具体累加/减）。

只要厘清了逻辑，一步一步按逻辑写，还是挺好实现的。

这次实验也“告诫”我，以前封装好的代码（如disnum，sy7,防抖动pbdebounce，四选一Mux4to1b4）要好好保存。它其实是之前很多实验的综合，光是找模块调用花了我好久，还要一个一个文件翻过去。

实验十三（计数器、定时器设计与应用）

实验报告

姓名：蒋仕彪　学号：3170102587　专业：求是科学班（计算机）1701

课程名称：逻辑与计算机设计基础实验

实验时间：2018-12-13

一、实验目的

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

二、实验内容和原理

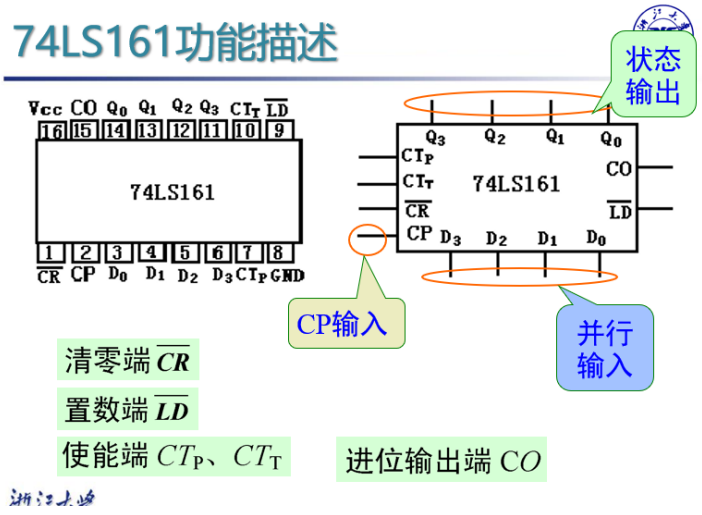
2.1 实验内容

- 任务1：采用行为描述设计同步四位二进制计数器74LS161
- 任务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

- 74LS161是常用的四位二进制可预置的同步加法计数器
- 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



2.2.2 74LS161功能表

异步清0功能最优先

输入						输出			
$\overline{CR}$	$\overline{LD}$	CT_P	CT_T	CP	$D_3D_2D_1D_0$	Q_3	Q_2	Q_1	Q_0
0	×	×	×	×	××××	0	0	0	0
1	0	×	×	↑	$d_3d_2d_1d_0$	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			

