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：	李瀚轩
学　　院：	竺可桢学院
系：	所在系
专　　业：	计算机科学与技术
学　　号：	3220106039
指导教师：	董亚波

2023 年 12 月 13 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: 寄存器及寄存器传输设计

学生姓名： 李瀚轩 专业： 计算机科学与技术 学号： 3220106039

同组学生姓名： 指导老师： 董亚波

实验地点： 东四 509 实验日期： 2023 年 12 月 7 日

一、实验目的和要求

1. 掌握寄存器传输电路的工作原理
2. 掌握寄存器传输电路的设计方法
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

1 实验内容

基于 ALU 的数据传输应用设计

2 原理

2.1 寄存器

寄存器是一组二进制存储单元，一个寄存器可以用于存储一列二进制值，通常用于进行简单数据存储，移动和处理等操作，它能存储信息并保存多个时钟周期，能够用信号来控制‘保存’或者‘加载’信息

2.2.1 采用门控时钟的寄存器

如果 Load 信号为 1，则允许时钟信号通过，如果为 0 则阻止

2.2.2 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。

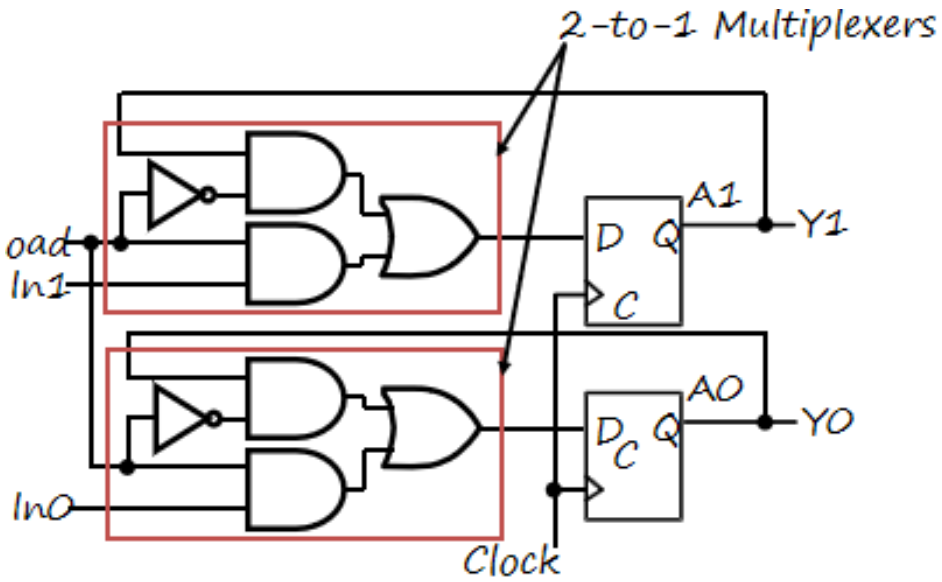

图 1: 采用 Load 控制反馈的寄存器

2.2 寄存器传输

2.2.1 寄存器传输方式

寄存器传输是寄存器中数据的传输和处理，有三个基本单元：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

2.2.2 采用寄存器传输原理的寄存器

- 功能：SW[2] 拨动一次，计一次数
- Load 控制模块：在 SW[2] 的上升沿产生一个时钟周期宽度的 Load 信号
- 自增/自减器可以用 4 位加减法器实现
- 功能：- SW[2]：寄存器加载
- SW[0]：向上/向下计数
- SW[15]：寄存器清零

2.3 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，但这个结构不能实现多个寄存器相互之间的并行传输操作

