

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:	王晓宇
学 院:	竺可桢学院
专 业:	计算机
邮 箱:	1657946908@qq.com
QQ 号:	1657946908
电 话:	19550222634
指导教师:	洪奇军
报告日期:	2023 年 11 月 30 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 and 寄存器传输设计____

学生姓名：____王晓宇____ 学号：____3220104364____ 同组学生姓名：____

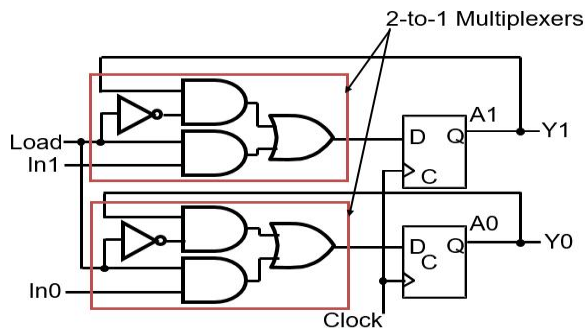
实验地点：____紫金港东四 509 室____ 实验日期：____2023____ 年 ____11____ 月 ____30____ 日

一、操作方法与实验步骤

➤实验原理

1. 寄存器

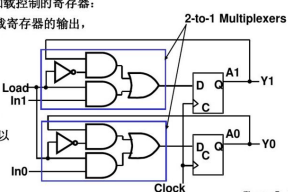
- 一组二进制存储单元
- 一个寄存器可以用于存储一系列二进制值，通常用于进行简单数据存储、移动和处理等操作
- 能存储信息并保存多个时钟周期，能用信号来控制“保存”或“加载”信息
- 如果 Load 信号为 1，允许时钟信号通过，如果为 0 则阻止时钟信号通过
- 例如：对于上升沿触发的边沿触发器
- 或负向脉冲触发的主从触发器
- 采用 Load 控制反馈的寄存器



■采用 Load 控制反馈的寄存器

7.1.4 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，且
 - 选择性地使用加载控制来改变寄存器的内容
- 例如：2位带并行加载控制的寄存器：
- 当 Load = 0 时，加载寄存器的输出，所以能保持当前值
- 当 Load = 1 时，加载输入数据，所以可以载入新数据
- 硬件上比门控时钟方法复杂，但是可以避免时序问题



2. 寄存器传输

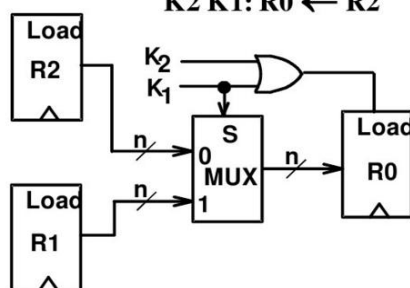
- 寄存器传输：寄存器中数据的传输和处理
- 三个基本单元：寄存器组、操作、操作控制
- 基本操作：加载、计数、移位、加法、按位操作等

3. 基于多路选择器总线的寄存器传输

7.6.1 基于多路选择器的传输

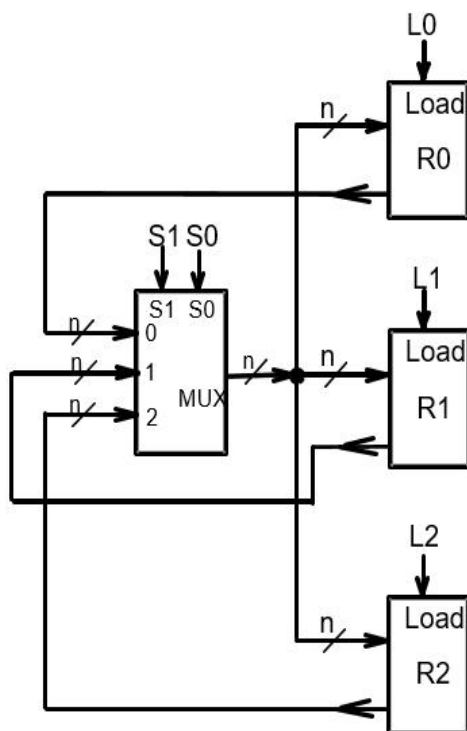
- 连接寄存器的多路选择器可以产生灵活的数据传输结构（注意：为清晰起见图中省略了时钟信号）