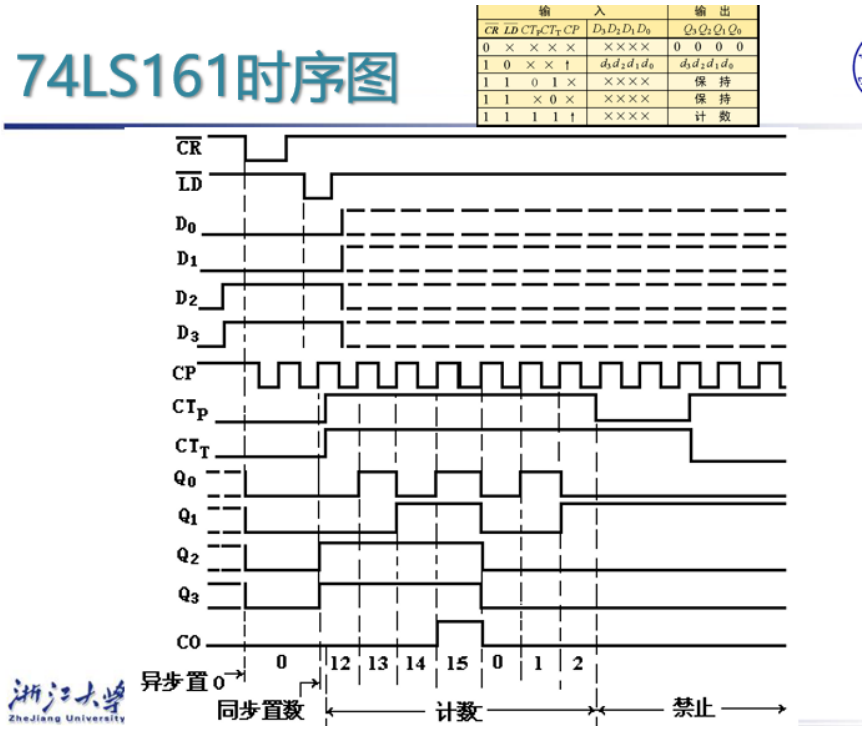
同步并行置数

CP上升沿有效

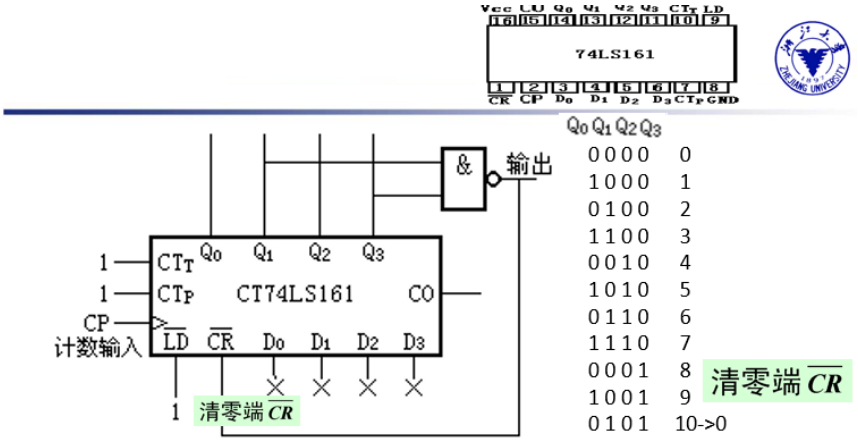
$CO = Q_3Q_2Q_1Q_0CT_T$

浙江大学 ZheJiang University

2.2.3 74LS161 时序图



2.2.4 实现十进制计数器

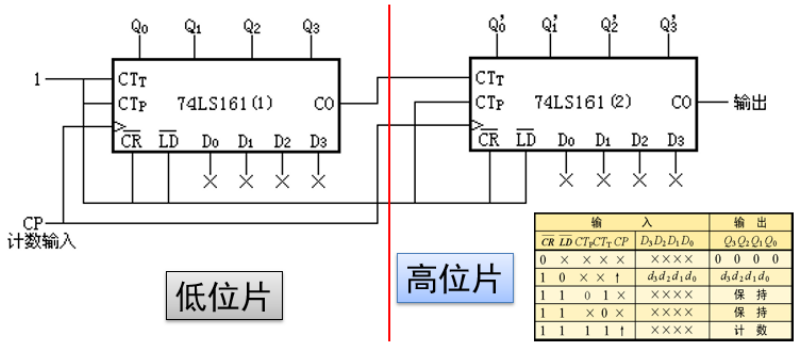


利用与非门拾取状态1010

实现十进制计数 （0000到1001）

改变与非门的输入信号，可以实现其它进制计数。

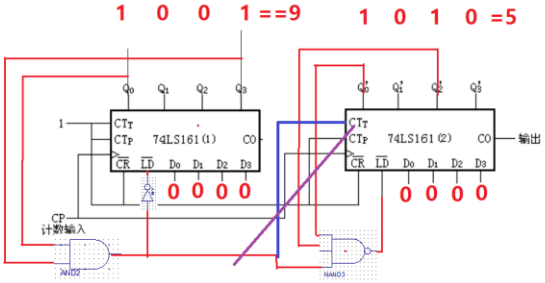
2.2.5 实现 16×16 进制计数器



在计到1111以前，CO_1＝0，高位片保持原状态不变

在计到1111时，CO_1＝1，高位片在下一个CP加一

2.2.6 分钟 60 进制（十进制显示）



CTT=1 displaynumber[7:4]+1

分钟60进制说明

三、实验设备与材料

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

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

- 新建工程
  - 工程名称用My74LS161。
  - Top Level Source Type用HDL
- 用行为描述设计
  - CR是异步清零
  - LD是同步置位

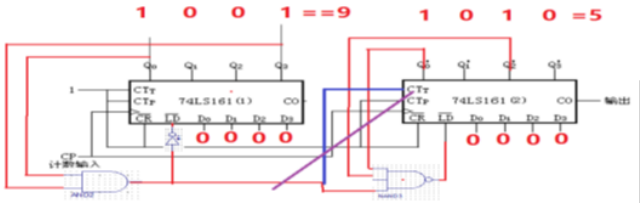
输 入						输 出
$\overline{CR}$	$\overline{LD}$	CT_P	CT_T	CP	$D_3D_2D_1D_0$	$Q_3Q_2Q_1Q_0$
0	×	×	×	×	××××	0 0 0 0
1	0	×	×	↑	$d_3d_2d_1d_0$	$d_3d_2d_1d_0$
1	1	0	1	×	××××	保 持
1	1	×	0	×	××××	保 持
1	1	1	1	↑	××××	计 数

4.2 基于74LS161设计时钟应用

- 新建工程
  - 工程名称用MyClock。
  - Top Level Source Type用HDL
- 用结构化描述设计
  - 调用My74LS161
  - 调用分频模块，用100ms作为分的驱动时钟
  - 调用显示模块
  - （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。）
- 下载验证

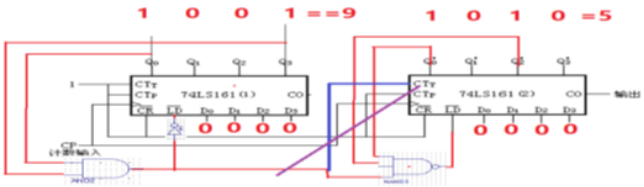
4.3 实现十进制计数

```
my74LS161 m0  (.CR(1'b1),
.Ld(~(displaynumber[3] & displaynumber[0])),
.CTT(1'b1), .CTP(1'b1), .CP(clk_100ms), .D(4'b0), .Q
(displaynumber[3:0]));
//1001==9时 Q=D ==》  Q=0   ==>
displaynumber[3:0]=0
```



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			

```
my74LS161 m1(.CR(1'b1), .Ld(~(displaynumber[6] &
displaynumber[4] & displaynumber[3] & displaynumber[0])),
.CTT(displaynumber[3]&displaynumber[0]),
.CTP(1'b1), .CP(clk_100ms), .D(4'b0), .Q(displaynumber[7:4]));
//0101 1001==》59   Q=D ==》  Q=0,displaynumber[7:4]=0
//CTT:1001==9时   CTT=1(CTP=1)==> 计数  displaynumber[3:0]=9
时 displaynumber[7:4]+1
```



