第五次实验报告

ALU 与寄存器堆设计实验

22336313 郑鸿鑫 6 班

一． 实验目的

1. 了解算术逻辑单元 ALU 的原理
2. 熟悉并运用 verilog 语言设计 ALU
3. 学习寄存器堆的数据传送与读写工作原理，掌握寄存器堆读写的设计方法

二． 实验内容

本实验实现了两个基础模块 ALU 模块和 reg file 寄存器堆模块，ALU 实验：

ALU 有一个三位的控制信号 F。

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

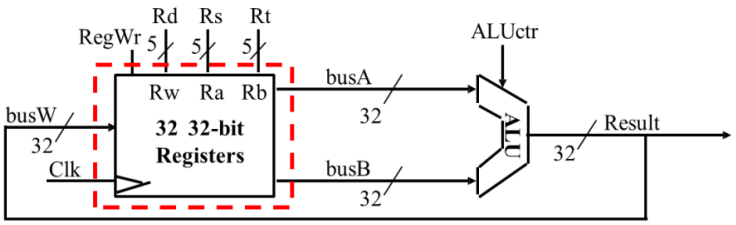
$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\bar{A}$
001	A - B	***101***	***SLT***
010	A AND B	110	未使用
011	A OR B	111	未使用

F 为 0 到 5 时对应不同的算术运算功能，6，7 为闲置无功能。

ALU 中内置一个 32 位二进制数（默认为 32'b01），作为 num2 传输到 ALU 端口 A。将 SW4~SW11 输入 num1，经过符号扩展后，输入到 ALU 的端口 B，运算结束后，ALU 会将结果输出。

寄存器堆实验：

寄存器堆的原理图如下：



寄存器堆接收两个读寄存器的编号和一个写寄存器的编号，接收时钟信号，写进写寄存器的值，输出两个读寄存器中存的值，然后输出到 ALU 计算得到结果，若 ALUctr 为 1，则将计算结果写回寄存器堆，否则将 32'b00 写回到寄存器堆。

三． 实验过程

为两个实验分别建立两个 vivado 工程，编写模块代码，然后进行模块综合，编写仿真文件测试，待结果无误后，连接 basys3 开发板运行程序。

ALU 实验代码：（解释已写在注释中）

```
module calculate(
    input wire [7:0] num1,//接收num1
    input wire [2:0] op,//接收ALU的控制信号
    output wire [31:0] result//作为计算结果输出
    );
    wire [31:0] num2;
    assign num2 = 32'h01;//内置32位num2，默认为0
    wire [31:0] Sign_extend;
    assign Sign_extend = {24'b0,num1};//对num1进行符号扩展，约定为无符号数
    reg [31:0] s;//存储计算结果
    always@(*)begin
        case(op)
            0:begin
                s = num2 + Sign_extend;
                                //控制信号为000，执行加法A + B
            end
            1:begin
                s = num2 - Sign_extend;
                                //控制信号为001，执行减法A - B
```

```
            end
            2:begin
                s = num2 & Sign_extend;
                            //控制信号为010，执行按位与运算A & B
            end
            3:begin
                s = num2 | Sign_extend;
                            //控制信号为011，执行按位或运算A | B
            end
            4:begin
                s = ~num2;
                            //控制信号为100，执行按位求反运算~A
            end
            5:begin
                if(num1 < num2)  s = 32'h01;
                else  s = 32'h00;
                 //控制信号为101，执行STL 运算 小于则置位 否则置0
            end
            default: s = 32'h00;
        endcase
    end
    assign result = s;//将结果赋值到输出上
endmodule
```

7 段数码管显示代码：

```
module display(
    input wire clk,reset,
    input wire [31:0]s,
    output wire [6:0]seg,
    output reg [3:0]ans
    );
    //add your own code here
    reg [20:0]count;
    reg [3:0] digit;
    always@(posedge clk,posedge reset)
    if(reset)
        count = 0;
    else
        count = count + 1;
    always @(posedge clk)
    begin
        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
    end
    seg7 U4(.din(digit),.dout(seg));
endmodule
```