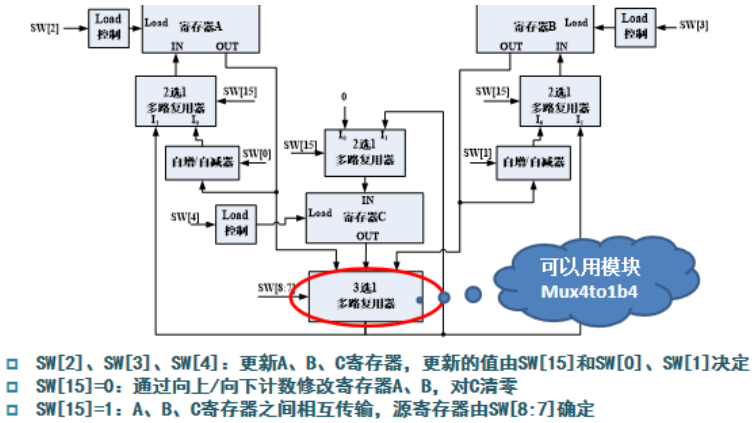

图 2: 基于多路选择器总线的寄存器传输

2.4 寄存器传输应用设计

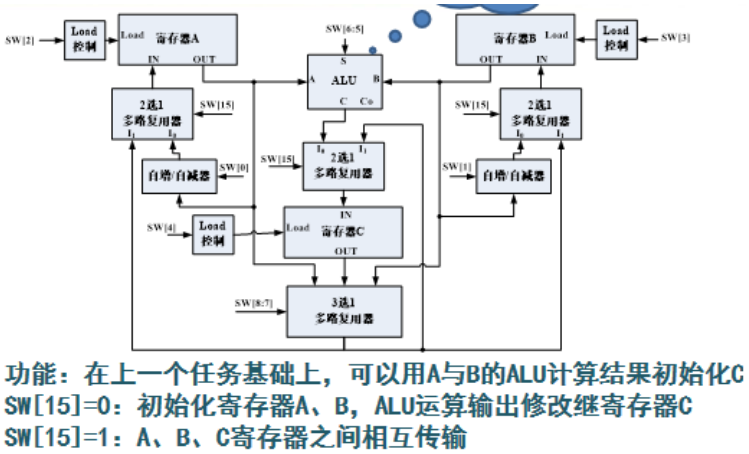

图 3: 寄存器传输应用设计

三、实验过程和数据记录

3.1 采用寄存器传输原理设计计数器

我们需要验证寄存器的设置初值功能，自增自减功能，并合理设计 4 位数码管上的显示内容。

1. 新建工程 MyRegCounter
2. 新建 verilog module 源文件 MyRegister4b,Verilog 代码如下：

```
module MyRegister4b(
input wire clk,
input [3:0] IN,
input wire Load,
output reg [3:0] OUT
);
initial OUT = 0;
always @(posedge clk) begin
```

```
        if (Load) OUT <= IN;
        end
        endmodule
```

3. 新建 Verilog 源文件 Load_Gen,Verilog 代码如下：

```
    module Load_Gen(
        input wire clk,
        input wire clk_1ms,
        input wire btn_in,
        output reg Load_out
    );
         initial Load_out=0;
         wire btn_out;
         reg old_btn;
         pbdebounce p0(clk_1ms,btn_in,btn_out);
         always @(posedge clk) begin
                 if((old_btn==1'b0)&&(btn_out==1'b1))
                         Load_out<=1'b1;
                 else
                         Load_out<=1'b0;
                 end
        always@(posedge clk) begin
                 old_btn<=btn_out;
        end
endmodule
```

4. 将之前实验课得到的模块 ADD SOURCE

5. 新建 top，set as top module，Verilog 代码如下：

```
module top(
        input clk,
        input [15:0] SW,

        output [3:0] AN,
        output [7:0] Segment
    );
```

```
    wire Load_A;
    wire [3:0] A,A_IN,A1;
    wire[31:0] clk_div;


    MyRegister4b RegA(.clk(clk),.IN(A_IN),.Load(Load_A),.OUT(A
    Load_Gen m0(.clk(clk),.clk_1ms(clk_div[17]),.btn_in(SW[2])
    clkdiv m3(clk,1'b0,clk_div);
    AddSub4b m4(.A(A),.B(4'b0001),.Ctrl(SW[0]),.S(A1));
    assign A_IN=(SW[15]==1'b0)?A1:4'b0000;
    DispNum m8(.clk(clk),.HEXS({A,A1,A_IN,4'b0000}),.LES(4'b0)
endmodule
```