浙江大学 ZheJiang University

五、实验结果与分析

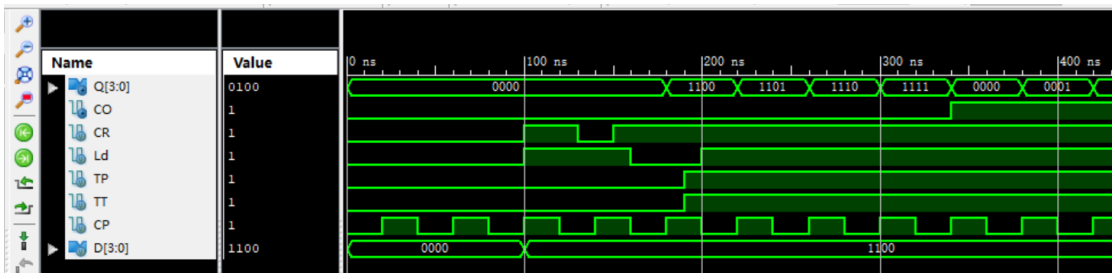
5.1 设计同步四位二进制计数器74LS161

```
odule work(
   input CR,CP,Ld,TP,TT,
    input wire [3:0]D,
    output reg [3:0]Q,
    output reg CO
    );
    initial begin
        Q = 4'b0000;
        CO = 0;
    end

    always @(posedge CP or negedge CR) begin
       if (~CR) begin
           Q[3:0] <= 4'b0000;
        end
        else if (~Ld) begin
            Q[3:0] <= D[3:0];
        end
        else if (CR && Ld && TP && TT) begin
            if (Q == 4'b1111) begin
              CO <= 1;
              Q <= 4'b0000;
            end
            else begin
                Q <= Q + 4'b0001;
           end
        end
    end

endmodule
```

◘ 以上为74LS161的设计代码

```
initial begin
        CR = 0;
        Ld = 0;
        TP = 0;
        TT = 0;
        D = 0;

        #100;
        CR = 1;
        Ld = 1;
        D = 4'b1100;
        TT = 0;
        TP = 0;
        #30 CR=0;
        #20 CR=1;
        #10 Ld=0;
        #30 TT=1;
        TP=1;
        #10 Ld=1;

        #510;
        CR=0;
        #20 CR=1;
        #500;
    end
    initial forever begin
       CP = 0; #20;
        CP = 1; #20;
    end
```

◘ 以上为仿真代码



◘ 以上为仿真波形（容易发现，reg Q是正常被修改的，符合我们的需求）

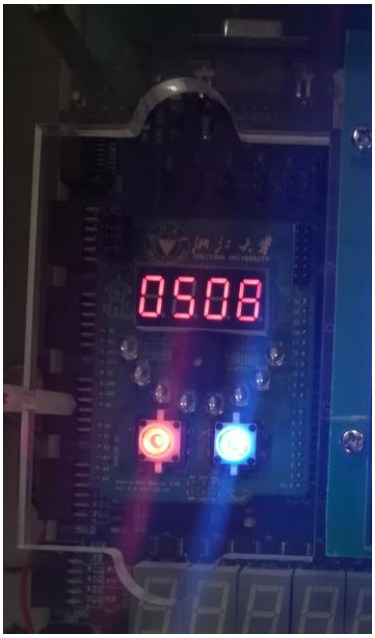
5.2 基于74LS161设计时钟应用

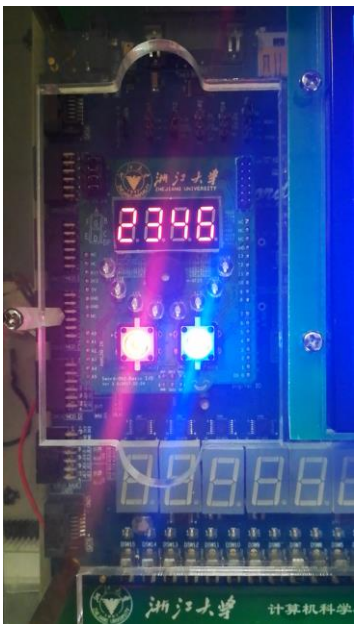
```
module Top(
    input wire clk,
     output wire [7:0] SEGMENT,
     output wire [3:0] AN
    );
     //wire [15:0]num;
     wire [15:0] num;
     wire [31:0] Time;
     clkdiv m0(clk,0,Time);
     work m1(.CR(1'b1),.CP(Time[20]),.Ld(~(num[3] &&
num[0])),.TP(1'b1),.TT(1'b1),.D(4'b0000),.Q(num[3:0]));
     work m2(.CR(1'b1),.CP(Time[20]),.Ld(~(num[6] && num[4] && num[3] &&
num[0])),.TP(1'b1),.TT(num[3] && num[0]),.D(4'b0000),.Q(num[7:4]));
     work m3(.CR(1'b1),.CP(Time[20]),.Ld(~((num[13] && num[9] && num[8] ||
num[11] && num[8]) && num[6] && num[4] && num[3] &&
num[0])),.TP(1'b1),.TT(num[6] && num[4] && num[3] &&
num[0]),.D(4'b0000),.Q(num[11:8]));
     work m4(.CR(1'b1),.CP(Time[20]),.Ld(~(num[13] && num[9] && num[8] &&
num[6] && num[4] && num[3] && num[0])),.TP(1'b1),.TT(num[11] && num[8] &&
num[8] && num[6] && num[4] && num[3] && num[0]),.D(4'b0000),.Q(num[15:12]));
     disnum m5(clk,num,4'b0000,4'b0000,1'b0000,AN,SEGMENT);
endmodule
```

