南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

计算机组成原理

实验二报告

班　　级 9191062301

学生姓名 孙傲歆

学　　号 919106840333

任课教师 王晓峰

2021 年 11 月

实验二　运算器实验

1.实验目的：

1.熟悉 LS-CPU-EXB-002 实验箱和软件平台 vivado
2.掌握利用该实验箱各项功能开发组成原理实验的方法
3.理解并掌握加法器的原理和设计
4.熟悉并运用 verilog 语言进行电路设计

2.实验设备：

1.装有 Xilinx Vivado 的计算机一台
2.LS-CPU-EXB-002 教学系统实验箱一套

3.实验内容：

1.熟悉硬件平台，学习软件平台和设计流程
2.熟悉计算机中运算器的原理并使用 verilog 语言编写相应代码，要求至少完成八种运算，包括算术运算和逻辑运算。
3.将以上设计作为一个单独的模块，设计一个外围模块去调用它，外围模块中需要调用封装好的触摸屏模块，通过实验箱提供的开关控制输入数据及运算类型，根据输入不同显示不同的运算结果。
4.进行综合布局布线，下载到实验箱的 FPGA 板上进行演示

4.实验思路：

本次实验要求实现运算器 alu，并且完成至少 8 个功能。这里我选择实现的 8 个功能分别是：加法、减法、有符号比较、无符号比较、按位与、按位或非、按位或、按位异或。

加法与减法的实现直接调用实验 1 中的加法模块即可。

按位与、按位或非、按位或、按位异或的实现也十分简单，分别直接调用“&”、“~”、“|”、“^”即可。

而有符号比较和无符号比较的实验较为复杂，或者所巧妙。首先使用加法模块对两数进行减法运算，对于无符号比较判断其大于 0 还是小于等于 0 即可；而对于有符号比较则是判断进位情况是大于 0 还是小于等于 0。

5.实验代码：

由于实验的调用 FPGA 板上的 IO 接口和触摸屏的相关代码已经给出，且其代

码量较大，这里仅展示主要部分的代码和引脚代码。

1）alu.v

```
module alu(
    input    [11:0] alu_control,
    input    [31:0] alu_src1,
    input    [31:0] alu_src2,
    output     [31:0] alu_result
    );
    reg [31:0] alu_result;
    wire alu_add;     //加法
    wire alu_sub;     //减法
    wire alu_slt;     //有符号比较，小于置位
    wire alu_sltu;    //无符号比较，小于置位
    wire alu_and;     //按位与
    wire alu_nor;     //按位或非
    wire alu_or;      //按位或
    wire alu_xor;     //按位异或

    assign alu_add    = alu_control[11];
    assign alu_sub    = alu_control[10];
    assign alu_slt    = alu_control[ 9];
    assign alu_sltu = alu_control[ 8];
    assign alu_and    = alu_control[ 7];
    assign alu_nor    = alu_control[ 6];
    assign alu_or     = alu_control[ 5];
    assign alu_xor    = alu_control[ 4];

    wire [31:0] add_sub_result;
    wire [31:0] slt_result;
    wire [31:0] sltu_result;
    wire [31:0] and_result;
    wire [31:0] nor_result;
    wire [31:0] or_result;
    wire [31:0] xor_result;

    wire signed [31:0] temp_src1;
    assign temp_src1 = alu_src1;
    assign and_result = alu_src1 & alu_src2;
    assign or_result    = alu_src1 | alu_src2;
    assign nor_result = ~or_result;
    assign xor_result = alu_src1 ^ alu_src2;
    assign slt_result = adder_result[31] ? 1'b1 : 1'b0;
    assign sltu_result = adder_cout ? 1'b0 : 1'b1;
```

```
    wire [31:0] adder_operand1;
    wire [31:0] adder_operand2;
    wire        adder_cin     ;
    wire [31:0] adder_result  ;
    wire        adder_cout    ;
    assign adder_operand1 = alu_src1;
    assign adder_operand2 = alu_add ? alu_src2 : ~alu_src2;
    assign adder_cin      = ~alu_add;
    adder adder_module(       //调用加法模块
    .operand1(adder_operand1),
    .operand2(adder_operand2),
    .cin     (adder_cin     ),
    .result  (adder_result  ),
    .cout    (adder_cout    )
    );
assign add_sub_result = adder_result;
    always@(*)
    begin
        if(alu_add | alu_sub)
            alu_result <= add_sub_result;
            else if(alu_slt)
            alu_result <= slt_result;
            else if(alu_sltu)
            alu_result <= sltu_result;
            else if(alu_and)
            alu_result <= and_result;
            else if(alu_nor)
            alu_result <= nor_result;
            else if(alu_or)
            alu_result <= or_result;
            else if(alu_xor)
            alu_result <= xor_result;
    end
endmodule
```

2）alu_display.v

```
//触摸板输入
    always @(posedge clk)
    begin
        if (!resetn)
        begin
            alu_control <= 12'd0;
        end
        else if (input_valid && input_sel==2'b00)
```

