实验报告

实验名称：ALU 设计实验

院系：计算机学院 19 级计算机科学与技术

班级：　计科 2 班

姓名：　施天予

指导老师：陈刚

一、 实验内容

1、 根据 ALU 原理图，使用 Verilog 语言定义 ALU 模块，其中输入输出端口参照实验原理，运算指令码长度为[2:0]。

2、 内置一个 32 位 num2(值为 32h’ 01)作为输入到运算器端口 A。

3、 将 sw0-sw7 输入到 num1，经过符号扩展至 32 位后，输入到运算器的端口 B。

4、 运算器支持“加、减、与、或”4 种运算，需要 3 位（8 个操作）。将 sw15-sw13 输入到 op 作为运算器的控制信号。

5、 实现 SLT 功能。

6、 将计算 32 位结果 s 显示到七段数码管（16 进制）。

7、 验证表 1 中所有功能。

8、 给出 RTL 源程序（.v 文件）。

一、 实验原理

下图给出了一个具有 N 位输入和 N 位输出的算数逻辑单元的电路符号。算术逻辑单元接收说明执行哪个功能的控制信号 F，执行对应功能后输出 N 位结果。

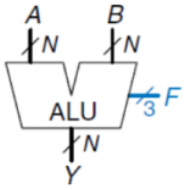


图 1

实验要求实现以下算术运算功能，其对应的指令码及功能如下:

表 1

$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\overline{A}$
001	A - B	***101***	***SLT***
010	A AND B	110	未使用
011	A OR B	111	未使用

本次实验将 ALU 输出结果通过板载七段数码管进行显示验证，原理图如下图所示：

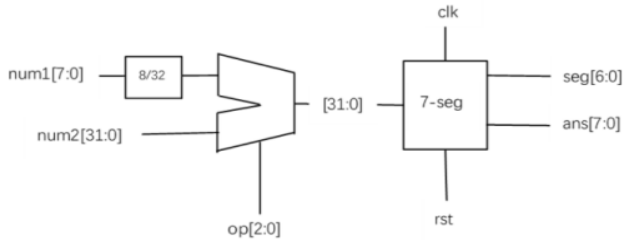


图 2

二、 实验设计

因为 sw0-sw7 输入到 num1，sw15-sw13 输入到 op，则约束文件如下：

```
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

```

分配 sw13(U1)为 op[0]， sw14(T1)为 op[1]， sw15(R2)为 op[2]

```
set_property PACKAGE_PIN U1 [get_ports {op[0]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {op[0]}]
set_property PACKAGE_PIN T1 [get_ports {op[1]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {op[1]}]
set_property PACKAGE_PIN R2 [get_ports {op[2]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {op[2]}]

```

分配 sw0(V17)为 num1[0]，sw1(V16)为 num1[1]，sw2(W16)为 num1[2]，sw3(W17)为 num1[3]， sw4(W15)为 num1[4]， sw5(V15)为 num1[5]，sw6(W14)为 num1[6]， sw7(W13)为 num1[7]

```
set_property PACKAGE_PIN V17 [get_ports {num1[0]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[0]}]
set_property PACKAGE_PIN V16 [get_ports {num1[1]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[1]}]
set_property PACKAGE_PIN W16 [get_ports {num1[2]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[2]}]
set_property PACKAGE_PIN W17 [get_ports {num1[3]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[3]}]
set_property PACKAGE_PIN W15 [get_ports {num1[4]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[4]}]
set_property PACKAGE_PIN V15 [get_ports {num1[5]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[5]}]
set_property PACKAGE_PIN W14 [get_ports {num1[6]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[6]}]
set_property PACKAGE_PIN W13 [get_ports {num1[7]}]
       set_property IOSTANDARD LVCMOS33 [get_ports {num1[7]}]

```

```
set_property PACKAGE_PIN W7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN W6 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U8 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V5 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]

set_property PACKAGE_PIN U2 [get_ports {ans[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[0]}]
set_property PACKAGE_PIN U4 [get_ports {ans[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[1]}]
set_property PACKAGE_PIN V4 [get_ports {ans[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[2]}]
set_property PACKAGE_PIN W4 [get_ports {ans[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[3]}]

set_property PACKAGE_PIN U18 [get_ports rst]
        set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

top.v

```
`timescale 1ns / 1ps

module top(
    input clk,
    input rst,
    input [2:0] op,
    input [7:0] num1,
    output [3:0] ans, //select for seg
    output [6:0] seg  //segment digital
    );
    wire [31:0] s;
    calculate U1(.num1(num1),.op(op),.result(s));

    display U2(.clk(clk),.reset(rst),.s(s),.ans(ans),.seg(seg));
endmodule

```

display.v

```
`timescale 1ns / 1ps

module display(
    input wire clk,reset,
    input wire [31:0]s,
    output wire [6:0]seg,
    output reg [3:0]ans
    );
    reg [20:0]count;
    reg [4:0]digit;
    always@(posedge clk,posedge reset)
    if(reset)
        count = 0;
    else
        count = count + 1;

    always @(posedge clk)
    case(count[20:19])
        0:begin
            ans = 4'b1110;
            digit = s[3:0];
        end

        1:begin
            ans = 4'b1101;
            digit = s[7:4];
        end


        2:begin
            ans = 4'b1011;
            digit =s[11:8];
        end

        3:begin
            ans = 4'b0111;
            digit = s[15:12];
        end
    endcase

    seg7 U4(.din(digit),.dout(seg));
endmodule
```

seg7.v

```
`timescale 1ns / 1ps

module seg7(
    input wire [3:0]din,
    output reg [6:0]dout
    );

    always@(*)
    case(din)
        5'h0:dout = 7'b000_0001;
        5'h1:dout = 7'b100_1111;
        5'h2:dout = 7'b001_0010;
        5'h3:dout = 7'b000_0110;
        5'h4:dout = 7'b100_1100;
        5'h5:dout = 7'b010_0100;
        5'h6:dout = 7'b010_0000;
        5'h7:dout = 7'b000_1111;
        5'h8:dout = 7'b000_0000;
        5'h9:dout = 7'b000_0100;
        5'ha:dout = 7'b000_1000;
        5'hb:dout = 7'b110_0000;
        5'hc:dout = 7'b011_0001;
        5'hd:dout = 7'b100_0010;
        5'he:dout = 7'b011_0000;
        5'hf:dout = 7'b011_1000;
        default:dout = 7'b111_1111;
    endcase
endmodule
```

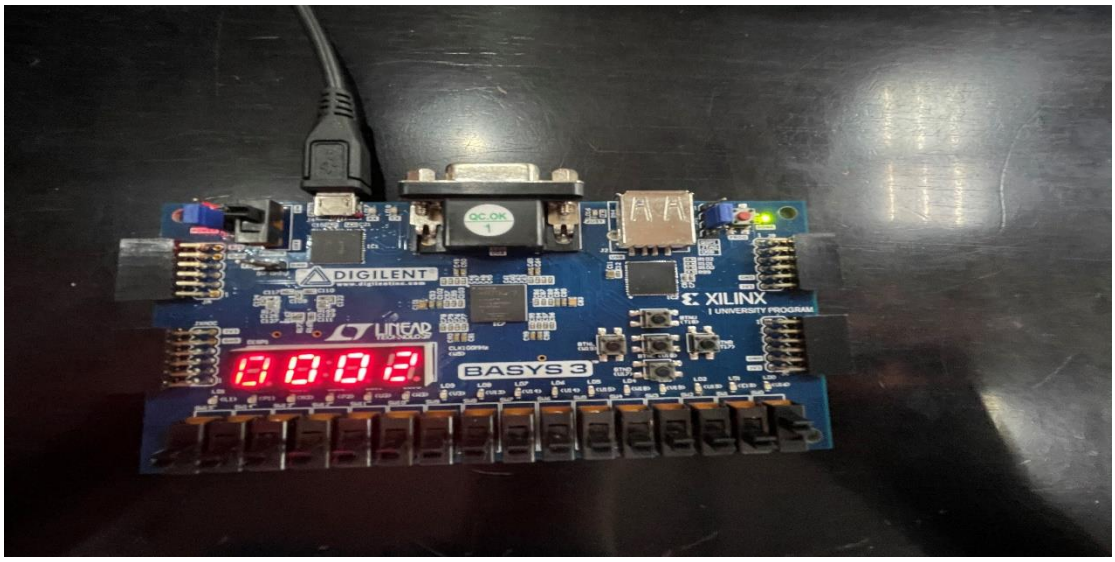
calculate.v