◘ 以上为top.v代码

```
net "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "SEGMENT[0]"  LOC = AB22 | IOSTANDARD = LVCMOS33;#a
NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33;#b
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;#g
NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33;#point

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

◘ 以上为引脚定义





◘ 以上为成果展示

六、实验心得

设计模块的部分变得轻松了：掌握了verilog语言后，几行就搞定了，不像以前的画图连接，要画死人。

本来以为这次的实验特别简单：以前这么复杂的实验都做过了，现在只要做一个CLOCK就行？但是CLOCK上四位数字合适从头循环着实让我思考了很久。

我尝试了很多很多可能的写法，最后才成功。首先，每一位+1的条件都要附加上：后面几位达到了临界条件；把每一位清零的命令要求就更严格了。

第一位的判断最坑了，如果第一位是0/1，第二位是9（且后面达到了临界要求），就要+1；但是如果第一位是2，只要第二位是3（且后面达到临界要求）即可。

实验十四（同步时序电路设计）实验报告

姓名：蒋仕彪 学号：3170102587 专业：求是科学班（计算机）1701

课程名称：逻辑与计算机设计基础实验

实验时间：2018-12-13

一、实验目的

- 掌握支持并行输入的移位寄存器的工作原理
- 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

2.1 实验内容：

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

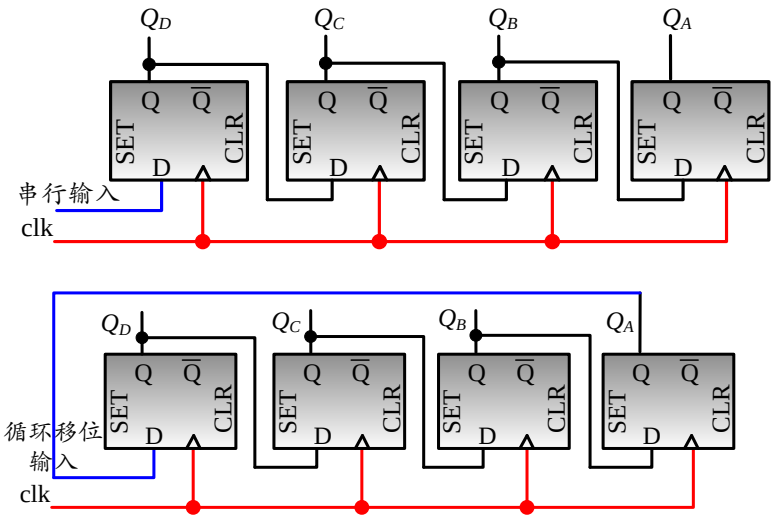
2.2 实验原理：

2.2.1 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

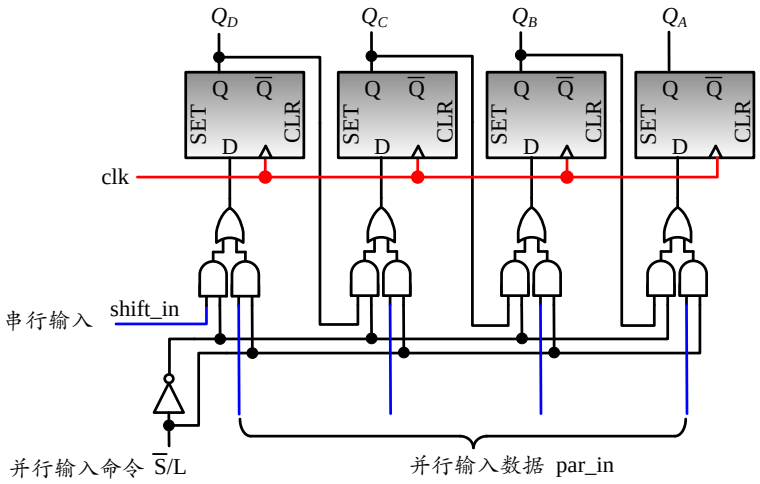
2.2.2 串行输入右移移位寄存器

- 使用 D 触发器构成串行输入的右移移位寄存器

