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：	龙永奇
学　　院：	计算机科学与技术学院
系：	本系
专　　业：	计算机科学与技术
学　　号：	3220105907
指导教师：	董亚波

2023 年 12 月 19 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：寄存器和寄存器传输设计

学生姓名：龙永奇 专业：计算机科学与技术 学号：3220105907

同组学生姓名：贾一多 指导老师：董亚波

实验地点：东 4-509 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

内容：

1. 任务：基于 ALU 的数据传输应用设计

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. **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：

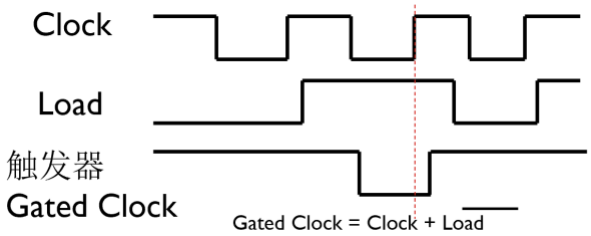

图 2.1 采用门控时钟寄存器

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，且选择性地使用加载控制来改变寄存器的内容：

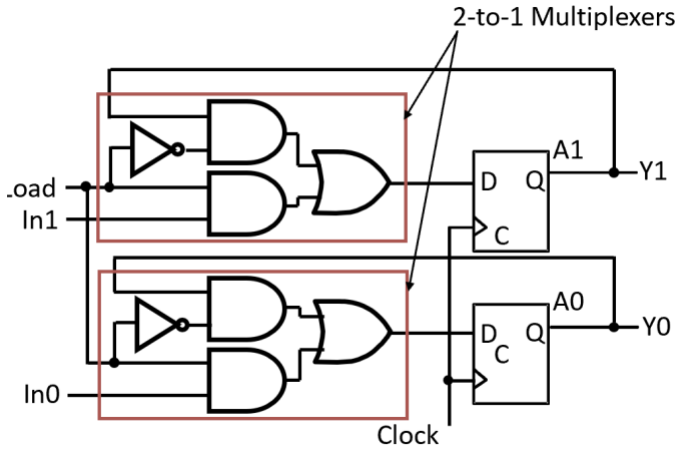

图 2.2 采用 Load 控制反馈的寄存器

```
reg [3:0] OUT;
……
always @ (posedge clk) begin
    if (Load) OUT <= IN;
end
……
```

4. 寄存器传输

- 寄存器传输：寄存器中数据的传输和处理

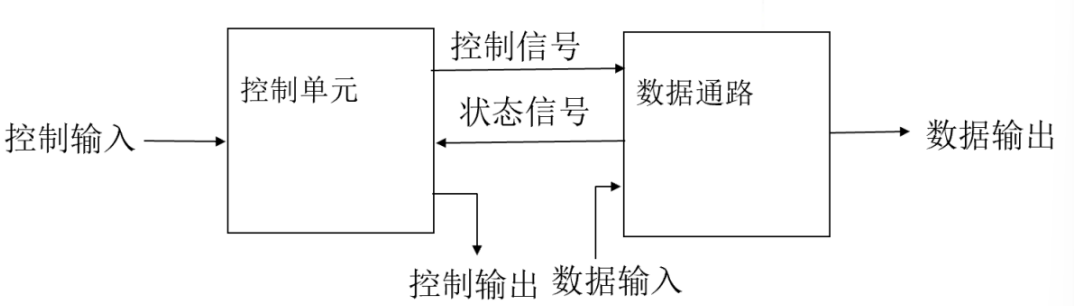

图 2.3 寄存器传输方式

- 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

5. **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 1 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/下计数