6. 仿真

(a) 修改 Load_Gen 模块代码，去除输入引脚 clk_1ms，去掉按键去抖动模块，代码如下：

```
        module Load_Gen(
    input wire clk,
    input wire clk_1ms,
    input wire btn_in,
    output reg Load_out
);
     initial Load_out=0;
     //wire btn_out;
     reg old_btn;
     //pbdebounce p0(clk_1ms,btn_in,btn_out);
     always @(posedge clk) begin
            if((old_btn==1'b0)&&(btn_in==1'b1))
                    Load_out<=1'b1;
            else
                    Load_out<=1'b0;
            end
    always@(posedge clk) begin
            old_btn<=btn_in;
```

```
        end
endmodule
```

(b) 修改 top.v 代码，去掉 DispNum 模块，在输入端口增加 output wire [15:0] num, 去掉端口 AN,SEGMENT, 把 A 寄存器，自增自减 1 的结果，寄存器当前的值作为仿真结果输出，代码如下：

```
        module top(
        input clk,
        input [15:0] SW,
        output wire [15:0] num,
        //output [3:0] AN,
        //output [7:0] Segment
    );

         wire Load_A;
         wire [3:0] A,A_IN,A1;
         wire[31:0] clk_div;
        assign num = {A, A1, A_IN, 4'b0000};
         MyRegister4b RegA(.clk(clk),.IN(A_IN),.Load(Load_A),.O
         Load_Gen m0(.clk(clk),.clk_1ms(clk_div[17]),.btn_in(SW
         clkdiv m3(clk,1'b0,clk_div);
         AddSub4b m4(.A(A),.B(4'b0001),.Ctrl(SW[0]),.S(A1));
         assign A_IN=(SW[15]==1'b0)?A1:4'b0000;
         //DispNum m8(.clk(clk),.HEXS({A,A1,A_IN,4'b0000}),.LES
endmodule
```

(c) 仿真激励代码：

```
        module top_sim;

        // Inputs
        reg clk;
        reg [15:0] SW;
        wire [15:0] num;

        // Instantiate the Unit Under Test (UUT)
        top uut (
```

```
        .clk(clk),
        .SW(SW),
        .num(num)
    );

    initial begin
        // Initialize Inputs
            SW = 0;
            SW[15] = 1;
        SW[2] = 0; #50
        SW[2] = 1; #40
        SW[15] = 0;
        SW[0] = 0;
        SW[2] = 0;#60
        SW[2] = 0;#50
        SW[2] = 1;#50
        SW[2] = 0;#50
        SW[2] = 1;#50
        SW[2] = 0;#50
        // Wait 100 ns for global reset to finish
        #100;
    end
      always begin
            clk=1;#10;
            clk=0;#10;
        // Add stimulus here


    end

endmodule
```

得到如下波形图：

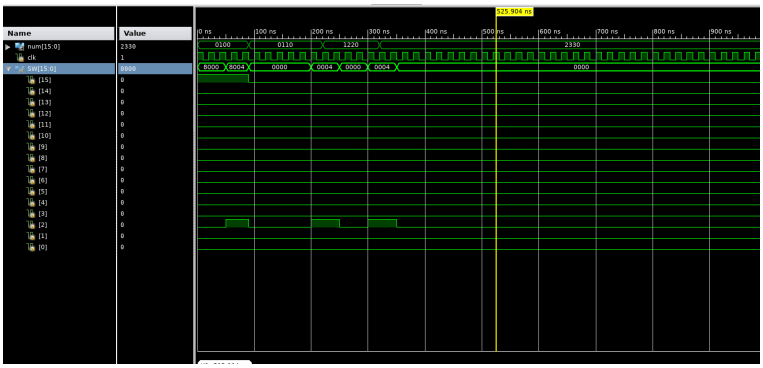

图 4: 仿真波形图

可以看到，SW[15]=1 时 2to1Mux 选择 0，对寄存器赋初值。每次拨动并驳回 SW[2] 可以使寄存器写入新的值。仿真结果符合预期。上板结果也符合预期，具体图片见任务三。