```
`timescale 1ns / 1ps

module calculate(
    input wire [7:0] num1,
    input wire [2:0] op,
    output reg[31:0] result
    );
    wire [31:0] num2;
    wire [31:0] Sign_extend;

    assign num2 = 32'h00000001;                    //num2值为32'h01
    assign Sign_extend = {{24{1'b0}},num1[7:0]};   //将num1扩展为32位

```

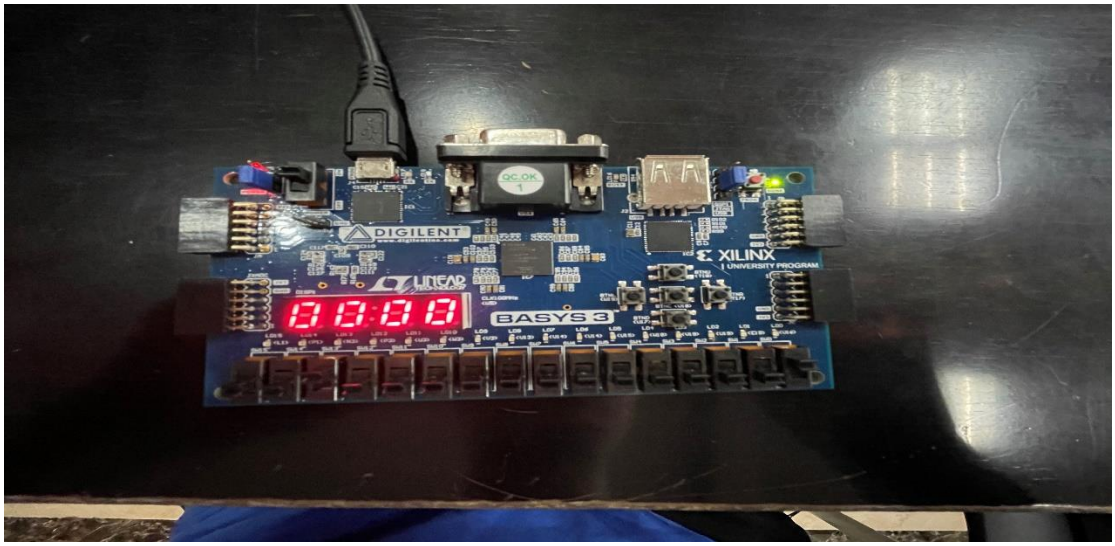
```
14      always @(op,Sign_extend,num2) begin
15          case (op)
16              3'b000: result <= Sign_extend + num2;          //op位000，加运算
17              3'b001: result <= Sign_extend - num2;          //op位001，减运算
18              3'b010: result <= Sign_extend & num2;          //op位010，与运算
19              3'b011: result <= Sign_extend | num2;          //op位011，或运算
20              3'b100: result <= ~Sign_extend;                //op位100，num1取反
21              3'b101: result <= Sign_extend < num2 ? 1:0;    //op位101，SLT运算
22              default: result <= 0;                          //否则返回0
23          endcase
24      end
25  endmodule
```

四、实验结果与分析

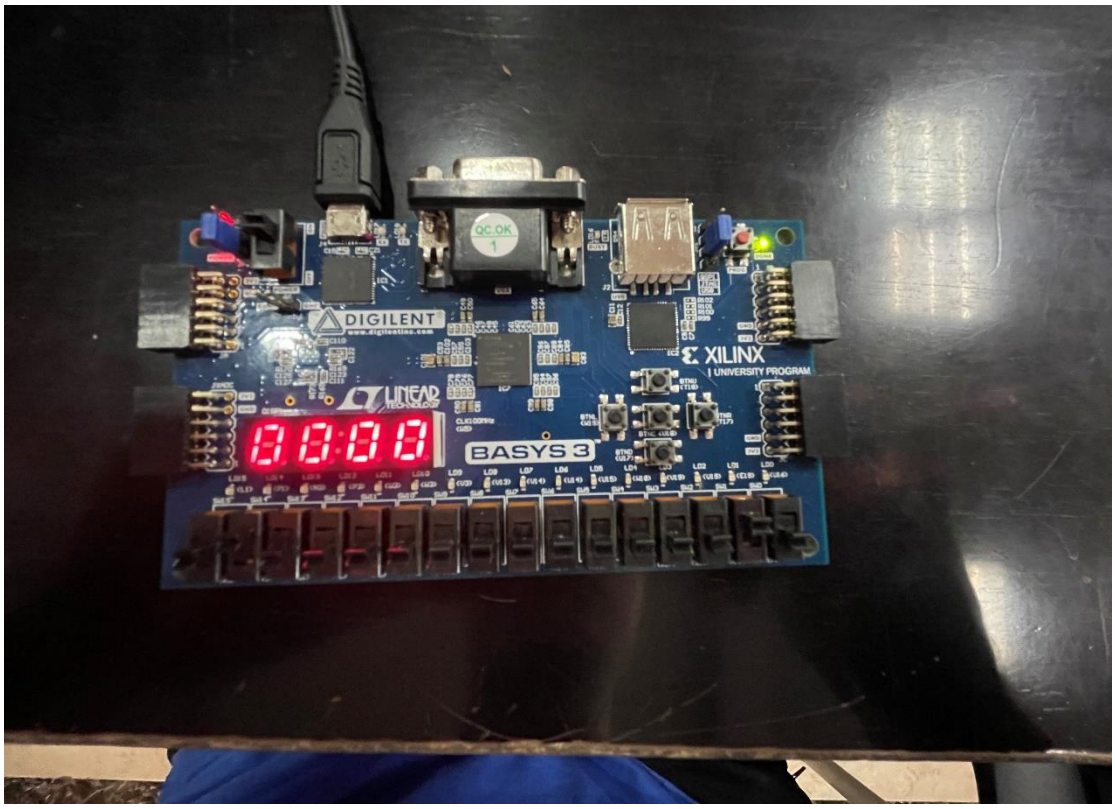
op=000 num1=1 num1+num2=2



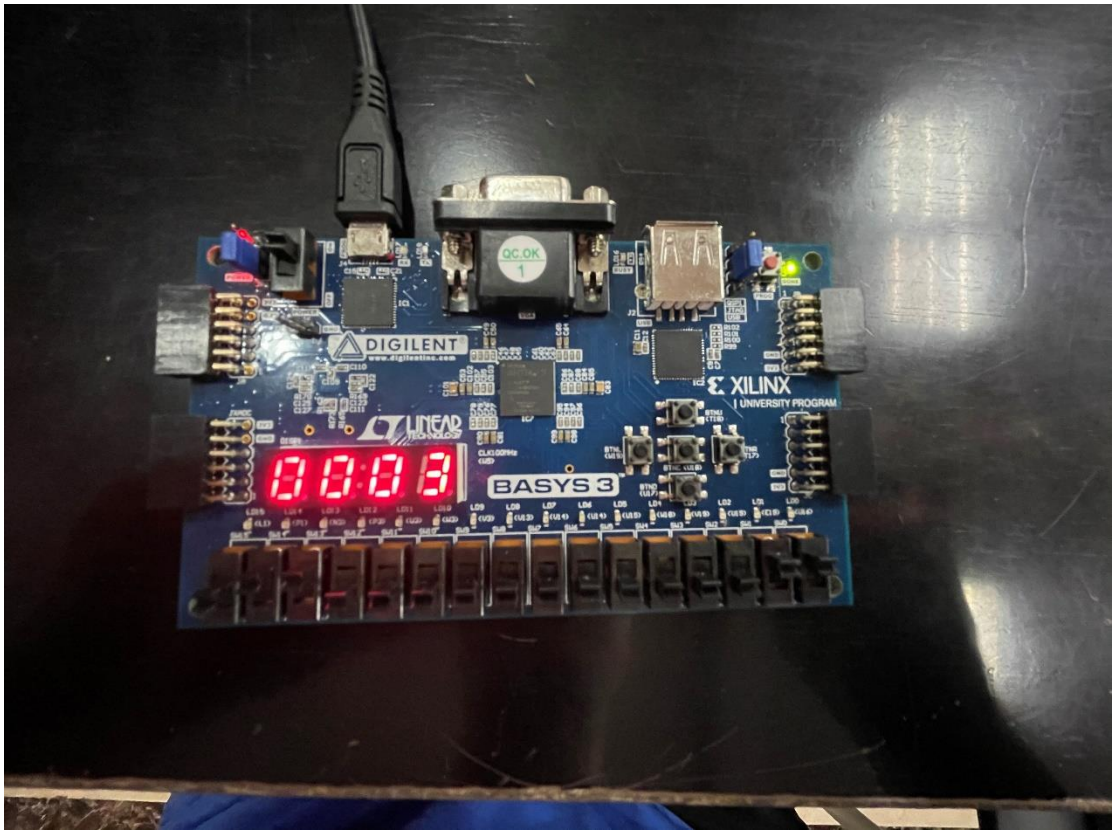
op=001 num1=1 num1-num2=0



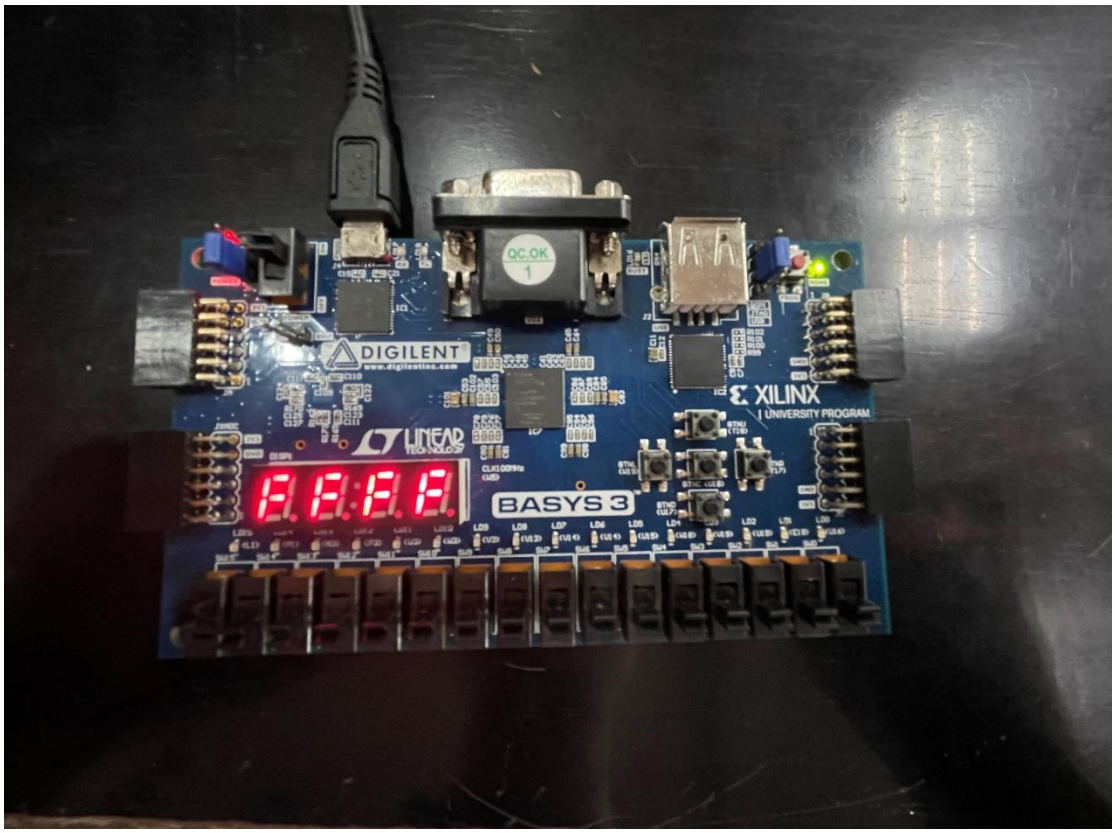
op=010 num1=0b00000010 num1&num2=0



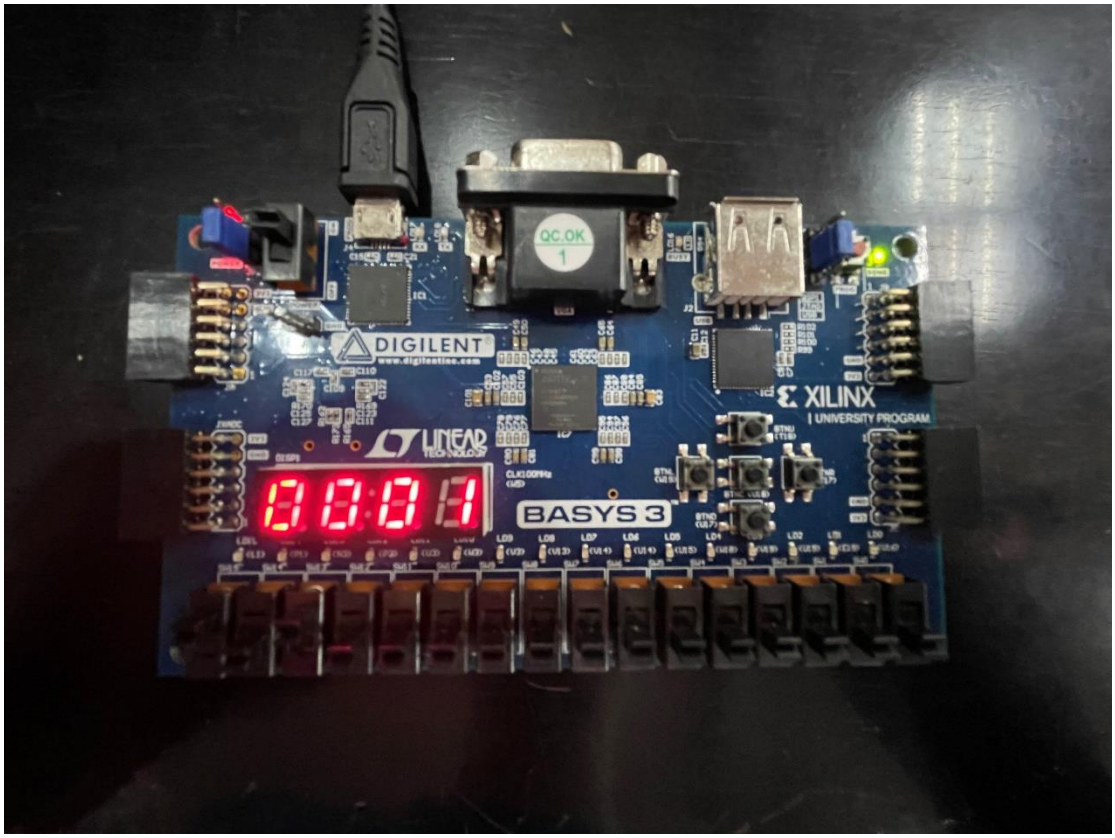
op=011 num1=0b00000011 num1|num2=0x0003



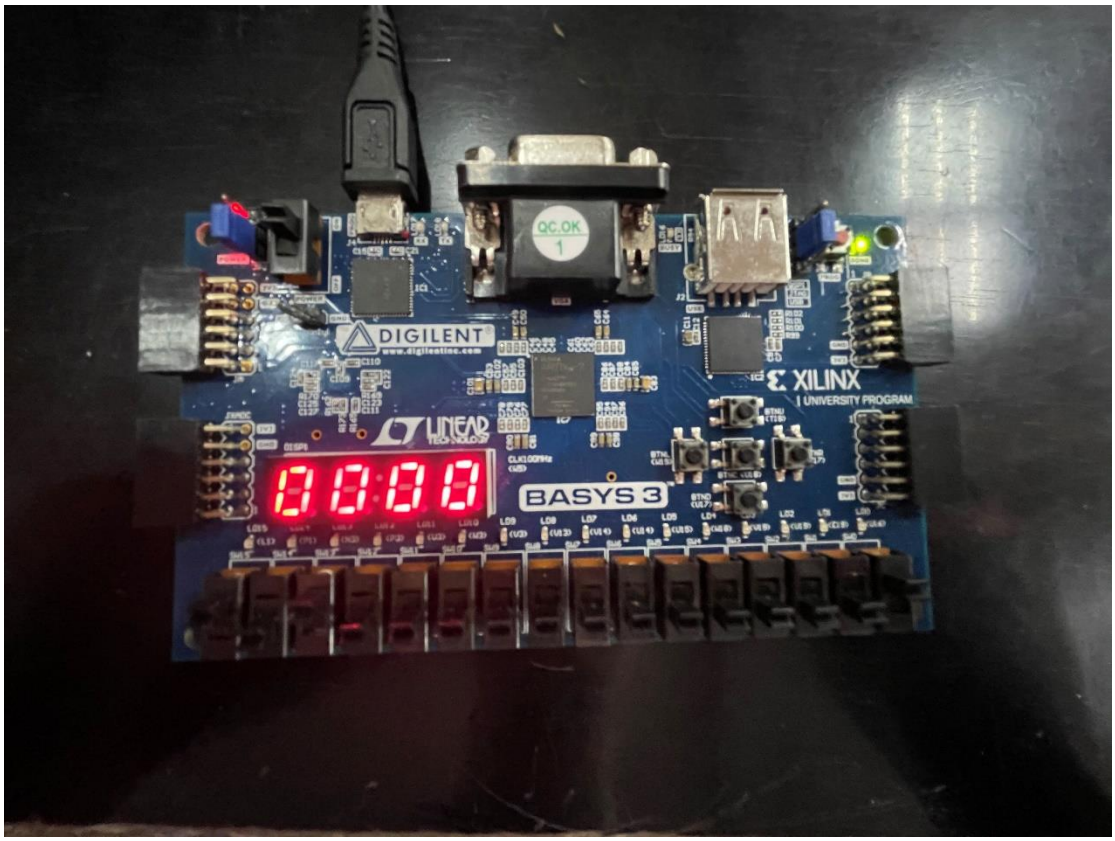
op=100 num1=0 ~num1=0xFFFF



op=101 num1=1 SLT(num1,num2)=1



op=101 num1=0 SLT(num1,num2)=0



五、实验总结

这次实验虽然我们要写的不是很多，但我依然在老师的代码和自己写的代码中学到很多。首先是在约束文件中分配管脚，其次学会了 if，case，always，assign 等语句的使用。相比于串行执行的 C/C++不同，Verilog 语言是并行执行的，所以在设计起来有所不同。对于一个 ALU 的设计，按照功能表分配寄存器十分关键。在实验中我发现，assign 赋值的信号为 wire 型，always 结构块下的信号为 reg 型； 另一个区别是，assign 用于描述组合逻辑，always 用于描述时序逻辑，仿真时只有 always 块内的输入信号产生变化，该块内描述的信号才会产生变化。