     SW[15]：寄存器清零

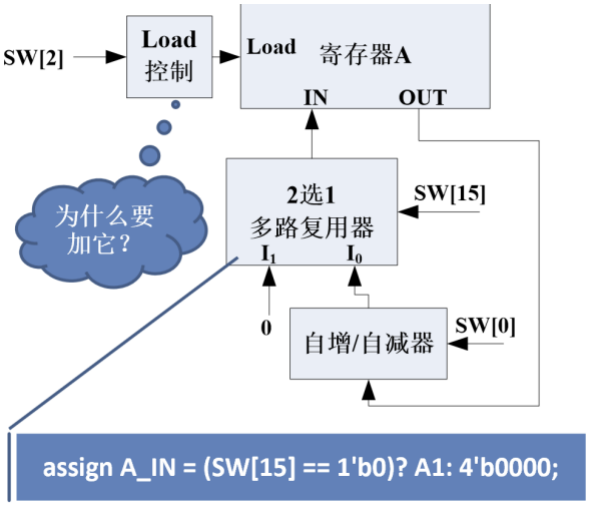

图 2.4 采用寄存器传输原理的寄存器示意图

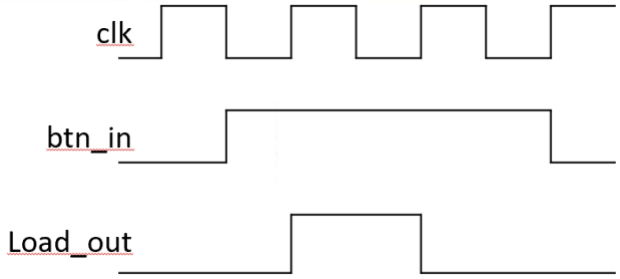

图 2.5 对应波形图

```
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)) //btn 出现上升
沿
            Load_out <= 1'b1;
        else
            Load_out <= 1'b0;
     end
     always@(posedge clk) begin      //保存上一个周期 btn 的状态
        old_btn <= btn_out;
     end
endmodule
```