模块综合代码：

```
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
```

regfile 寄存器堆实验代码：

```
module regfile(
    input clk,//接收时钟信号
    input [4:0] raddr1,//读寄存器 1 的编号
    output [31:0] rdata1,//读寄存器 1 的输出
    input [4:0]raddr2,//读寄存器 2 的编号
    output [31:0] rdata2,//读寄存器 2 的输出
    input wren,//寄存器堆的写使能信号
    input [4:0] waddr,//写寄存器的编号
    input [31:0] wdata//写寄存器的输出
```

```
    );
    reg[31:0] rf[31:0];//寄存器堆
    initial begin
        rf[1] = 32'b01;//将 1 号初始化为 1
        rf[2] = 32'b10;//将 2 号初始化为 2
    end
    always@(posedge clk)begin
        if(wren) rf[waddr] <= wdata;//写入写寄存器
    end
    assign rdata1 = (raddr1 == 5'b0) ? 32'b0 : rf[raddr1];
    assign rdata2 = (raddr2 == 5'b0) ? 32'b0 : rf[raddr2];
    //将读到的值输出
endmodule
```

alu 代码：

```
module alu(
    input wire [31:0]data1,
    input wire [31:0]data2,
    input aluctr,//alu 控制信号
    output wire [31:0] result
    );
    assign result = (aluctr) ? (data1 + data2) : 32'b0;
    //控制信号为 1，输出两个输入的和，否则输出 0
endmodule
```

寄存器堆仿真测试代码：

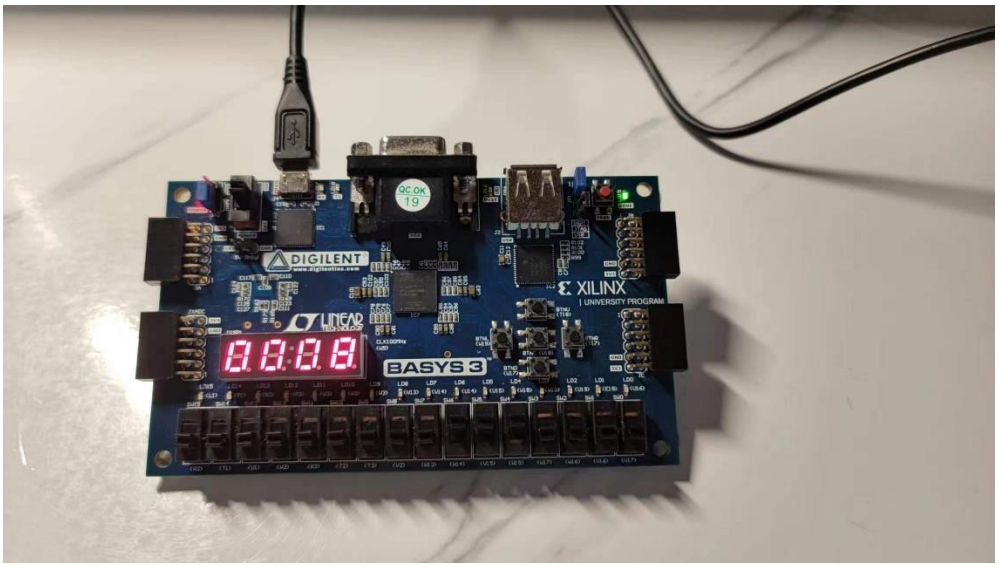
```
module top;
    reg clk;
    reg [4:0] raddr1;
    reg [4:0] raddr2;
    reg wren;
    reg aluctr;
    reg [4:0] waddr;
    reg [31:0] wdata;
    wire [31:0] rdata1;
    wire [31:0] rdata2;
    wire [31:0] result;//输入输出
    regfile r(
        .clk(clk),
        .raddr1(raddr1),
        .raddr2(raddr2),
        .rdata1(rdata1),
```

```
        .rdata2(rdata2),
        .wren(wren),
        .waddr(waddr),
        .wdata(wdata)
    );//实例化寄存器堆
    alu a(
        .data1(rdata1),
        .data2(rdata2),
        .aluctr(aluctr),
        .result(result)
    );//实例化 alu
    initial begin
        raddr1 = 1;
        raddr2 = 2;
        waddr = 3;
        wdata = 32'b1;
    end//初始化赋值，将寄存器 1 和寄存器 2 作为读寄存器
    parameter PERIOD=20;
    always begin
    clk=0;
    #(PERIOD/2);
    clk=1;
    #(PERIOD/2);
    end//时钟信号翻转
    always begin
    wren = 0;
    #(PERIOD);
    wren = 1;wdata = result;waddr = waddr + 1;
    #(PERIOD);//写使能信号翻转，每次成功写入后，将写寄存器的编号换
成下一个
    end
    always begin
    aluctr = 0;
    #(PERIOD*2);
    aluctr = 1;
    #(PERIOD*2);
    end//aluctr 控制信号翻转
endmodule
```