实验报告

实验名称：寄存器堆设计实验

院系：计算机学院 19 级计算机科学与技术

班级：　计科 2 班

姓名：　施天予

指导老师：陈刚

一、 实验内容

1、 根据寄存器堆原理图，使用 Verilog 语言定义 register file 模块，其实现方式可以参照实验给出的参照设计。

2、 将所涉及的 ALU 与 register file 模块连起来，形成图 3 所示的系统。

3、 根据图 4 中红色框里面的原理，完成一次模拟存储器写入，通过 MemtoReg 控制信号来确定 busW 信号是来自存储器的常熟还是 ALU 输出的结果。

4、 在$t1,$t2 初始值的基础上，模拟指令 add $t0,$t1,$t2 的过程，通过仿真验证 add 计算过程的正确性。

5、 给出 RTL 源程序（.v 文件）

二、实验原理

下图红色框里给出了一个寄存器堆电路符号，寄存器堆有 32 个 registers，能够通过 3 个 5 位地址总线 Rw，Ra，Rb 寻址到寄存器，RW 是写信号，Rw=1 时，将 busW 的数据存储到 Rw 的地址所对应的 register 里面。Rw=0 时，根据 Ra，Rb 地址读取寄存器的值，放到 busA 和 busB 上。

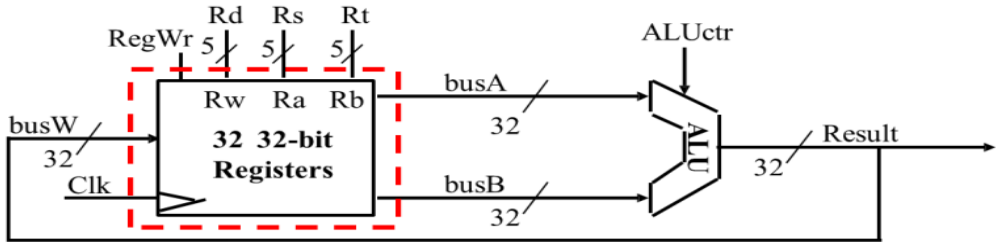
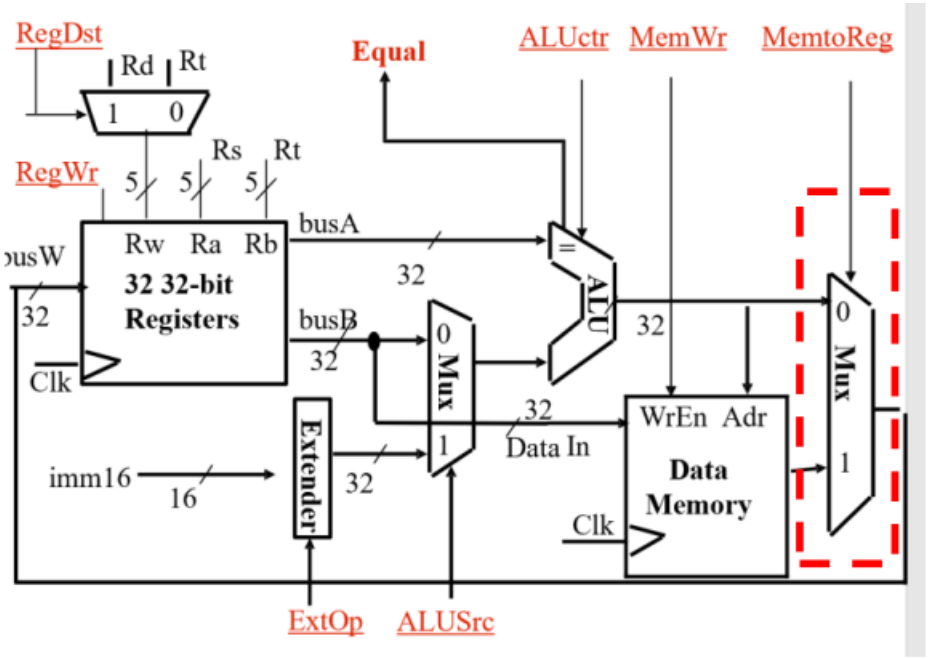


图 3



三、实验设计

因为添加了一个 memtoreg 控制信号，拨码数量不够，所以我只能把寄存器的地址改为了 4 位。

设置 sw3-sw0 为 ra_addr，sw7-sw4 为 rb_addr，sw11-sw8 为 rd_addr，sw15 为 Rw 控制读/写，s14 为 memtoreg 控制从 ALU 输出还是存储器的常数写入寄存器。

约束文件如下：

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
# Addr
set_property PACKAGE_PIN V17 [get_ports {ra_addr[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {ra_addr[0]}]
set_property PACKAGE_PIN V16 [get_ports {ra_addr[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {ra_addr[1]}]
set_property PACKAGE_PIN W16 [get_ports {ra_addr[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {ra_addr[2]}]
set_property PACKAGE_PIN W17 [get_ports {ra_addr[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {ra_addr[3]}]
```

```
set_property PACKAGE_PIN W15 [get_ports {rb_addr[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rb_addr[0]}]
set_property PACKAGE_PIN V15 [get_ports {rb_addr[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rb_addr[1]}]
set_property PACKAGE_PIN W14 [get_ports {rb_addr[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rb_addr[2]}]