6. **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

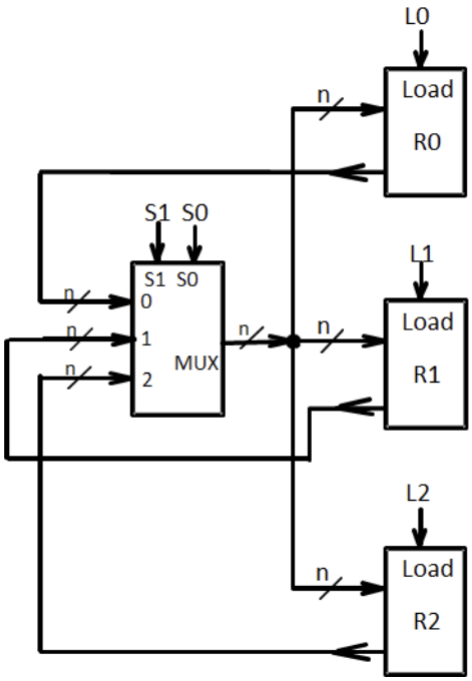

图 2.6 基于多路选择器总线的寄存器传输

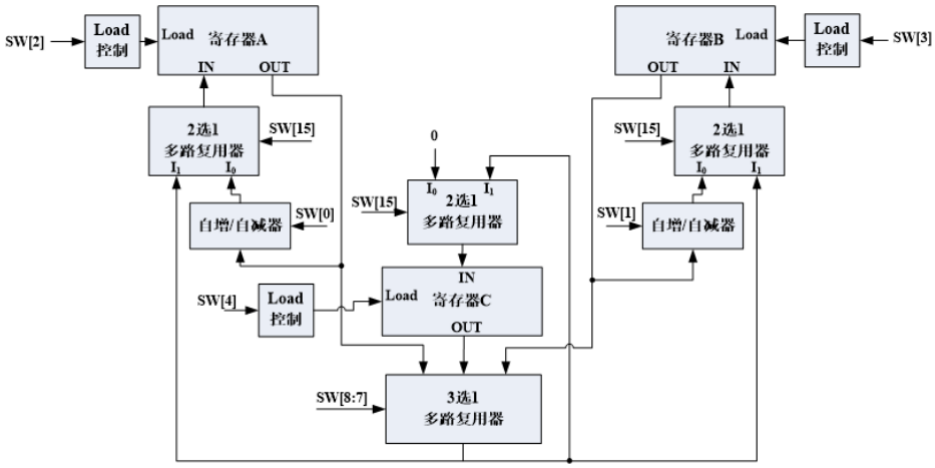

图 2.7 基于多路选择器总线的寄存器传输流程图

- SW[2]、SW[3]、SW[4]：更新 A、B、C 寄存器，更新的值由 SW[15]和 SW[0]、SW[1]决定
- SW[15]=0：通过向上/向下计数修改寄存器 A、B，对 C 清零
- SW[15]=1：A、B、C 寄存器之间相互传输，源寄存器由 SW[8:7]确定

7. 寄存器传输应用设计

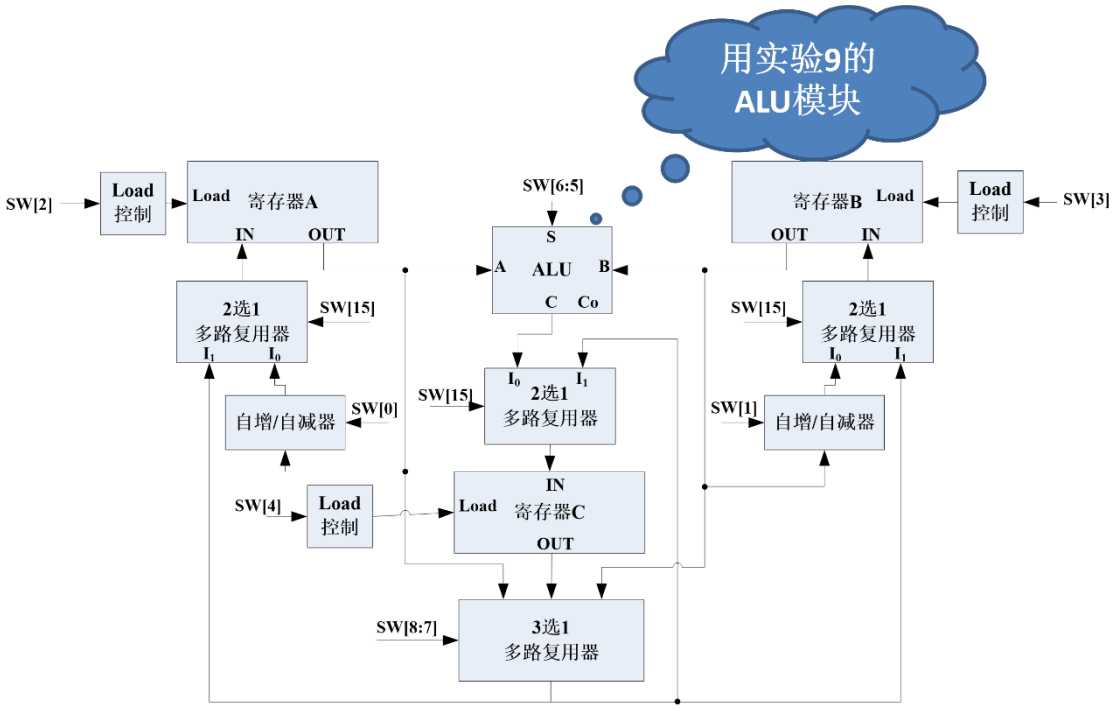

图 2.8 寄存器传输应用设计流程图

- 功能：在上一个任务基础上，可以用 A 与 B 的 ALU 计算结果初始化 C
- SW[15]=0：初始化寄存器 A、B，ALU 运算输出修改继寄存器 C

- SW[15]=1：A、B、C 寄存器之间相互传输

三、实验过程和数据记录

1. 采用寄存器传输原理设计计数器

(a)新建工程 MyRegCounter，Top Level Source 为 HDL

(b)新建 Verilog 类型源文件 MyRegister4b

(c)用 Verilog 代码方式设计，输入代码如下：

```
`timescale 1ns / 1ps
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
```