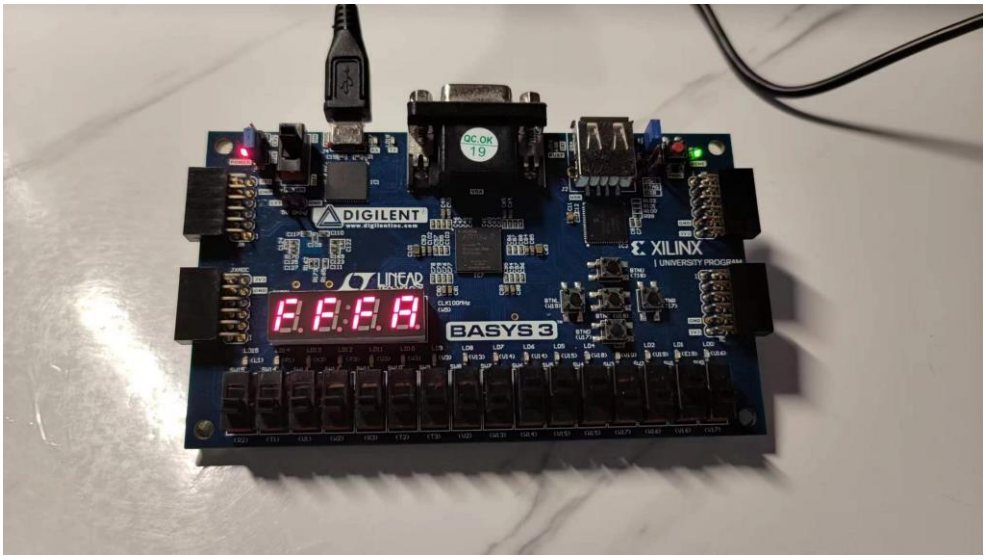
四． 实验结果分析

Alu 实验上板结果：（num1 输入为 SW4~SW11，控制信号 F 输入为 SW1~SW3）

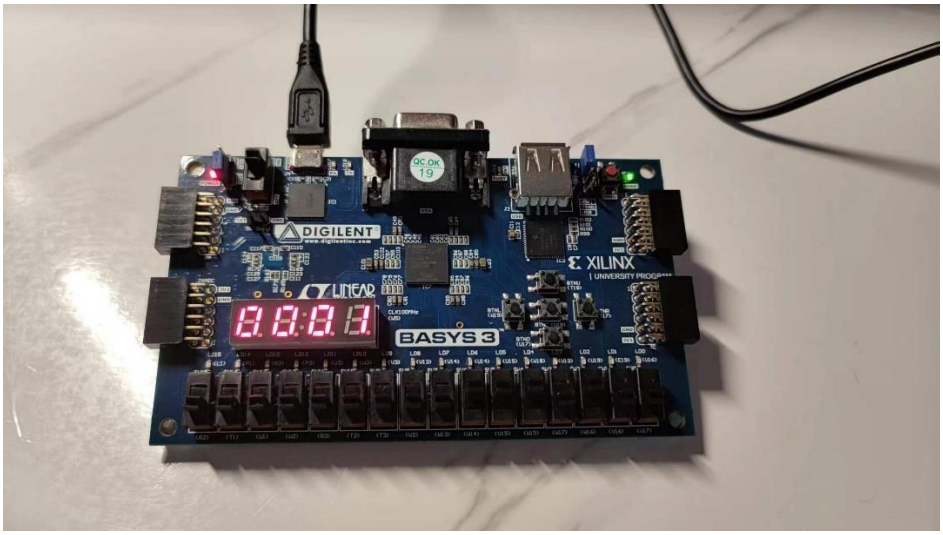
加法示例：（1 + 7 = 8）



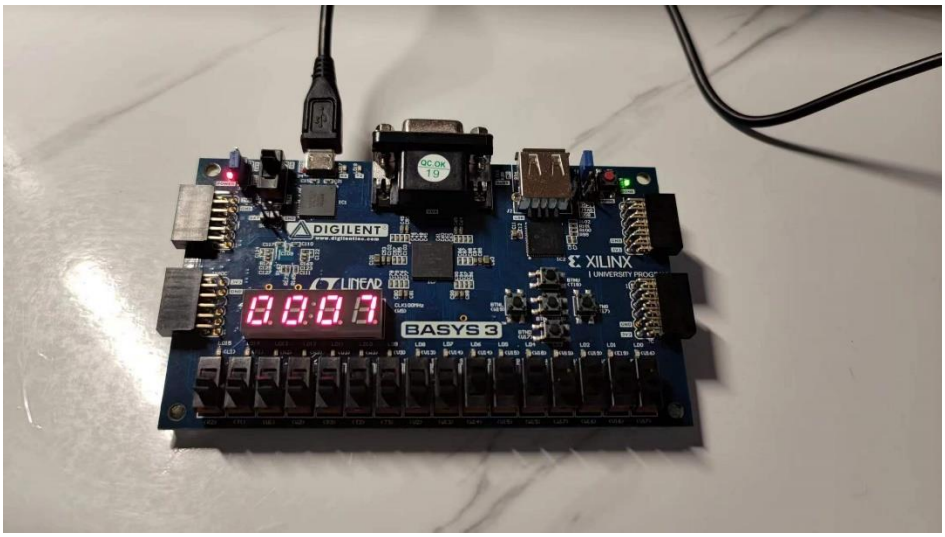
减法示例：（1 - 7 = -6）（图中 FFFA 为补码）



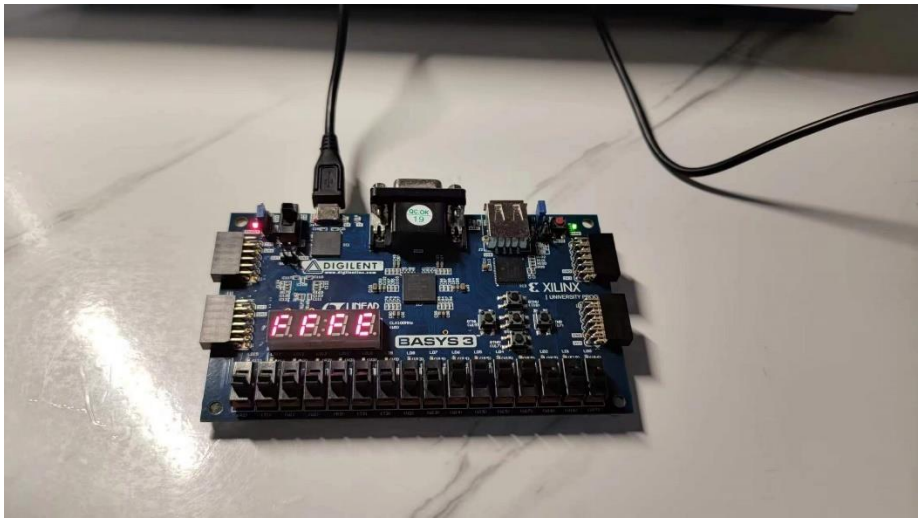