- 传输是： $K1: R0 \leftarrow R1$
 $K2 \bar{K1}: R0 \leftarrow R2$

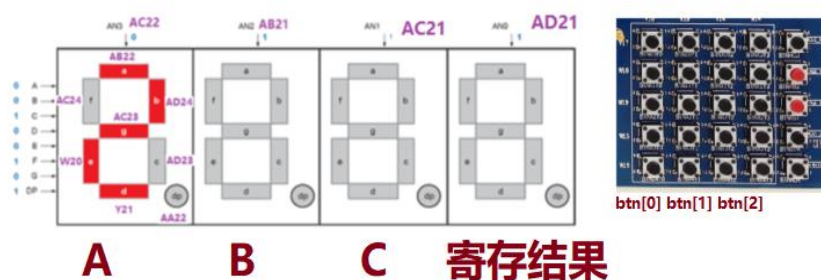


由一个多路选择器驱动的总线可以降低硬件开销

这个结构不能实现多个寄存器相互之间的并行传输操作

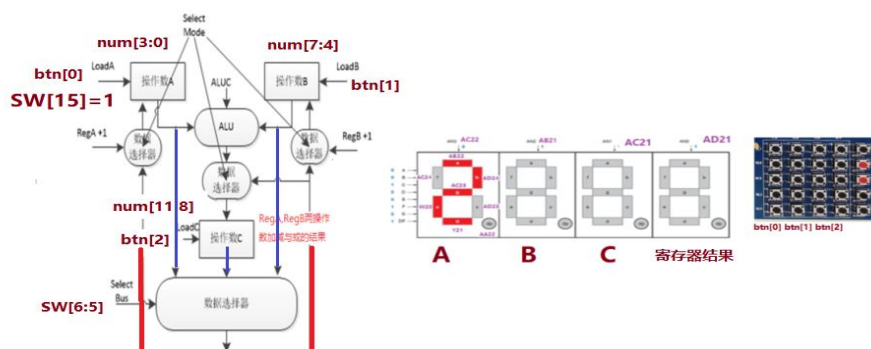


Mode2: 数据传输控制



A	B	C	寄存结果
00000000	00000000	00000000	00000000
00000001	00000001	00000001	00000001
00000010	00000010	00000010	00000010
00000011	00000011	00000011	00000011
00000100	00000100	00000100	00000100
00000101	00000101	00000101	00000101
00000110	00000110	00000110	00000110
00000111	00000111	00000111	00000111
00001000	00001000	00001000	00001000
00001001	00001001	00001001	00001001
00001010	00001010	00001010	00001010
00001011	00001011	00001011	00001011
00001100	00001100	00001100	00001100
00001101	00001101	00001101	00001101
00001110	00001110	00001110	00001110
00001111	00001111	00001111	00001111
00010000	00010000	00010000	00010000
00010001	00010001	00010001	00010001
00010010	00010010	00010010	00010010
00010011	00010011	00010011	00010011
00010100	00010100	00010100	00010100
00010101	00010101	00010101	00010101
00010110	00010110	00010110	00010110
00010111	00010111	00010111	00010111
00011000	00011000	00011000	00011000
00011001	00011001	00011001	00011001
00011010	00011010	00011010	00011010
00011011	00011011	00011011	00011011
00011100	00011100	00011100	00011100
00011101	00011101	00011101	00011101
00011110	00011110	00011110	00011110
00011111	00011111	00011111	00011111
00100000	00100000	00100000	00100000
00100001	00100001	00100001	00100001
00100010	00100010	00100010	00100010
00100011	00100011	00100011	00100011
00100100	00100100	00100100	00100100
00100101	00100101	00100101	00100101
00100110	00100110	00100110	00100110
00100111	00100111	00100111	00100111
00101000	00101000	00101000	00101000
00101001	00101001	00101001	00101001
00101010	00101010	00101010	00101010
00101011	00101011	00101011	00101011
00101100	00101100	00101100	00101100
00101101	00101101	00101101	00101101
00101110	00101110	00101110	00101110
00101111	00101111	00101111	00101111
00110000	00110000	00110000	00110000
00110001	00110001	00110001	00110001
00110010	00110010	00110010	00110010
00110011	00110011	00110011	00110011
00110100	00110100	00110100	00110100
00110101	00110101	00110101	00110101
00110110	00110110	00110110	00110110
00110111	00110111	00110111	00110111
00111000	00111000	00111000	00111000
00111001	00111001	00111001	00111001
00111010	00111010	00111010	00111010
00111011	00111011	00111011	00111011
00111100	00111100	00111100	00111100
00111101	00111101	00111101	00111101
00111110	00111110	00111110	00111110
00111111	00111111	00111111	00111111
01000000			