7. 引脚约束文件

```
NET "SEGMENT[0]" LOC = AB22 | IOSTANDARD = LVCMOS33 ;#a
NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33 ;#b
NET "SEGMENT[2]" LOC = AD23 | IOSTANDARD = LVCMOS33 ;#c
NET "SEGMENT[3]" LOC = Y21 | IOSTANDARD = LVCMOS33 ;#d
NET "SEGMENT[4]" LOC = W20 | IOSTANDARD = LVCMOS33 ;#e
NET "SEGMENT[5]" LOC = AC24 | IOSTANDARD = LVCMOS33 ;#f
NET "SEGMENT[6]" LOC = AC23 | IOSTANDARD = LVCMOS33 ;#g
NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33 ;#point

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

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

NET "SW[0]" LOC = AA10 | IOSTANDARD = LVCMOS15;

NET "SW[2]" LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "SW[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
```

3.2 基于多路选择器总线的寄存器传输

我们需要验证 ABC 寄存器的设置初值功能，AB 寄存器的自增，自减功能，并验证 ABC 寄存器之间的传输功能，并合理设计 4 位数码管上的显示内容

1. 新建工程 RegDataPathTrans，Top Level Source Type 为 HDL.
2. 将之前实验课得到的模块 ADD SOURCE
3. 新建 Verilog 源文件 top, 并设置为 top module，代码如下：

```
        module top(
input clk,
input [15:0] SW,
output [3:0] AN,
output [7:0] Segment
);
wire [3:0] A_IN, A_I0, A_OUT;
wire [3:0] B_IN, B_I0, B_OUT;
wire [3:0] C_IN, C_I0, C_OUT;
wire [3:0] I1;
wire [31:0] clk_div;

clkdiv m0(clk,1'b0,clk_div);
MyRegister4b m1(.clk(clk), .IN(A_IN), .Load(Load_A), .OUT(A_OUT));
Load_Gen m2(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[2]), .Loa
AddSub4b m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]),.S(A_I0));
assign A_IN = (SW[15] == 1'b0)? A_I0: I1;
MyRegister4b m4(.clk(clk), .IN(B_IN), .Load(Load_B), .OUT(B_OUT));
Load_Gen m5(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[3]),.Load
AddSub4b m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]),.S(B_I0));
assign B_IN = (SW[15] == 1'b0)? B_I0: I1;
MyRegister4b m7(.clk(clk), .IN(C_IN), .Load(Load_C), .OUT(C_OUT));
Load_Gen m8(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[4]),.Load
assign C_IN = (SW[15] == 1'b1)? I1: 4'b0000;
Mux4to1b4 m9( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT),.I3(4'b0000), .s(
DispNum m10(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,4'b0000}), .LES(
```

```
endmodule
```

4. 引脚约束文件如下：

```
NET "SEGMENT[0]" LOC = AB22 | IOSTANDARD = LVCMOS33 ;#a
NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33 ;#b
NET "SEGMENT[2]" LOC = AD23 | IOSTANDARD = LVCMOS33 ;#c
NET "SEGMENT[3]" LOC = Y21 | IOSTANDARD = LVCMOS33 ;#d
NET "SEGMENT[4]" LOC = W20 | IOSTANDARD = LVCMOS33 ;#e
NET "SEGMENT[5]" LOC = AC24 | IOSTANDARD = LVCMOS33 ;#f
NET "SEGMENT[6]" LOC = AC23 | IOSTANDARD = LVCMOS33 ;#g
NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33 ;#point

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

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[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[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
```

上板结果符合预期，将在任务三中给出。

3.3 基于 ALU 的数据传输应用设计

、任务三，我们需要验证 ABC 寄存器的设置初值功能，验证 A，B 寄存器的自增，自减功能；验证 ALU 运算功能；验证寄存器传输功能；合理设计 4 位数码管上显示的内容。