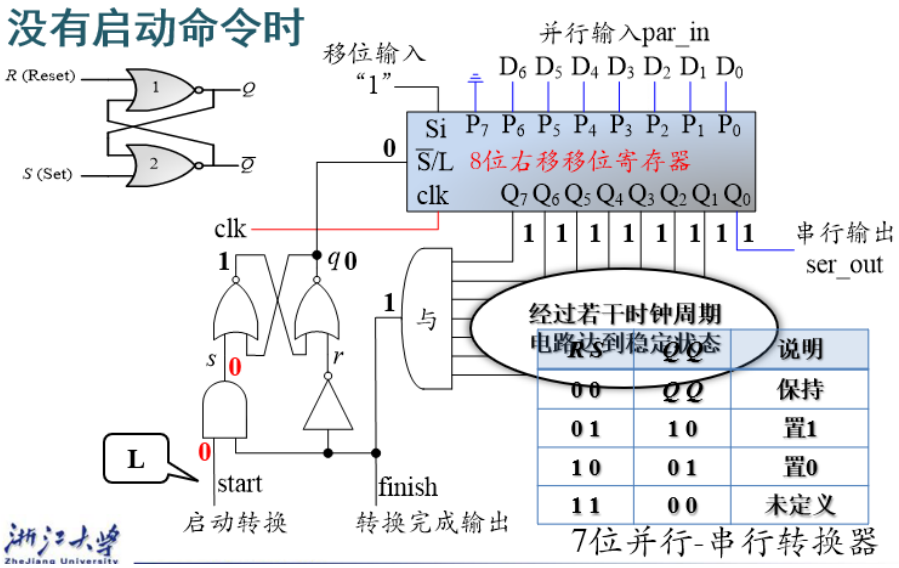


2.2.3 带并行输入的右移移位寄存器

- 数据输入方式：串行输入、并行输入

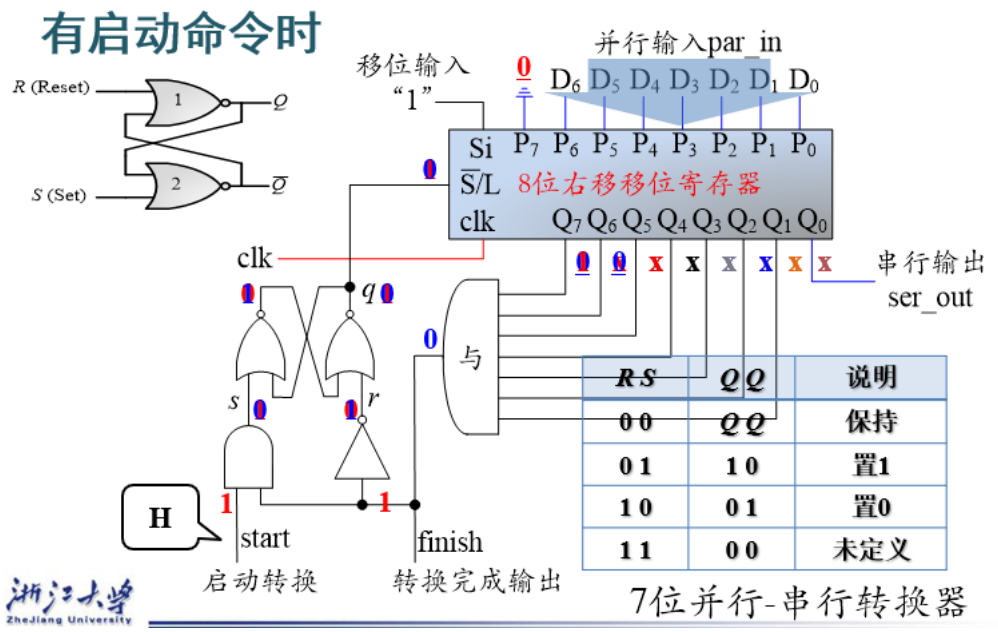


2.2.4 并行—串行转换器（1）



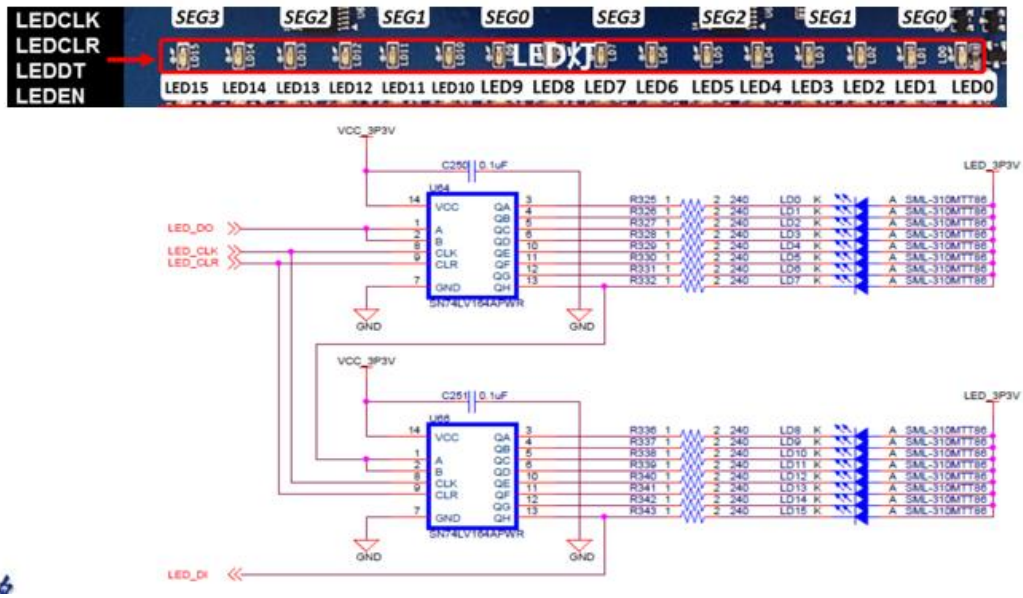
$\overline{R}$ $\overline{S}$	Q^{n+1} $\overline{Q}^{n+1}$	说明
0 0	Q^n $\overline{Q}^n$	保持
0 1	1 0	置1
1 0	0 1	置0
1 1	0 0	未定义

2.2.5 并行—串行转换器（2）



2.2.6 接口说明：主板 LED 灯

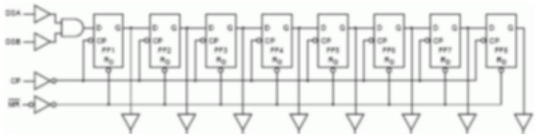
- 采用 2 个 74LV164A 构成 16 位串行输入并行输出移位寄存器
- 并行输出控制 16 个 LED 灯

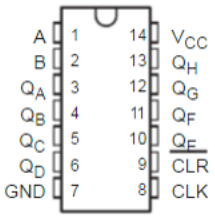


2.2.6 74LV164A

74LV164A

图 3. 逻辑图



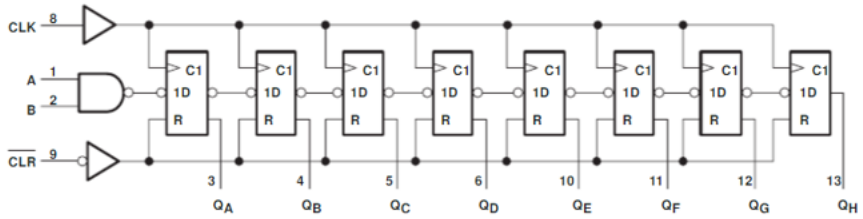


FUNCTION TABLE

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.



8 位串入、并出移位寄存器

1. 概述

74HC164、74HCT164 是高速硅门 CMOS 器件，与低功耗肖特基型 TTL (LSTTL) 器件的引脚兼容。74HC164、74HCT164 是 8 位边沿触发式移位寄存器，串行输入数据，然后并行输出。数据通过两个输入端（DSA 或 DSB）之一串行输入；任一输入端可以用作高电平使能端，控制另一输入端的数据输入。两个输入端或者连接在一起，或者把不用的输入端接高电平，一定不要悬空。

时钟 (CP) 每次由低变高时，数据右移一位，输入到 Q0，Q0 是两个数据输入端（DSA 和 DSB）的逻辑与，它将上升时钟沿之前保持一个建立时间的长度。

主复位 (MR) 输入端上的一个低电平将使其它所有输入端都无效，同时非同步地清除寄存器，强制所有的输出为低电平。

图 3. 逻辑图

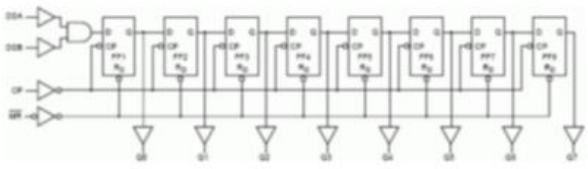


图 3. 逻

三、主要仪器设备