set_property PACKAGE_PIN W13 [get_ports {rb_addr[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rb_addr[3]}]
set_property PACKAGE_PIN V2 [get_ports {rd_addr[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rd_addr[0]}]
set_property PACKAGE_PIN T3 [get_ports {rd_addr[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rd_addr[1]}]
set_property PACKAGE_PIN T2 [get_ports {rd_addr[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rd_addr[2]}]
```

```
set_property PACKAGE_PIN R3 [get_ports {rd_addr[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rd_addr[3]}]
# sel
set_property PACKAGE_PIN R2 [get_ports {sel}]
                set_property IOSTANDARD LVCMOS33 [get_ports {sel}]
# memtoreg
set_property PACKAGE_PIN T1 [get_ports {memtoreg}]
                set_property IOSTANDARD LVCMOS33 [get_ports {memtoreg}]
# seg
set_property PACKAGE_PIN W7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN W6 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
```

```
set_property PACKAGE_PIN U8 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V5 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
```

```
# ans
set_property PACKAGE_PIN U2 [get_ports {ans[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[0]}]
set_property PACKAGE_PIN U4 [get_ports {ans[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[1]}]
set_property PACKAGE_PIN V4 [get_ports {ans[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[2]}]
set_property PACKAGE_PIN W4 [get_ports {ans[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[3]}]
# Buttons
set_property PACKAGE_PIN U18 [get_ports rst]
        set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

top.v