(d)新建 Verilog 源文件 Load_Gen，输入代码如下：

```
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)) //btn 出现
上升沿
            Load_out <= 1'b1;
        else
            Load_out <= 1'b0;
     end
     always@(posedge clk) begin     //保存上一个周期 btn 的状态
```

```
        old_btn <= btn_out;
    end
endmodule
```

(e)将之前设计的 clkdiv，AddSub4b 模块调入项目中

(f)新建 Top 文件，右键 Set as Top Module，用 Verilog 代码方式设计，输入代码如下：

```
`timescale 1ns / 1ps
module Top(
        input clk,
        input [15:0] SW,
        output [3:0] AN,
        output [7:0] Segment
        output wire [15:0] num // new
        );

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

        assign num = {A, A1, A_IN, 4'b0000}; // new

        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]), .Load_out
(Load_A));
        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_sch m8(.clk(clk), .HEXS({A, A1, A_IN,
4'b0000}), .LES(4'b0), .points(4'b0), .RST(1'b0), .AN(AN), .Se
gment(SEGMENT));
endmodule
```

(g)建立仿真文件 Top_sim.v，根据仿真要求：

修改 P13 的 Load_Gen 模块代码

去掉输入引脚：input wire clk_1ms,

去掉按键去抖模块： pbdebounce p0(...)

新代码如下：

```
`timescale 1ns / 1ps
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
```

完成任务一 Top 文件的仿真

P18 页去掉 DispNum m8(...)模块，不做数码管输出

模块输入端口增加 output wire [15:0] num,

模块输出端口去掉 AN 和 SEGMENT

模块内增加 assign num={A, A1, A_IN, 4'b0000};

把 A 寄存器、A 寄存器自增/自减 1 的结果，寄存器 A 输入这 3 个数据作为仿真结果输出 num

新代码如下：

```
`timescale 1ns / 1ps
module Top(
        input clk,
        input [15:0] SW,
        //output [3:0] AN,
        //output [7:0] Segment
        output wire [15:0] num // new
        );

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

```
        wire [31:0] clk_div;

        assign num = {A, A1, A_IN, 4'b0000}; // new

        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]), .Load_out
(Load_A));
        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_sch m8(.clk(clk), .HEXS({A, A1, A_IN,
4'b0000}), .LES(4'b0), .points(4'b0), .RST(1'b0), .AN(AN), .Se
gment(SEGMENT));
endmodule
```

- 输入以下仿真代码：

  SW[15]拨到 1，SW[2]上下一次，寄存器 A 清零

  SW[15]拨到 0，SW[0]拨到 0，SW[2]上下拨动 1 次，寄存器 A 加 1。

  在此过程中验证 A1、A_IN 和 A 的结果

  SW[2]再拨动 1 次，A 寄存器再加 1

  SW[0]拨到 1，SW[2]拨动 3 次，A 寄存器累积减 3

  在上述过程中观察仿真波形里 num 总线输出结果，验证 A、A1、A_IN 的变化过程是否符合预期

```
`timescale 1ns / 1ps
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
        SW = 0;

        SW[15] = 1;
        SW[2] = 0; #50;
        SW[2] = 1; #50;

        SW[15] = 0;
        SW[0] = 0;
        SW[2] = 0; #50;
        SW[2] = 1; #50;

        SW[2] = 0; #50;
        SW[2] = 1; #50;