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

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

- 新建工程
  - 工程名称用 ShfitReg8b。
  - Top Level Source Type 用 HDL
- 用结构化描述设计
- 波形仿真

4.2 设计跑马灯应用用

- 新建工程
  - 工程名称用 MyMarquee。
  - Top Level Source Type 用 HDL
- 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
- 下载验证
  - 用 sw[0]和 sw[1]作为 regA 和 regB 的按键自增控制输入
  - sw[2]=1，并行输入，将{RegA,RegB}赋给移位寄存器
  - sw[2]=0，串行/循环右移移位
  - sw[4]作为移位寄存器的模式选择：
    - sw[4]=0，串行右移，串行输入值为 sw[3]
    - sw[4]=1，循环右移
  - 8 位的移位寄存器的值用 LED 灯表示：

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

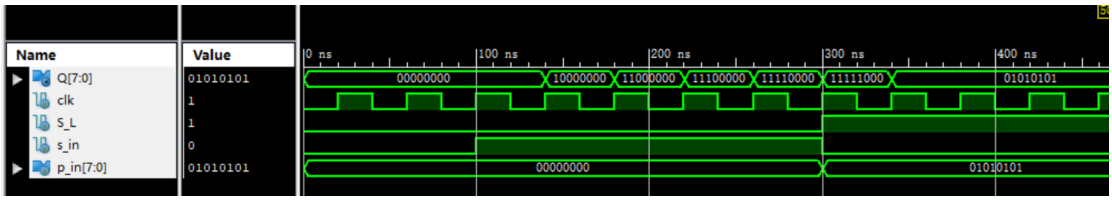
```
module shift_reg(
    input wire clk, S_L, s_in,
    input wire [7:0] p_in,
    output wire [7:0] Q
);
    wire [7:0] mid, L, R;
    wire INVS;
    FD FD7(.C(clk), .D(mid[7]), .Q(Q[7]));
    FD FD6(.C(clk), .D(mid[6]), .Q(Q[6]));
    FD FD5(.C(clk), .D(mid[5]), .Q(Q[5]));
    FD FD4(.C(clk), .D(mid[4]), .Q(Q[4]));
    FD FD3(.C(clk), .D(mid[3]), .Q(Q[3]));
    FD FD2(.C(clk), .D(mid[2]), .Q(Q[2]));
    FD FD1(.C(clk), .D(mid[1]), .Q(Q[1]));
    FD FD0(.C(clk), .D(mid[0]), .Q(Q[0]));