10-总线数据上的选择 C。按 btn[2] 存储到 num[11:8]

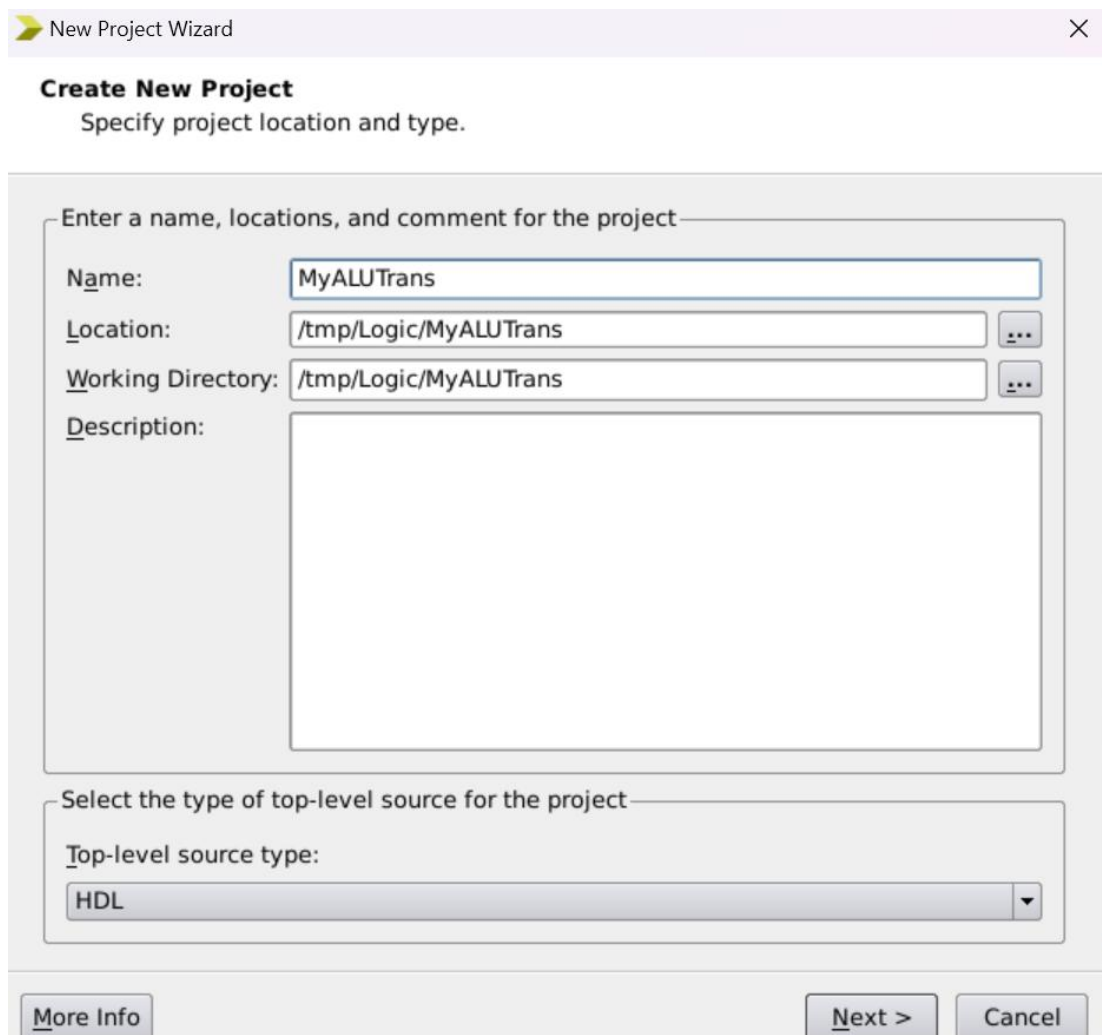


➤任务：基于 ALU 的数据传输应用设计

新建工程

工程名称用 MyALUTrans

Top Level Source Type 用 HDL



New Project Wizard

Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name: MyALUTrans

Location: /tmp/Logic/MyALUTrans

Working Directory: /tmp/Logic/MyALUTrans

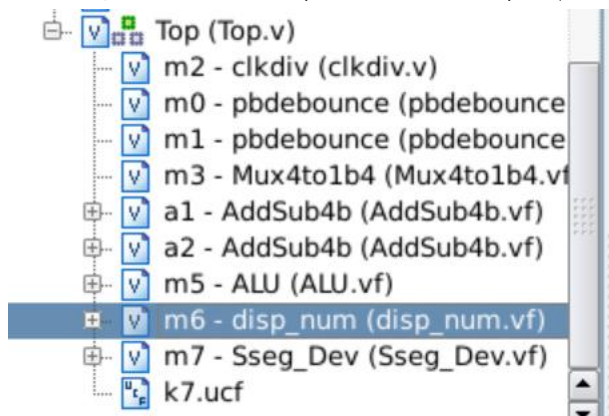
Description:

Select the type of top-level source for the project

Top-level source type: HDL

More Info Next > Cancel

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



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

Mode1:

按键控制输入:

btn[0] 作为按键信号控制 RegA(自加或自减)。

btn[1] 作为按键信号控制 RegB。 btn[2] 作为按键信号对 RegC 赋值。

按键加/减 1 控制:

sw[2]=0 加/1 减使 RegA(自加或自减), 对应 btn_out[0]即去抖动后的 BTN4Y0 按键。