        SW[0] = 1;
        SW[2] = 0; #50;
        SW[2] = 1; #50;
        SW[2] = 0; #50;
        SW[2] = 1; #50;
        SW[2] = 0; #50;
        SW[2] = 1; #50;
        #50;
    end
    always begin
        clk = 1; #5;
        clk = 0; #5;
    end
endmodule
```

(h)得到波形图如下：

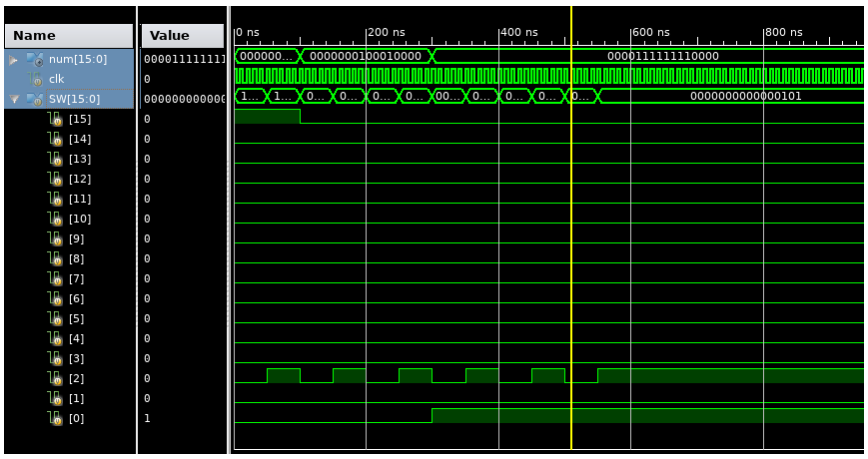

图 3.1 仿真波形图

2. **基于多路选择器总线的寄存器传输**

(a)新建源文件 RegDataPathTrans，Top Level Source 为 HDL

(b)将之前设计的 clkdiv，MyRegister4b，Load_Gen，AddSub4b，Mux4to1b4，DispNum 模块调入工程中

(c)新建 Verilog 文件 Top，右键 Set as Top Module，用 Verilog 代码方式设计，输入代码如下：

```
`timescale 1ns / 1ps
module Top(
        input clk,
        input [15:0] SW,
        output [3:0] AN,
        output [7:0] Segment
    );

        wire [3:0] Load_A, Load_B, Load_C;
        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 [31:0] clk_div;
        wire [3:0] I1;

        clkdiv m0(clk, 1'b0, clk_div);

        // Register A
        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]), .Load_out
(Load_A));
        AddSub4b
m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]), .S(A_I0));
        assign A_IN = (SW[15] == 1'b0)? A_I0: I1;

        // Register B
        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_out
(Load_B));
        AddSub4b
m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]), .S(B_I0));
```

```
        assign B_IN = (SW[15] == 1'b0)? B_I0: I1;

        // Register C
        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_out
(Load_C));
        assign C_IN = (SW[15] == 1'b0)? I1: 4'b0000;

        Mux4to1b4
m9( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT), .I3(4'b0000), .S(SW[8:
7]), .o(I1));
        DispNum_sch m10(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,
4'b0000}), .LES(4'b0), .points(4'b0), .RST(1'b0), .AN(AN), .Se
gment(Segment));
endmodule
```

3. 基于 ALU 的数据传输应用设计

(a)新建工程 MyALUTrans，Top Level Source Type 使用 HDL

(b)将之前设计的 clkdiv，MyRegister4b，Load_Gen，AddSub4b，Mux4to1b4，dispNum，myALU 模块调入项目

(c)Top 模块代码如下：

```
`timescale 1ns / 1ps
module Top(
        input clk,
        input [15:0] SW,
        output [3:0] AN,
        output [7:0] Segment
    );

        wire [3:0] 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 [31:0] clk_div;
        wire [3:0] I1, I0;
        clkdiv m0(clk, 1'b0, clk_div);

        // Register A
        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]), .Load_out
(Load_A));
        AddSub4b
m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]), .S(A_I0));
        assign A_IN = (SW[15] == 1'b0)? A_I0: I1;