1. 新建工程 MyALUTrans

2. 将之前实验课得到的模块 ADD Source

3. 新建 Verilog 源文件 top, 并设置为 top module，代码如下：

```
module top(
input clk,
input [15:0] SW,
//output wire [15:0] num
output [3:0] AN,
output [7:0] Segment
);

wire Load_A, Load_B, Load_C, carry;
wire [3:0] A_IN, A_I0, A_OUT;
wire [3:0] B_IN, B_I0, B_OUT;
wire [3:0] C_IN, C_I0, C_OUT;
wire [3:0] I1, I0;
wire [31:0] clk_div;

//assign num={A_OUT,B_OUT,C_OUT,I1};

clkdiv m0(clk, 1'b0, clk_div);
MyRegister4b m1(.clk(clk), .IN(A_IN), .Load(Load_A), .OUT(A
Load_Gen m2(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[2]
AddSub4b m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]),.S(A_I0));
assign A_IN = (SW[15] == 1'b0)? A_I0: I1;
MyRegister4b m4(.clk(clk), .IN(B_IN), .Load(Load_B), .OUT(B
Load_Gen m5(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[3]
AddSub4b m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]),.S(B_I0));
```

```
        assign B_IN = (SW[15] == 1'b0)? B_I0: I1;
        MyRegister4b m7(.clk(clk), .IN(C_IN), .Load(Load_C), .OUT(C
Load_Gen m8(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[4]), .Loa
        assign C_IN = (SW[15] == 1'b1)? I1: I0;
        myALU m9( .S(SW[6:5]), .A(A_OUT), .B(B_OUT), .C(I0), .Co(ca
        Mux4to1b4 m10( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT),.I3(4'b00
        DispNum m11(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,I1}), .LE

endmodule
```

4. 仿真

(a) 修改 Load_Gen 模块代码，与任务一相同

(b) 修改 top.v 代码，代码如下：

```
        module top(
    input clk,
    input [15:0] SW,
    output wire [15:0] num
    //output [3:0] AN,
    //output [7:0] Segment
);

    wire Load_A, Load_B, Load_C, carry;
    wire [3:0] A_IN, A_I0, A_OUT;
    wire [3:0] B_IN, B_I0, B_OUT;
    wire [3:0] C_IN, C_I0, C_OUT;
    wire [3:0] I1, I0;
    wire [31:0] clk_div;

    assign num={A_OUT,B_OUT,C_OUT,I1};

    clkdiv m0(clk, 1'b0, clk_div);
    MyRegister4b m1(.clk(clk), .IN(A_IN), .Load(Load_A), .O
    Load_Gen m2(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW
    AddSub4b m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]),.S(A_I
    assign A_IN = (SW[15] == 1'b0)? A_I0: I1;
```

```
        MyRegister4b m4(.clk(clk), .IN(B_IN), .Load(Load_B), .O
        Load_Gen m5(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW
        AddSub4b m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]),.S(B_I
        assign B_IN = (SW[15] == 1'b0)? B_I0: I1;
        MyRegister4b m7(.clk(clk), .IN(C_IN), .Load(Load_C), .O
Load_Gen m8(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[4]),
        assign C_IN = (SW[15] == 1'b1)? I1: I0;
        myALU m9( .S(SW[6:5]), .A(A_OUT), .B(B_OUT), .C(I0), .C
        Mux4to1b4 m10( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT),.I3(4
        //DispNum m11(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,I1}

endmodule
```