sw[1]=0 加/1 减, RegB(自加或自减)对应 btn_out[1] 即去抖动后的 BTN4Y1 按键。

btn_out[2]即去抖动后的 BTN4Y2 按键。

ALU 运算控制: sw[4:3], 00-加, 01-减, 10-与, 11-或

(RegC 是 Reg A, Reg B 加减与或的结果)

SW[0] 图形、文本显示

SW[1]=0 按 btn[1] B=B+1

SW[1]=1 按 btn[1] B=B-1

SW[2]=0 按 btn[0] A=A+1

SW[2]=1 按 btn[0] A=A-1

SW[4:3]=00 C=A+B

SW[4:3]=01 C=A-B

SW[4:3]=10 C=A&B

SW[4:3]=11 C=A|B 按 btn[2] 寄存器结果=C



Mode2:

sw[15]=1 Mode2 数据传输控制

sw[6:5]对应 SelectBus: 00-总线数据选择 A, 01-总线数据选择 B, 10-选择 C

btn[0] LoadA(num[3:0]){总线数据存储在 num[3:0], 位置在 AN3}。

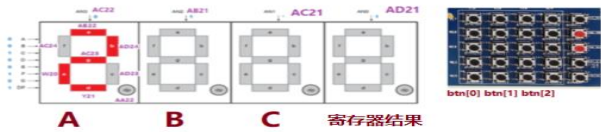
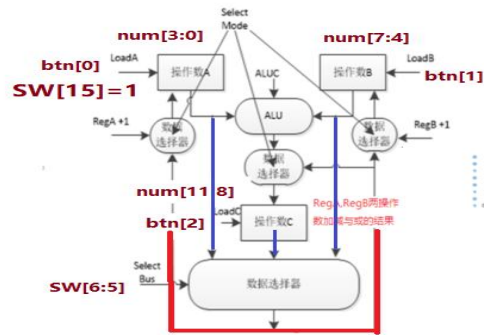
btn[1] LoadB(num[7:4]){总线数据存储在 num[7:4], 位置在 AN2}

btn[2] LoadC(ALU 运算结果) {总线数据存储在 num[11:8], 位置在 AN0}

(对 SW[6:5]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[3]: RegA,AN[2]: RegB,AN[1]:ALU 结果,AN[0]:Reg C