按位与运算示例：（1 & 7 = 1）



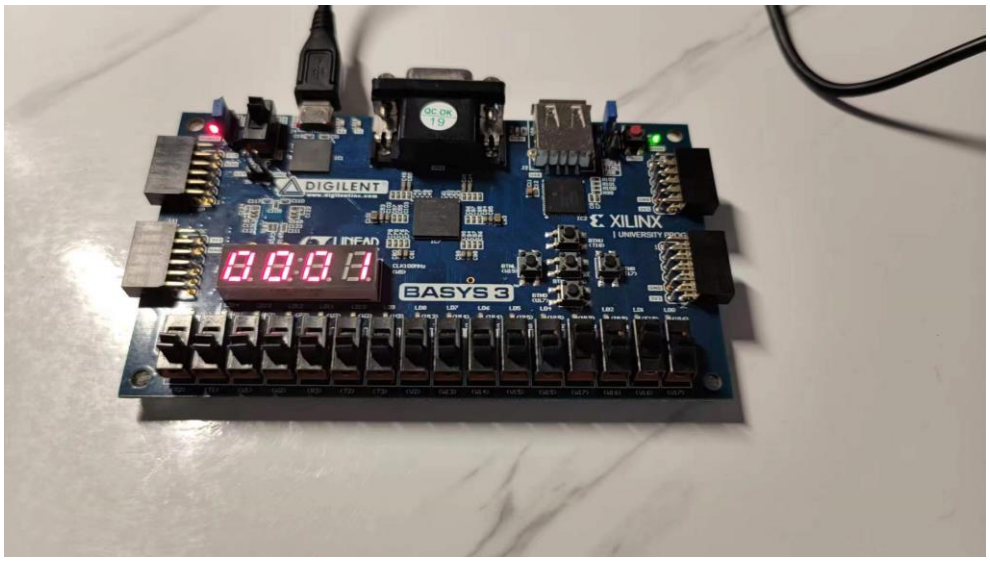
按位或运算示例：（1 | 7 = 7）



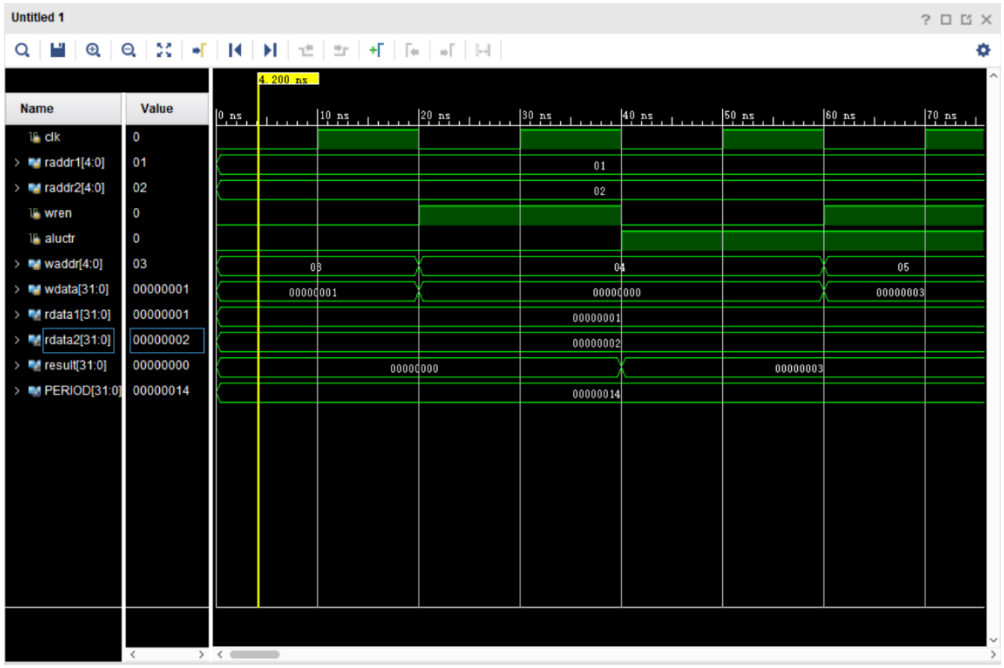
对 A 按位求反示例：（~1 = FFFE）



STL 功能示例（0 < 1 返回 1）：



寄存器堆实验仿真截图：



仿真测试代码中，我们让两个读寄存器位 1 号寄存器和 2 号寄存器（并将值初始化为 1 和 2），然后每次写回寄存器堆后，让写寄存器的编号变成下一个寄存器的编号，由于 wdata 初始化为 1，所以一开始为 1，接下来一个周期，因为写使能信号为 1，所以写入后 waddr+1 变为 4，由于 aluctr 仍为 0，所以写入 0，下一个周期当 aluctr 变为 1 后，wdata 为 1 号 2 号寄存器之和即 3，写入 5 号寄存

器，所以两个模块功能符合预期。

五． 实验总结

实验结果符合预期，ALU 模块和寄存器堆功能符合预期。

实验中遇到的问题：

assign 语句不能在 always 语句中赋值，因为 always 语句是时序逻辑，而 assign 不是。

赋值 32‘b2 会报错，因为已经规定为二进制，正确的写法为 32‘b10。