仿真代码如下：

```
module top_sim;

        // Inputs
        reg clk;
        reg [15:0] SW;

        // Outputs
        wire [15:0] num;

        // Instantiate the Unit Under Test (UUT)
        top uut (
                .clk(clk),
                .SW(SW),
                .num(num)
        );

        initial begin
                // Initialize Inputs
                SW = 0;
         SW[15] = 0;
         SW[0]=0;#20
```

```
 SW[2] = 1;#50
 SW[2] = 0;#50
 SW[2] = 1;#50
 SW[2] = 0;#50
 SW[2] = 1;#50
 SW[2] = 0;#50
 //SW[2] = 1;#50
 //SW[2] = 0;#50
        SW[1] = 1;
 SW[3] = 1;#50
 SW[3] = 0;#50
 SW[3] = 1;#50
 SW[3] = 0;#50;

 SW[6:5] = 2'b01;
 SW[4] = 1;#50
 SW[4] = 0;#50;

 SW[15] = 1;
 SW[8:7] = 2'b10;
 SW[2] = 1; #50
 SW[2] = 0; #50;

 SW[8:7] = 2'b01;
 SW[4] = 1;#50
 SW[4] = 0;#50;
        // Wait 100 ns for global reset to finish
end
always begin
        clk=1;#10;
        clk=0;#10;

        // Add stimulus here

end
```

```
endmodule
```

得到如下波形图

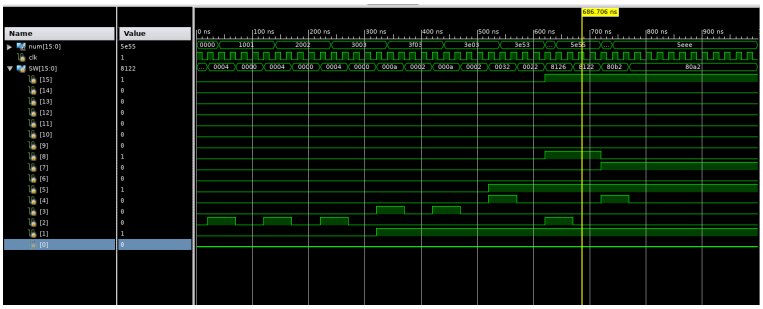

图 5: 仿真波形图

SW[15]=0 时电路是自增自减功能，通过开关 SW[0] 和 SW[1] 控制 A，B 寄存器的自增自减，最开始 SW[15]=0,SW[0]=0 表示对寄存器 A 选择自增功能，拨动 SW[2] 可以让寄存器自增，SW[1]=1 时拨动 SW[3] 让 B 自减。SW[6:5]=01 代表进行减法运算，拨动 SW[4] 可以将运算后的值赋给寄存器 C SW[15]=1 时利用 bus 传输数据，使用 3to1MUX 选择源寄存器，拨动 SW[8:7]=10 时代表传输 C 寄存器，拨动 SW[2] 就将 C 存入 A。仿真结果符合预期。

5. 上板验证，K7 如下：

```
NET "SEGMENT[0]" LOC = AB22 | IOSTANDARD = LVCMOS33 ;#a
NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33 ;#b
NET "SEGMENT[2]" LOC = AD23 | IOSTANDARD = LVCMOS33 ;#c
NET "SEGMENT[3]" LOC = Y21 | IOSTANDARD = LVCMOS33 ;#d
NET "SEGMENT[4]" LOC = W20 | IOSTANDARD = LVCMOS33 ;#e
NET "SEGMENT[5]" LOC = AC24 | IOSTANDARD = LVCMOS33 ;#f
NET "SEGMENT[6]" LOC = AC23 | IOSTANDARD = LVCMOS33 ;#g
NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33 ;#point

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

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[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[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;
```