SW[6:5]=00 总线中存 A 即 num[3:0] 数据
 SW[6:5]=01 总线中存 B 即 num[7:4] 数据
 SW[6:5]=10 总线存 regC 即 num[11:8]
 按 btn[0] 总线数据存入寄存器 num[3:0]
 按 btn[1] 总线数据存入寄存器 num[7:4]
 按 btn[2] 总线数据存入 num[11:8]



Top 模块:

```
///////////////////////////////////////////////////////////////////
module Top(
    input wire clk,
    input wire [2:0] BTN,
    input wire [15:0] SW,
    output wire [3:0] AN,
    output wire [7:0] SEGMENT,
    output wire seg_clk,
    output wire seg_clrn,
    output wire seg_sout,
    output wire SEG_PEN
);

    reg [15:0] num;
    wire [15:0] cal_num;
    wire [2:0] btn_out;
    //wire [3:0] C;
    wire [3:0] Result;
    wire Co;
    wire [31:0] clk_div;
    wire [15:0] disp_hexs;
    assign disp_hexs[15:12] = num[3:0];
    assign disp_hexs[11:8] = num[7:4];
    assign disp_hexs[7:4] = num[11:8];
    assign disp_hexs[3:0] = Result;
    assign BTNX4 = 1'b0;
    wire [3:0] A2;
    wire [3:0] B2;
    wire [3:0] C2;
```



```

wire [3:0] A1;
wire [3:0] B1;
wire [3:0] C1;
clkdiv m4(clk,0,clk_div);
pbdebounce m0(clk_div[17],BTN[0],btn_out[0]);
pbdebounce m1(clk_div[17],BTN[1],btn_out[1]);
pbdebounce m2(clk_div[17],BTN[2],btn_out[2]);
ALU m5(.S(SW[4:3]),.A(cal_num[3:0]),.B(cal_num[7:4]),.C(C1),.Co(Co));
Mux4to1b4 m3(.I0(num[3:0]),.I1(num[7:4]),.I2(num[11:8]),.I3(4'b0000),.S(SW[6:5]),.o(Result));
AddSub4b a1(.A(num[3:0]),.B(4'b0001),.Ctrl(SW[0]),.S(A1));
AddSub4b a2(.A(num[7:4]),.B(4'b0001),.Ctrl(SW[1]),.S(B1));
assign cal_num = num;
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;

//CreateNumber m3(btn_out,SW1,num);

disp_num m6(.clk(clk),.HEXS((disp_hexs,disp_hexs)),.LES(4'b0000),.points(4'b0000),.RST(1'b0),.AN(AN),.segment(SEGMENT));

Sseg_Dev m7(.clk(clk),.rst(1'b0),.Start(clk_div[20]),.flash(clk_div[25]),.HEXS((disp_hexs,disp_hexs)),.point((8'b00000000)),.LES(8'b00000000)),.

endmodule

```

UCF 引脚:



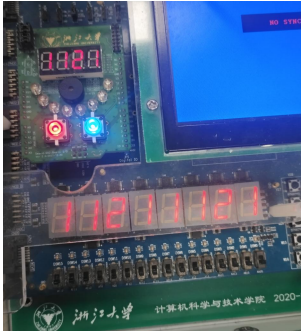
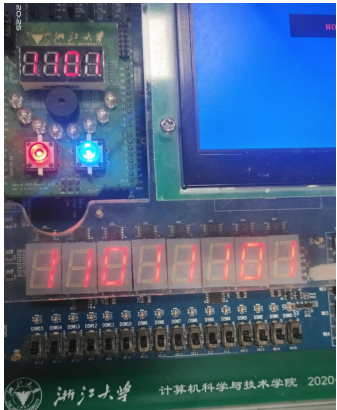
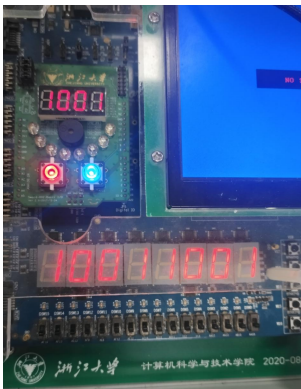
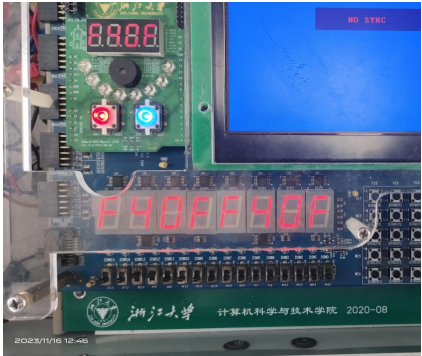
```

1 | NET "clk" LOC = AC18 | IOSTANDARD=LVCNMOS18;
2 | NET "BTN[0]" LOC=V18 | IOSTANDARD=LVCNMOS18;
3 | NET "BTN[0]" clock_dedicated_route = false;
4 | NET "BTN[1]" LOC=V19 | IOSTANDARD=LVCNMOS18;
5 | NET "BTN[1]" clock_dedicated_route = false;
6 | NET "BTN[2]" LOC=V14 | IOSTANDARD=LVCNMOS18;
7 | NET "BTN[2]" clock_dedicated_route = false;
8 | //NET "BTN[1]" LOC=W14 | IOSTANDARD=LVCNMOS18;
9 | //NET "BTN[1]" clock_dedicated_route = false;
0 | // NET "BTN[4]" LOC=W16 | IOSTANDARD=LVCNMOS18;
1 | //NET "led" LOC = W23 | IOSTANDARD=LVCNMOS33;
2 | NET "SW[0]" LOC=AF10 | IOSTANDARD=LVCNMOS15;
3 | NET "SW[1]" LOC=AF13 | IOSTANDARD=LVCNMOS15;
4 | NET "SW[2]" LOC=AE13 | IOSTANDARD=LVCNMOS15;
5 | NET "SW[3]" LOC=AF8 | IOSTANDARD=LVCNMOS15;
6 | NET "SW[15]" LOC=AA10 | IOSTANDARD=LVCNMOS15;
7 | NET "SW[14]" LOC=AB10 | IOSTANDARD=LVCNMOS15;
8 | NET "SW[13]" LOC=AA13 | IOSTANDARD=LVCNMOS15;
9 | NET "SW[12]" LOC=AA12 | IOSTANDARD=LVCNMOS15;
0 | NET "SW[11]" LOC=Y13 | IOSTANDARD=LVCNMOS15;
1 | NET "SW[10]" LOC=Y12 | IOSTANDARD=LVCNMOS15;
2 | NET "SW[9]" LOC=AD11 | IOSTANDARD=LVCNMOS15;
3 | NET "SW[8]" LOC=AD10 | IOSTANDARD=LVCNMOS15;
4 | NET "SW[7]" LOC=AE10 | IOSTANDARD=LVCNMOS15;
5 | NET "SW[6]" LOC=AE12 | IOSTANDARD=LVCNMOS15;
6 | NET "SW[5]" LOC=AF12 | IOSTANDARD=LVCNMOS15;
7 | NET "SW[4]" LOC=AE8 | IOSTANDARD=LVCNMOS15;
8 |
9 | NET "SEGMENT[0]" LOC=AB22 | IOSTANDARD=LVCNMOS33;#a
0 | NET "SEGMENT[1]" LOC=AD24 | IOSTANDARD=LVCNMOS33;#b
1 | NET "SEGMENT[2]" LOC=AD23 | IOSTANDARD=LVCNMOS33;#c
2 | NET "SEGMENT[3]" LOC=Y21 | IOSTANDARD=LVCNMOS33;#d
3 | NET "SEGMENT[4]" LOC=W20 | IOSTANDARD=LVCNMOS33;#e
4 | NET "SEGMENT[5]" LOC=AC24 | IOSTANDARD=LVCNMOS33;#f
5 | NET "SEGMENT[6]" LOC=AC23 | IOSTANDARD=LVCNMOS33;#g
6 | NET "SEGMENT[7]" LOC=AA22 | IOSTANDARD=LVCNMOS33;#point
7 | NET "AN[0]" LOC=AD21 | IOSTANDARD=LVCNMOS33;
8 | NET "AN[1]" LOC=AC21 | IOSTANDARD=LVCNMOS33;
9 | NET "AN[2]" LOC=AB21 | IOSTANDARD=LVCNMOS33;
0 | NET "AN[3]" LOC=AC22 | IOSTANDARD=LVCNMOS33;
1 |
2 | NET "seg_clk" LOC = M24 | IOSTANDARD=LVCNMOS33;
3 | NET "seg_clk" LOC = M20 | IOSTANDARD=LVCNMOS33;