        // Register B
        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_out
(Load_B));
        AddSub4b
m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]), .S(B_I0));
        assign B_IN = (SW[15] == 1'b0)? B_I0: I1;

        // Register C
        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_out
(Load_C));
        assign C_IN = (SW[15] == 1'b0)? I1: I0;

        Mux4to1b4
m9( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT), .I3(4'b0000), .s(SW[8:
7]), .o(I1));
        myALU
m10( .S(SW[6:5]), .A(A_OUT), .B(B_OUT), .C(I0), .Co(carry));
        DispNum_sch m11(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,
3'b000,
carry}), .LES(4'b0), .points(4'b0), .RST(1'b0), .AN(AN), .Segm
ent(Segment));
endmodule
```

(d)建立 Top_sim 仿真文件，根据仿真要求：

不输出 AN 和 SEGMENT，增加 assign num={A, B, C, Bus}，并作为 Top 模块的输出引脚进行观察

Bus 是 3 选 1 多路复用器的输出结果，即为总线数据

- 修改 top 代码如下：

```
`timescale 1ns / 1ps
module Top(
```

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

        wire [3:0] 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 [31:0] clk_div;
        wire [3:0] I1, I0;

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

        clkdiv m0(clk, 1'b0, clk_div);

        // Register A
        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]), .Load_out
(Load_A));
        AddSub4b
m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]), .S(A_I0));
        assign A_IN = (SW[15] == 1'b0)? A_I0: I1;

        // Register B
        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_out
(Load_B));
        AddSub4b
m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]), .S(B_I0));
        assign B_IN = (SW[15] == 1'b0)? B_I0: I1;

        // Register C
        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_out
(Load_C));
```

```
        assign C_IN = (SW[15] == 1'b0)? I1: I0;

        Mux4to1b4
m9( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT), .I3(4'b0000), .s(SW[8:
7]), .o(I1));
        myALU
m10( .S(SW[6:5]), .A(A_OUT), .B(B_OUT), .C(I0), .Co(carry));
        //DispNum_sch m11(.clk(clk), .HEXS({A_OUT, B_OUT,
C_OUT, 3'b000,
carry}), .LES(4'b0), .points(4'b0), .RST(1'b0), .AN(AN), .Segm
ent(Segment));
endmodule
```

控制 SW[15]、SW[0],拨动 SW[2],把 A 寄存器初始化到 0

控制 SW[15]、SW[0],拨动 SW[2],把 A 寄存器加到 3

类似控制方法，把 B 寄存器减到 E

SW[6:5]置为 01，控制 SW[15]和 SW[4]，把 C 寄存器初始化成 A-B

- 输入仿真激励代码如下：

```
`timescale 1ns / 1ps
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
        SW = 0;
        SW[15] = 0;
        SW[0] = 0;
        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; // add A to 3

        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;
    end

    always begin
        clk = 1; #5;
        clk = 0; #5;
    end

endmodule
```