```
`timescale 1ns / 1ps

module top(
    input clk,
    input rst,
    input [3:0] ra_addr,
    input [3:0] rb_addr,
    input [3:0] rd_addr,
    input sel,//选择读或者写
    input memtoreg,//选择写存储器的常数还是ALU的输出
    output [3:0] ans, //select for seg
    output [6:0] seg  //segment digital
    );
    wire [31:0] s;
    wire [31:0] mem0;
    wire [31:0] ra_data;
    wire [31:0] rb_data;
    wire [31:0] rd_data;
    wire we;
    wire [2:0] op;

    assign mem0=32'h00000001;//存储器的常数设为1
    assign we=sel;
    assign op=0;//默认为加法

    regfile U4(
        .clk(clk),
        .raddr1(ra_addr),
        .rdata1(ra_data),
        .raddr2(rb_addr),
        .rdata2(rb_data),
        .we(we),
        .waddr(rd_addr),
        .wdata(rd_data)
    );
    calculate U1(.num1(ra_data),.num2(rb_data),.op(op),.result(s));
    mux2_32b U3(.in0(s),.in1(mem0),.sel(memtoreg),.out(rd_data));
    display U2(.clk(clk),.reset(rst),.s(s),.ans(ans),.seg(seg));
endmodule
```

mux2_32b.v

```
module mux2_32b(
  input  [31:0] in0, in1,
  input  sel,
  output [31:0] out
);

assign out = ({32{sel==1'b0}} & in0)//如果memtoreg是0，写入ALU的输出
           | ({32{sel==1'b1}} & in1);//如果memtoreg是1，写入存储器的常数1

endmodule
```

regfile.v

```v
module regfile(
    input         clk,
    // READ PORT 1
    input  [ 3:0] raddr1,
    output [31:0] rdata1,
    // READ PORT 2
    input  [ 3:0] raddr2,
    output [31:0] rdata2,
    // WRITE PORT
    input         we,       //write enable, HIGH valid
    input  [ 3:0] waddr,
    input  [31:0] wdata
);
reg [31:0] rf[31:0];

//WRITE
always @(posedge clk) begin
    if (we) rf[waddr]<= wdata;
end
//READ OUT 1
assign rdata1 = (raddr1==4'b0) ? 32'b0 : rf[raddr1];
//READ OUT 2
assign rdata2 = (raddr2==4'b0) ? 32'b0 : rf[raddr2];

endmodule
```

calculate.v

```v
`timescale 1ns / 1ps

module calculate(
    input wire [31:0] num1,
    input wire [31:0] num2,
    input wire [2:0] op,
    output reg[31:0] result
    );