```

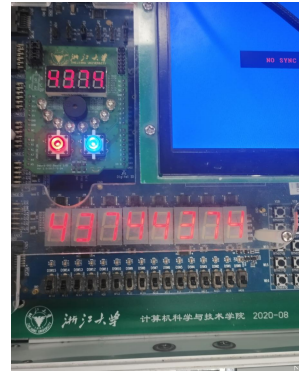
二、实验结果与分析

Mode1: ALU

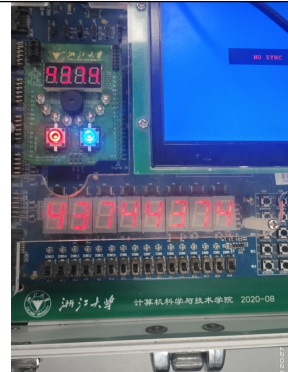
+	
-	
&	
	

Mode2: 寄存器传输

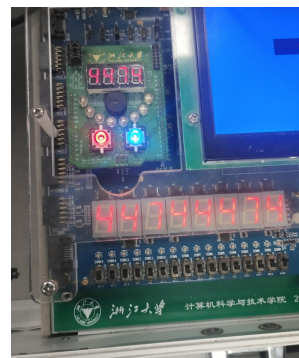
寄存器指向



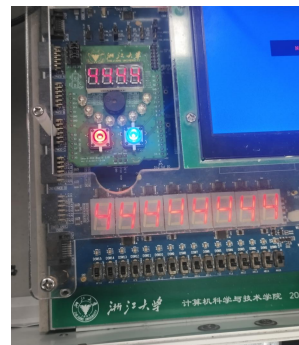
寄存器传输 1



寄存器传输 2



寄存器传输 3



三、讨论、心得

这次实验重新使用了 ALU 基础运算单元，相当于通过一个开关设定了两个模式：其中第一个模式是基本的 ALU 运算，第二个加入了寄存器模块，使得其他三位数码管能够读取寄存器的值并显示。总体难度不是太大，主要有之前实验设计的 ALU 单元直接使用，本次只需要 Mux4to1b4 即可实现寄存器的赋值，另外，为了实验简便，直接命名寄存器而不是使用 D 触发器之类的类寄存器。

实验心得的话，写点做实验的小技巧吧：其实每次的 top 文件生成 bit 文件的时候，第一次会很慢，第二次及以后只要你的改动不太大都可以很快生成 bit 文件。这样就给我们数码管显示的项目带来方便，因为你完全可以先将第一次未完成的项目先生成上板，看看是不是到这一步大致符合预期。因为一次完整的作业需要很多步的调试，你不能完全保证每一步都是对的，所以大概进行 70% 的时候就可以上一次板看看实验结果，这次实验我就按照这个办法测出我复制的 ALU.sym 有问题（上次跑完之后没有更新 sym 文件的生成），希望可以帮大家做快一点实验（。