    OR2 M7(.I0(L[7]), .I1(R[7]), .O(mid[7]));
    OR2 M6(.I0(L[6]), .I1(R[6]), .O(mid[6]));
    OR2 M5(.I0(L[5]), .I1(R[5]), .O(mid[5]));
    OR2 M4(.I0(L[4]), .I1(R[4]), .O(mid[4]));
    OR2 M3(.I0(L[3]), .I1(R[3]), .O(mid[3]));
    OR2 M2(.I0(L[2]), .I1(R[2]), .O(mid[2]));
    OR2 M1(.I0(L[1]), .I1(R[1]), .O(mid[1]));
    OR2 M0(.I0(L[0]), .I1(R[0]), .O(mid[0]));

    INV IN(.I(S_L), .O(INVS));

    AND2 P7(.I0(s_in), .I1(INVS), .O(L[7]));
    AND2 P6(.I0(Q[7]), .I1(INVS), .O(L[6]));
    AND2 P5(.I0(Q[6]), .I1(INVS), .O(L[5]));
    AND2 P4(.I0(Q[5]), .I1(INVS), .O(L[4]));
    AND2 P3(.I0(Q[4]), .I1(INVS), .O(L[3]));
    AND2 P2(.I0(Q[3]), .I1(INVS), .O(L[2]));
    AND2 P1(.I0(Q[2]), .I1(INVS), .O(L[1]));
    AND2 P0(.I0(Q[1]), .I1(INVS), .O(L[0]));

    AND2 Q7(.I0(p_in[7]), .I1(S_L), .O(R[7]));
    AND2 Q6(.I0(p_in[6]), .I1(S_L), .O(R[6]));
    AND2 Q5(.I0(p_in[5]), .I1(S_L), .O(R[5]));
```

```
    AND2 Q4(.I0(p_in[4]), .I1(S_L), .O(R[4]));
    AND2 Q3(.I0(p_in[3]), .I1(S_L), .O(R[3]));
    AND2 Q2(.I0(p_in[2]), .I1(S_L), .O(R[2]));
    AND2 Q1(.I0(p_in[1]), .I1(S_L), .O(R[1]));
    AND2 Q0(.I0(p_in[0]), .I1(S_L), .O(R[0]));
endmodule
```

- 以上为 verilog 设计代码

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

        #100;

        // Add stimulus here
        S_L = 0;
        s_in = 1;
        p_in = 0;

        #200;
        S_L = 1;
        s_in = 0;
        p_in = 8'b01010101;
        #500;
    end
    always begin
        clk = 0;
        #20;
        clk = 1;
        #20;
    end

endmodule
```

- 以上为仿真代码



◘ 以上为仿真结果

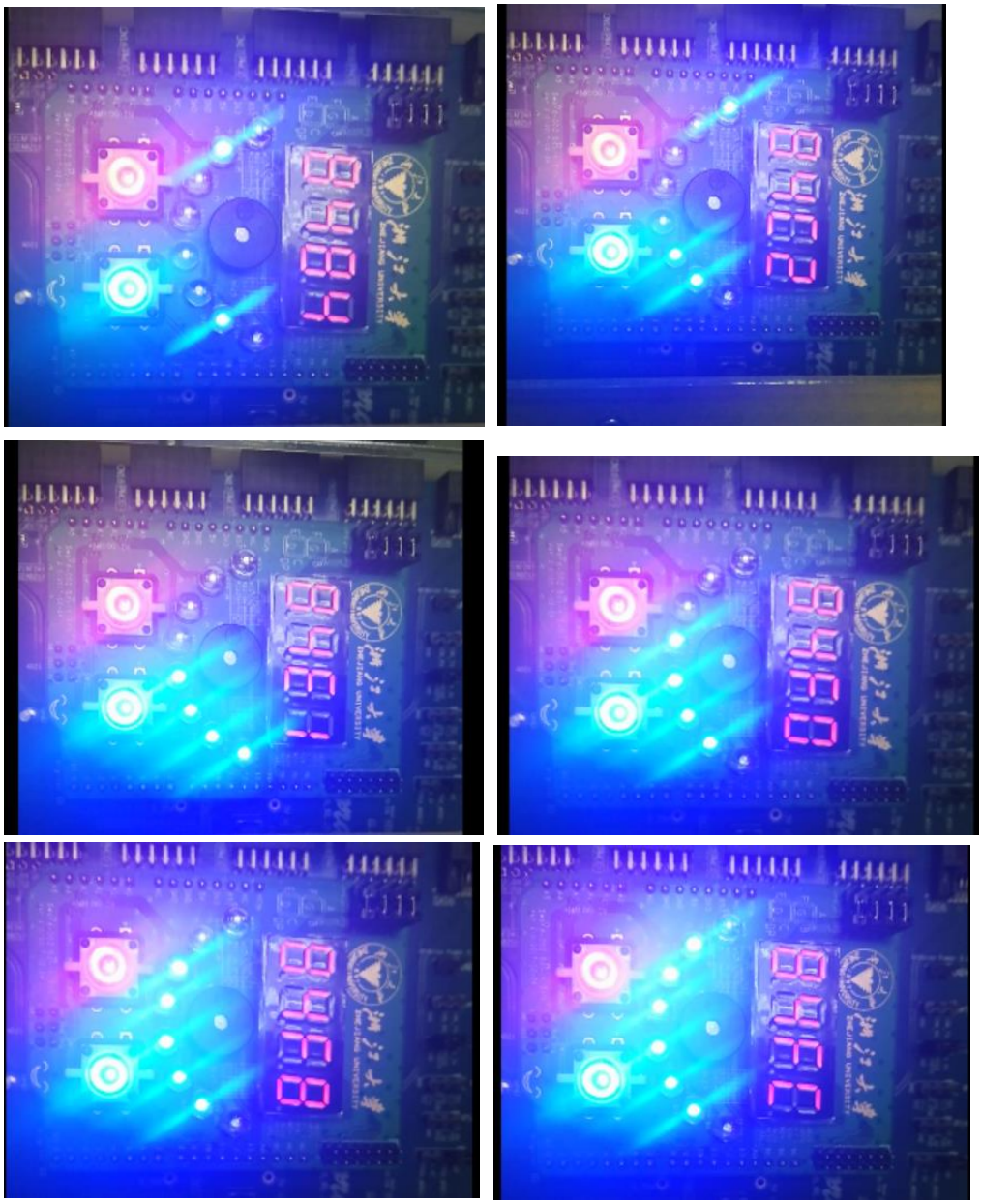
（注意到，在 100ns 出，串行输入 s_in 接入，所以在右移的同时不断多 1；在 200ns 后并行 s_L 开关启动，同时成功载入了 p_in[7:0]的数据。）