    always @(op,num1,num2) begin
        case (op)
            3'b000: result <= num1 + num2;          //op位000，加运算
            3'b001: result <= num1 - num2;          //op位001，减运算
            3'b010: result <= num1 & num2;          //op位010，与运算
            3'b011: result <= num1 | num2;          //op位011，或运算
            3'b100: result <= ~num1;                //op位100，num1取反
            3'b101: result <= num1 < num2 ? 1:0;    //op位101，SLT运算
            default: result <= 0;                           //否则返回0
        endcase
    end
endmodule
```

display.v

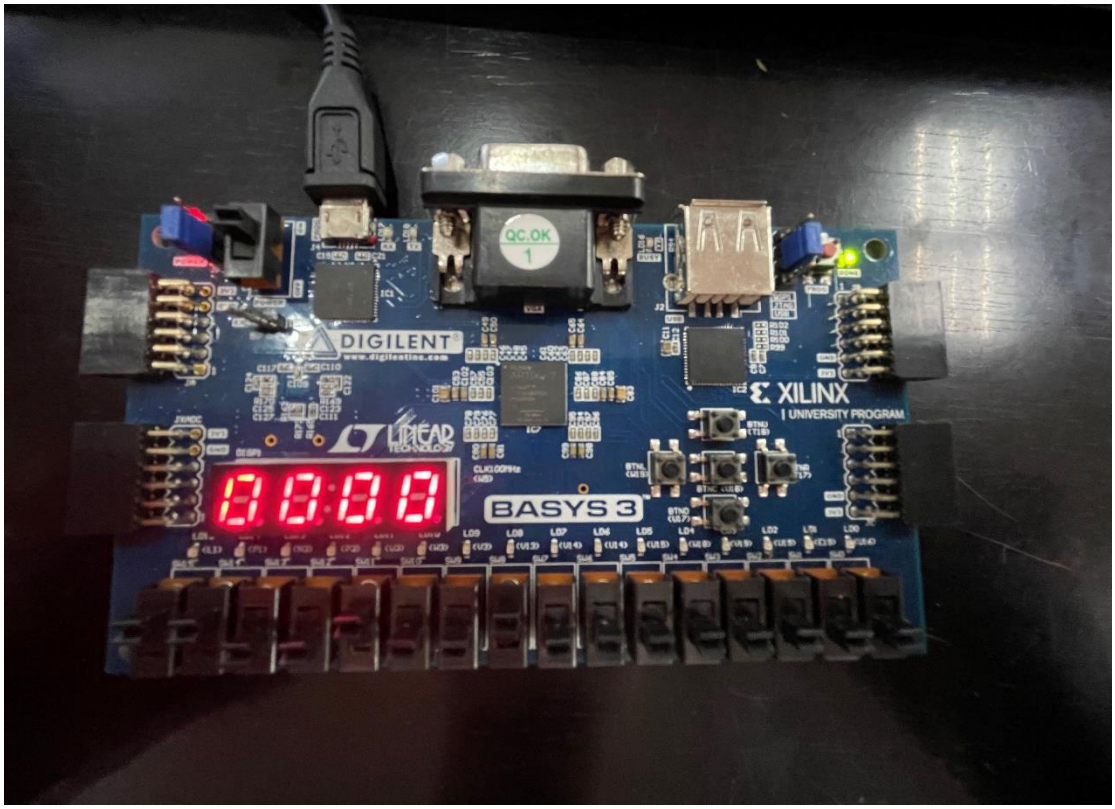
```
`timescale 1ns / 1ps

module display(
    input wire clk,reset,
    input wire [31:0]s,
    output wire [6:0]seg,
    output reg [3:0]ans
    );
    reg [20:0]count;
    reg [4:0]digit;
    always@(posedge clk,posedge reset)
    if(reset)
        count = 0;
    else
        count = count + 1;

    always @(posedge clk)
    case(count[20:19])
        0:begin
            ans = 4'b1110;
            digit = s[3:0];
        end

        1:begin
            ans = 4'b1101;
            digit = s[7:4];
        end

        2:begin
            ans = 4'b1011;
            digit =s[11:8];
        end

        3:begin
            ans = 4'b0111;
            digit = s[15:12];
        end
    endcase

    seg7 U4(.din(digit),.dout(seg));
endmodule
```

seg7.v

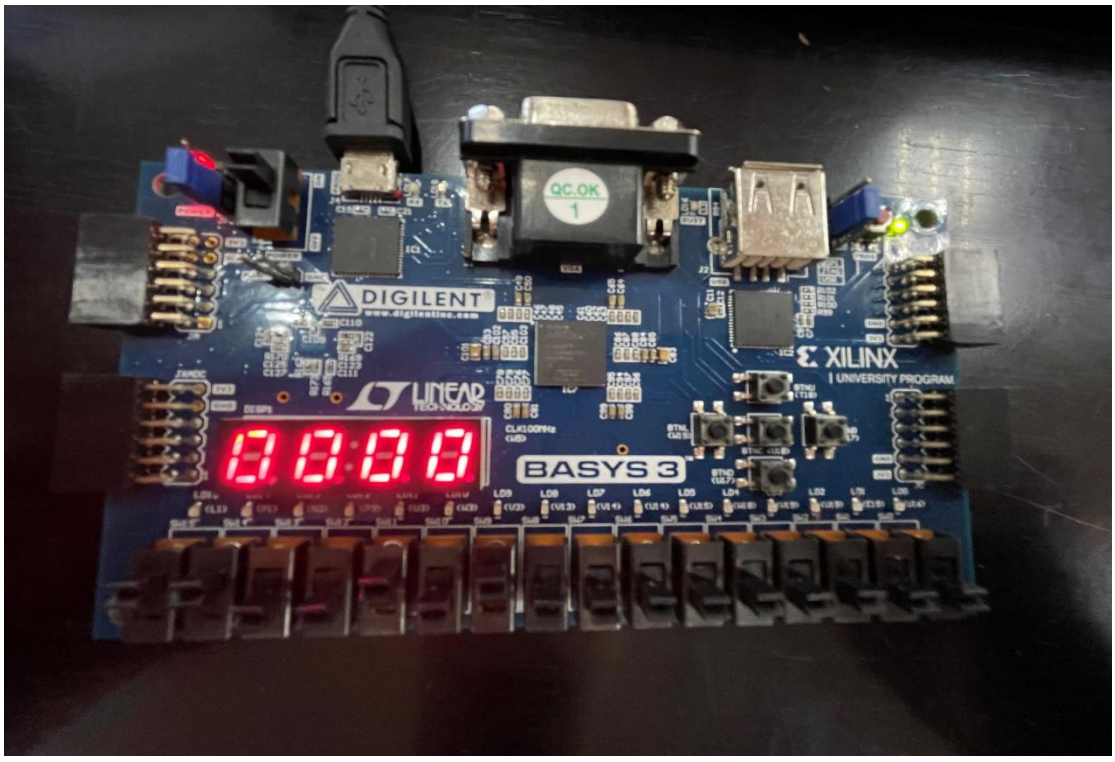
```
`timescale 1ns / 1ps

module seg7(
    input wire [4:0]din,
    output reg [6:0]dout
    );

    always@(*)
    case(din)
        5'h0:dout = 7'b000_0001;
        5'h1:dout = 7'b100_1111;
        5'h2:dout = 7'b001_0010;
        5'h3:dout = 7'b000_0110;
        5'h4:dout = 7'b100_1100;