6. 上板结果分析：总体功能已在验收时验证正确性，这里列举部分结果：

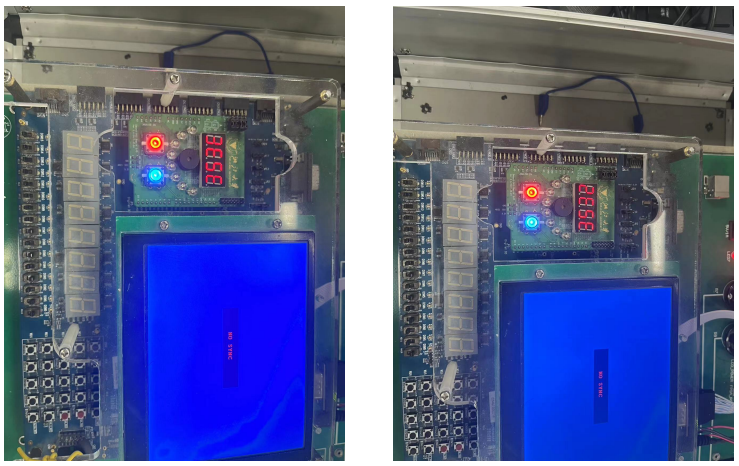

图 6: 验证 ALU 运算功能

由上图可看到 SW[15]=0,sw[0]=1，寄存器 A 自减，此时 SW[6:5]=01 为减法，拨动 SW[4] 后 C 的结果符合预期，AB 自增自减功能符合预期。

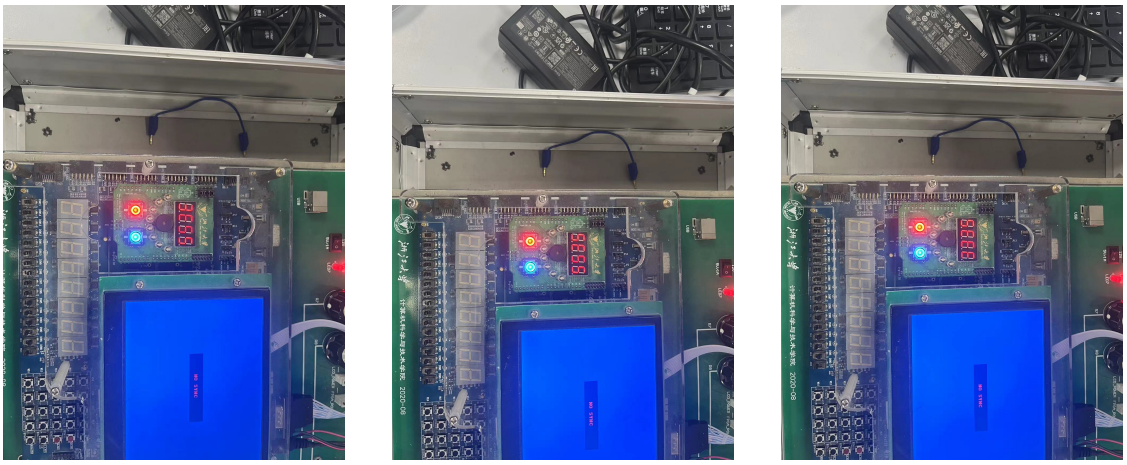

图 7: 验证寄存器传输功能

由上图可看到此时 SW[15]=1, 利用 bus 传输数据，首先 SW[8:7]=10，拨动 SW[2] 将 C 传输到 A，结果符合预期，再 SW[8:7]=01, 拨动 SW[4] 将 B 传输到 C，结果也符合预期。

四、实验结果分析

实验结果符合要求，仿真波形符合预期，Verilog 代码分析也已经给出。在任务三仿真的时候，我发现 Load_Gen 信号产生需要时间，如果在 clk 上升沿的那一刻改变则不会写入寄存器，必须提前准备好，因此在仿真的时候需要注意，不然在代码上有四次改变，实际上只加了 3。我觉得这就对应着上课讲的 setup_time，即需要提前准备好信号。

五、讨论与心得

这是第一次全部用 verilog 代码实现的实验，不用眼花缭乱地连线了 (ˇ ˇ)。但是在写 verilog 代码的时候难免会出现错误，比如打字的偏差，不过也比较好修正就是了。不幸的是，由于一些细细碎碎的错误，包括产生.bit 文件报错，以及上板后发现输出的错误，这次实验我做到了比较晚，这可能是第一个没有提前验收的使用 ISE 的实验叭。但是也提高了我的纠错与分析的能力。总而言之，虽然过程艰辛，但是结果是好的。