4.2 设计跑马灯应用

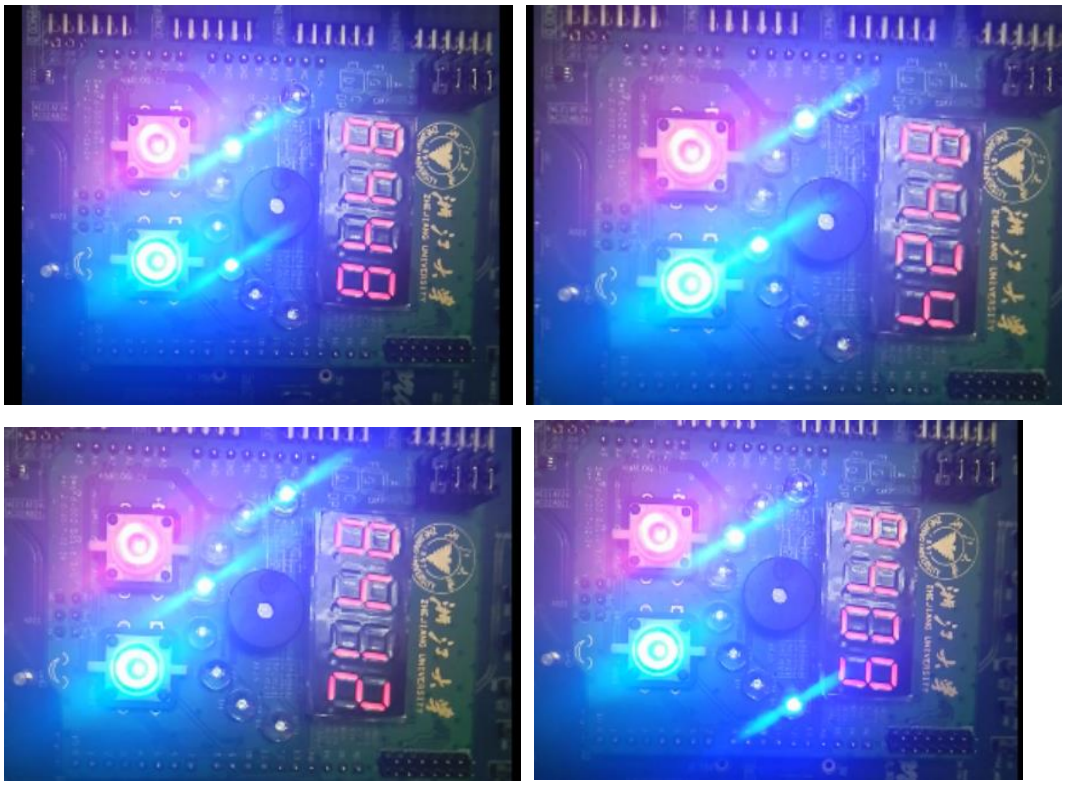
```
module Top(
    input wire clk,
     input wire [7:0] SW,
     output wire [7:0] SEGMENT,
     output wire [3:0] AN,
     output wire [7:0] LED
    );
     //wire [15:0]num;
     reg [7:0] num;
     wire [31:0] Time;
     wire clk_1s;

     clkdiv m0(clk,0,Time);
     assign clk_1s = Time[24];

     wire [3:0] A, B;

     AddSuber4b m1(.A(num[3:0]),.B(4'b0001),.Ctrl(0),.S(A));
     AddSuber4b m2(.A(num[7:4]),.B(4'b0001),.Ctrl(0),.S(B));

     always@(posedge SW[0]) num[3:0] = A;
     always@(posedge SW[1]) num[7:4] = B;

     assign s_in = SW[4] && LED[0] || (~SW[4]) && SW[3];

     shift_reg m3(clk_1s, SW[2], s_in, num, LED);
     disnum m4(clk, {num[7:0], LED}, 4'b0000, 4'b0000, 1'b0000, AN,
SEGMENT);
endmodule
```

```
    clk_100ms m1(clk,clk_100ms);

    disnum m2(clk, cnt , 4'b0 , 4'b0, 1'b0, AN, SEGMENT);

    assign LED[7:1] = 7'b1111111;

endmodule
```

◘ 以上为 top.v 的代码



◘ 上为成果展示 1（串行输入 1 并位移）



◘ 上为成果展示 2（普通位移）

六、实验心得

同样，因为可以用代码设计模块，shiftreg 模块设计的速度变快了，准确度变高了。不过，因为一共有八位，verilog 实现起来也稍微有些繁琐，每一种门的语句调用要复制八份然后改一改。

跑马灯的主要逻辑还是挺清晰的：和以前定义一样，SW[0]/SW[1]控制自增自减。SW[2]相当于是并行控制信号，=1 就把 reg 的值赋值到灯上来，=0 就不赋值。SW[4]是控制是普通循环右移还是串行循环右移。如果是串行循环右移的话，SW[3]是对应的串行输入键。总的来说，思路特别清晰，代码也能较快完成。

这个设计其实还有一个值得思考的地方，就是每次位移的 div 该怎么选。我最后选择一个肉眼看起来比较舒适的 Div[24]来进行操作。