        5'h5:dout = 7'b010_0100;
        5'h6:dout = 7'b010_0000;
        5'h7:dout = 7'b000_1111;
        5'h8:dout = 7'b000_0000;
        5'h9:dout = 7'b000_0100;
        5'ha:dout = 7'b000_1000;
        5'hb:dout = 7'b110_0000;
        5'hc:dout = 7'b011_0001;
        5'hd:dout = 7'b100_0010;
        5'he:dout = 7'b011_0000;
        5'hf:dout = 7'b011_1000;
        default:dout = 7'b111_1111;
    endcase
endmodule
```

四、实验结果与分析

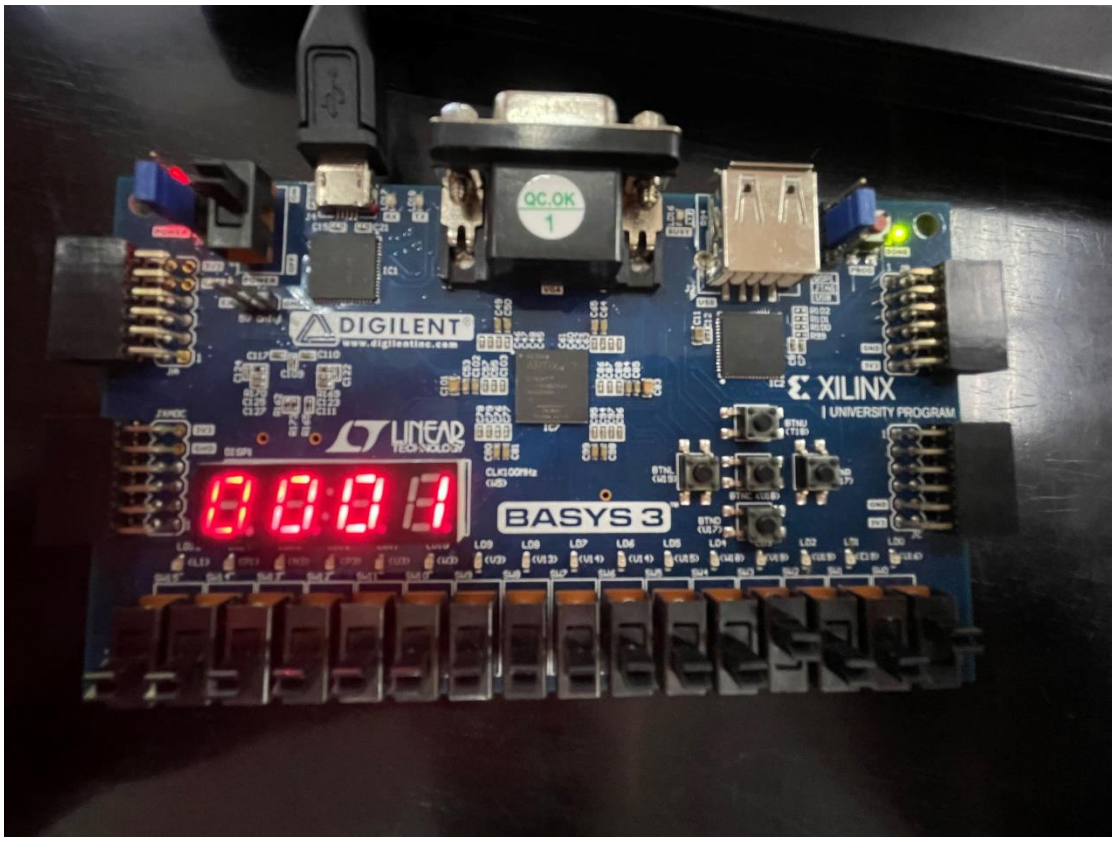
sel=1 写模式，memtoreg=1 选择存储器常数 1，写入寄存器$t1



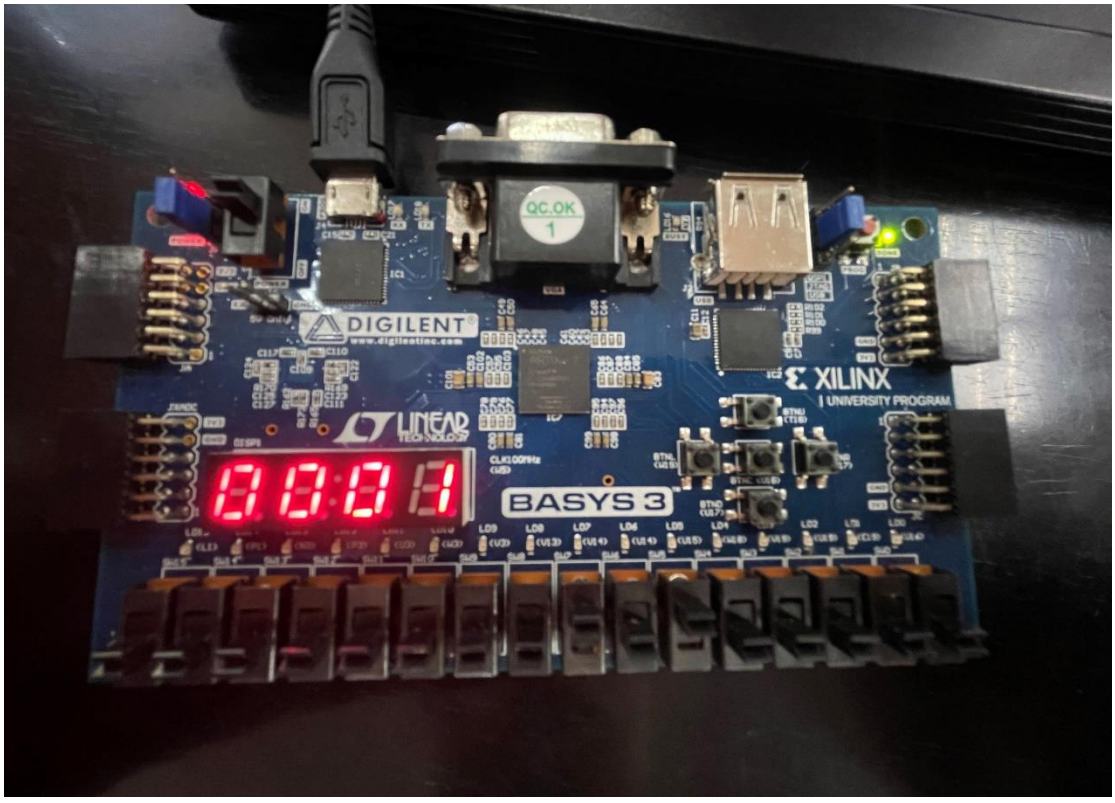