(e)得到仿真波形图如下：

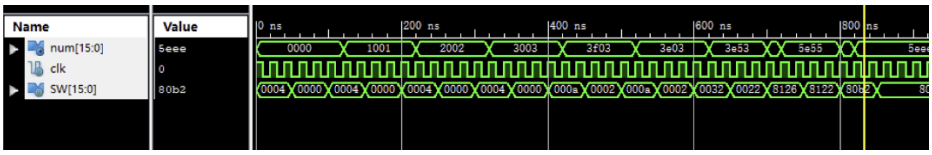

图 3.2 仿真波形图

4. **生成 bit 文件，下载到 SWORD 板上进行验证：**

(a)通过 SW[15]，SW[0]，SW[2]初始化置 0：

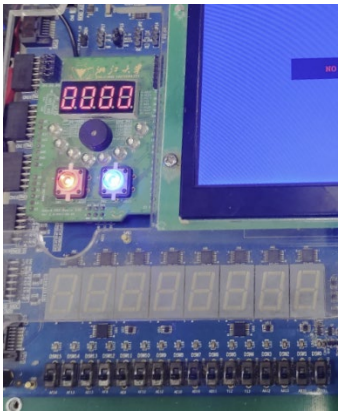

图 3.3 初始化

(b)A 加到 0x3

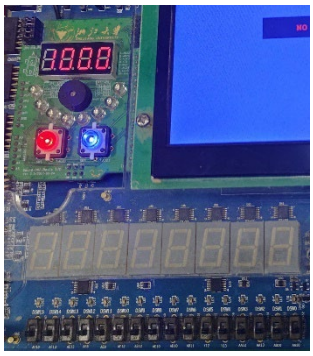

图 3.4 加 1

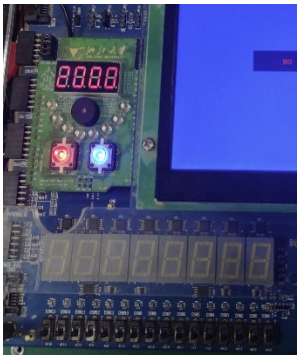

图 3.5 加 2

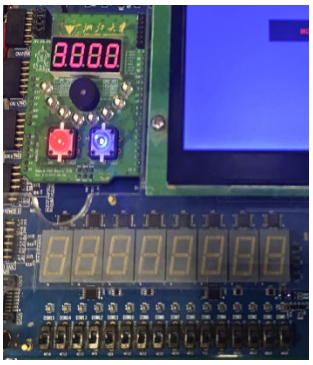

图 3.6 加 3

(c)B 减到 0xE

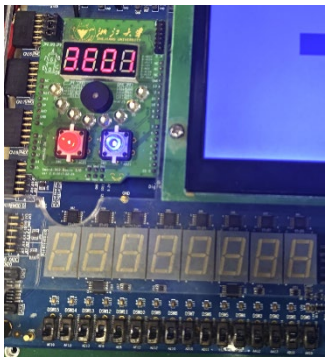

图 3.7 B 减到 E

(d)将 SW[6:5]置为 0 1，C 初始化为 A-B

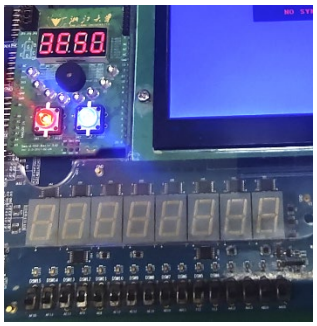

图 3.8 C = A - B

(e)控制 SW[8:7]将 C 传给 A

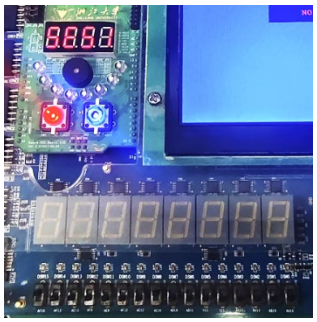

图 3.9 A <= C

(f)控制 SW[8:7]将 B 传给 C

图 3.10 C <= B

四、实验结果分析

本次实验通过仿真并使用 SWORD 板进行验证，实验结果符合预期。

在实验过程中，我们发现 Load_Gen 产生信号是需要一定时间的，即存在延迟，由于我们的 clk 是硬件层面，周期很短，因此需要下一个上升沿才能进行数据的写入。

五、讨论与心得

本次实验多亏了老师的讲解，在按键防抖的过程中需要去掉 1ms 的引脚，

因为时间太短，扫描得太快，这个地方卡了好长时间。和同学们讨论之后发现需要改变 Load_Gen 模块中的 btn_out 改为 in，否则 num 就一直为 0（这个模块改了好多次啊），最后在仿真的时候波形才正确。

感谢老师、助教以及本组同学的帮助，本次实验顺利完成！