sel=1 写模式，memtoreg=1 选择存储器常数 1，写入寄存器$t2



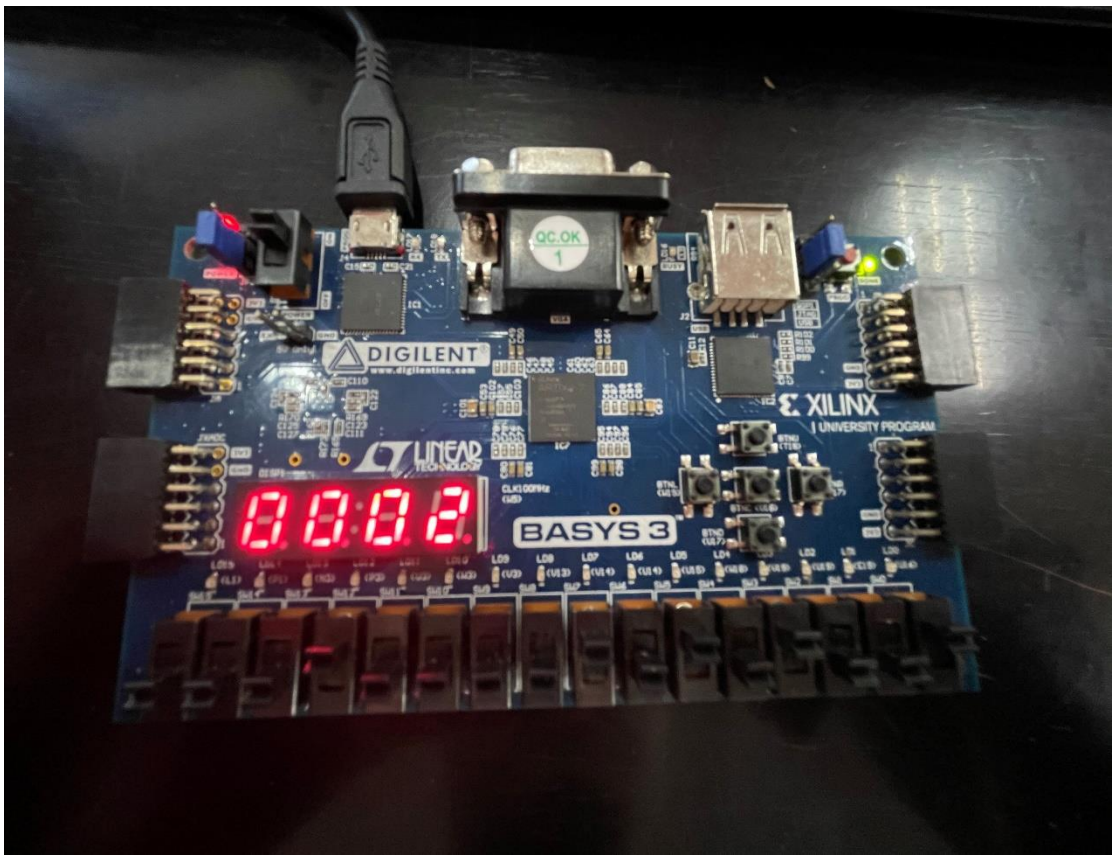
sel=0 读模式，发现寄存器$t1=1 结果正确



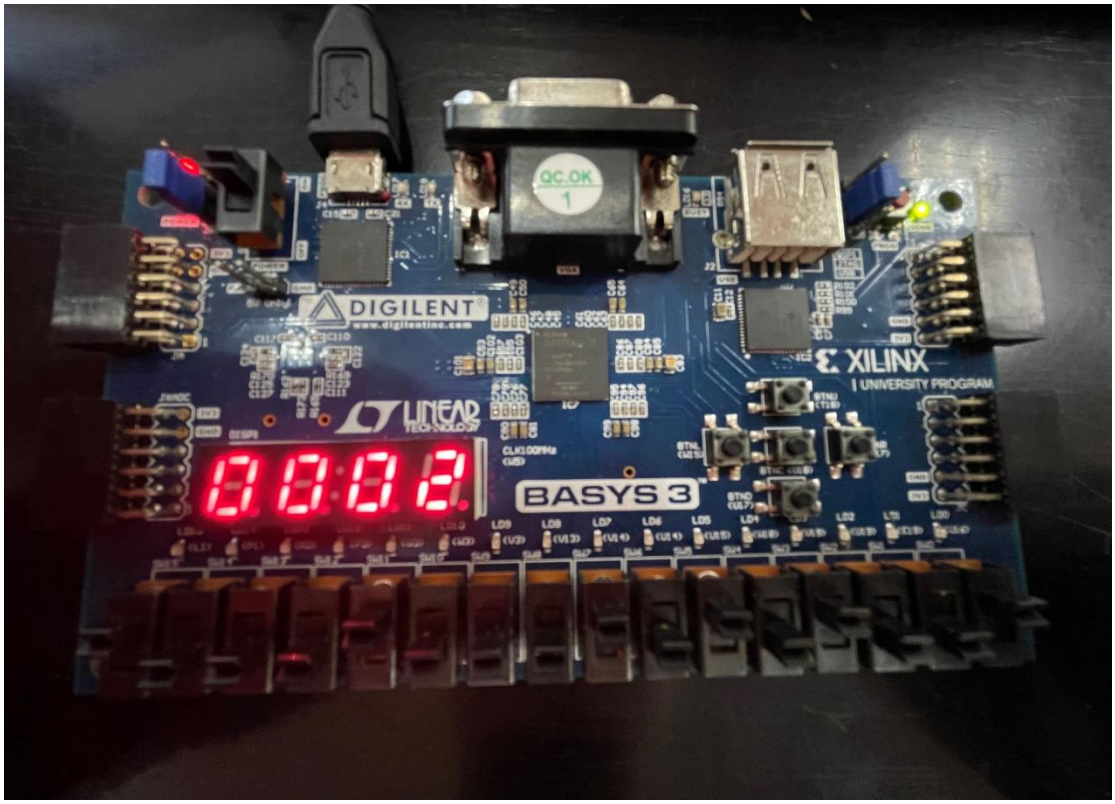
sel=0 读模式，发现寄存器$t2=1 结果正确



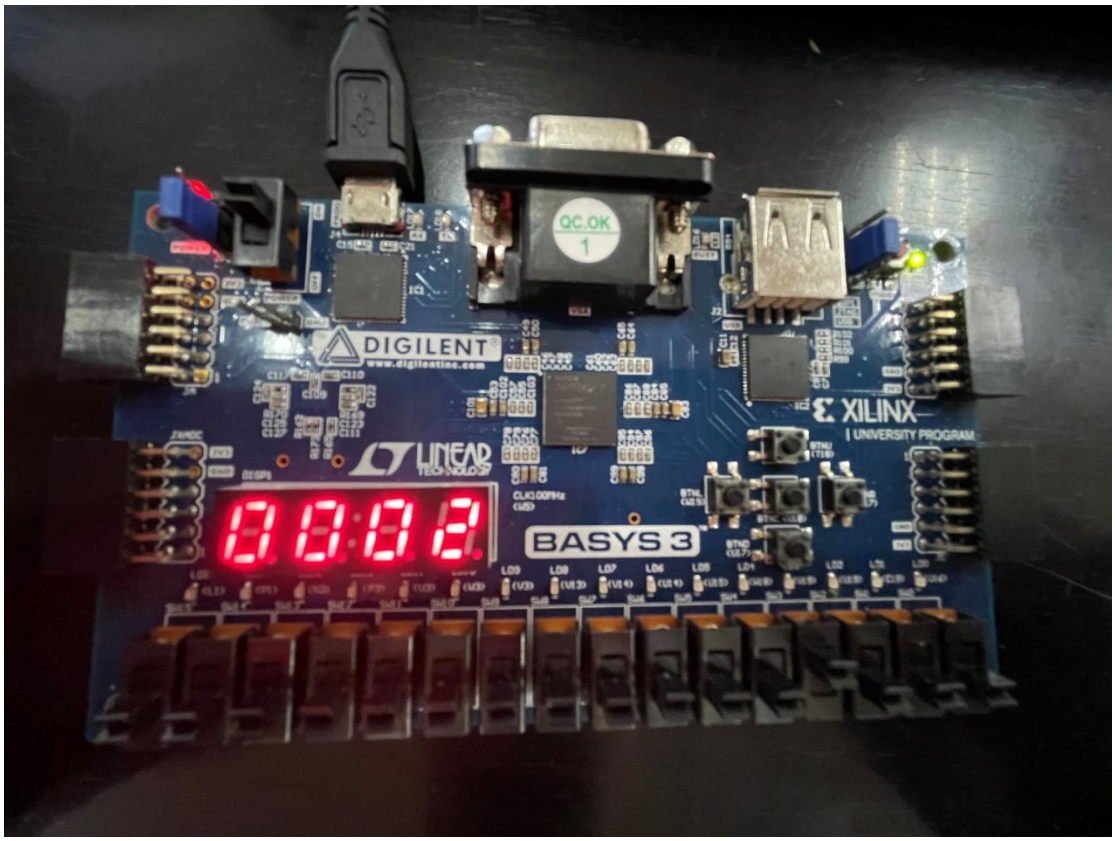
sel=0 读模式，ALU 计算结果$t1+$t2=2



sel=1 写模式，memtoreg=0 选择 ALU 计算结果，写入寄存器$t0



sel=0 读模式，发现寄存器$t0=2 结果正确



五、实验总结

寄存器的实验相比 ALU 的设计复杂了许多，我一开始有些地方没想明白，浪费了许多时间。比如我一开始把 sel 和 memtoreg 搞混了，共用了一个拨码，于是就得不到正确的结果；我还直接把 32'b1 的赋值写入寄存器模块里想直接用，导致该模块出现了异常。通过这次实验，我学会了寄存器的读写设计，也学会了 ALU 的设计。对于多路选择器的使用，我也学会了两种写法，受益匪浅。虽然实验过程中求而不得的感受是十分痛苦的，但完成实验时的喜悦也让我回味无穷。