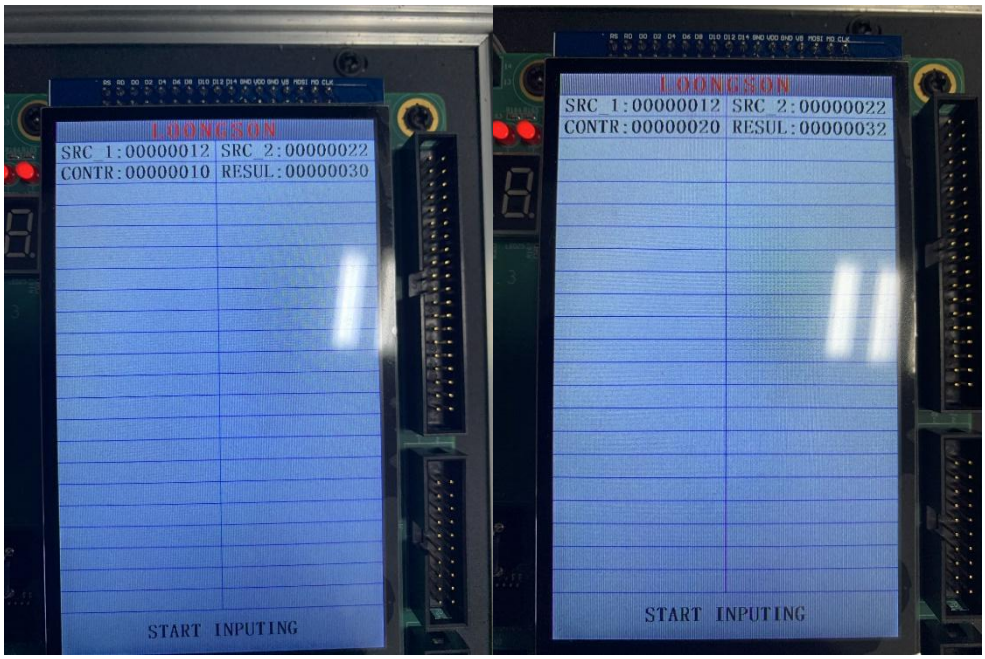
```
            begin
                alu_control <= input_value[11:0];
            end
        end

        always @(posedge clk)
        begin
            if (!resetn)
            begin
                alu_src1 <= 32'd0;
            end
            else if (input_valid && input_sel==2'b10)
            begin
                alu_src1 <= input_value;
            end
        end

        always @(posedge clk)
        begin
            if (!resetn)
            begin
                alu_src2 <= 32'd0;
            end
            else if (input_valid && input_sel==2'b11)
            begin
                alu_src2 <= input_value;
            end
        end
    //触摸板输出
        always @(posedge clk)
        begin
            case(display_number)
                6'd1 :
                begin
                    display_valid <= 1'b1;
                    display_name  <= "SRC_1";
                    display_value <= alu_src1;
                end
                6'd2 :
                begin
                    display_valid <= 1'b1;
                    display_name  <= "SRC_2";
                    display_value <= alu_src2;
                end
```

```
            6'd3 :
            begin
                display_valid <= 1'b1;
                display_name    <= "CONTR";
                display_value <={20'd0, alu_control};
            end
            6'd4 :
            begin
                display_valid <= 1'b1;
                display_name    <= "RESUL";
                display_value <= alu_result;
            end
            default :
            begin
                display_valid <= 1'b0;
                display_name    <= 40'd0;
                display_value <= 32'd0;
            end
        endcase
    end
```

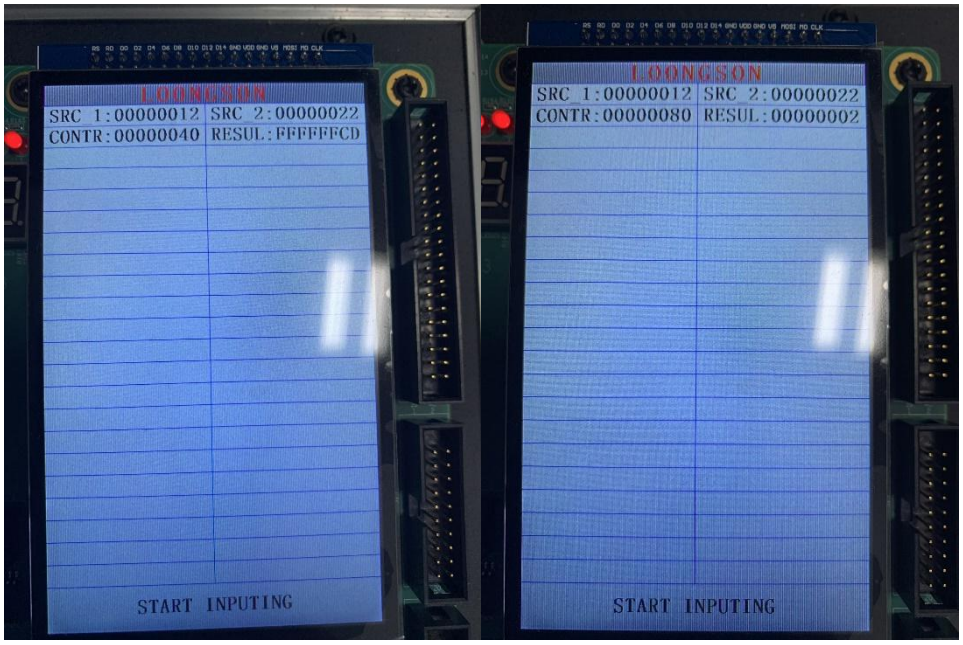
6. 结果分析：

控制信号与对应的功能如下：

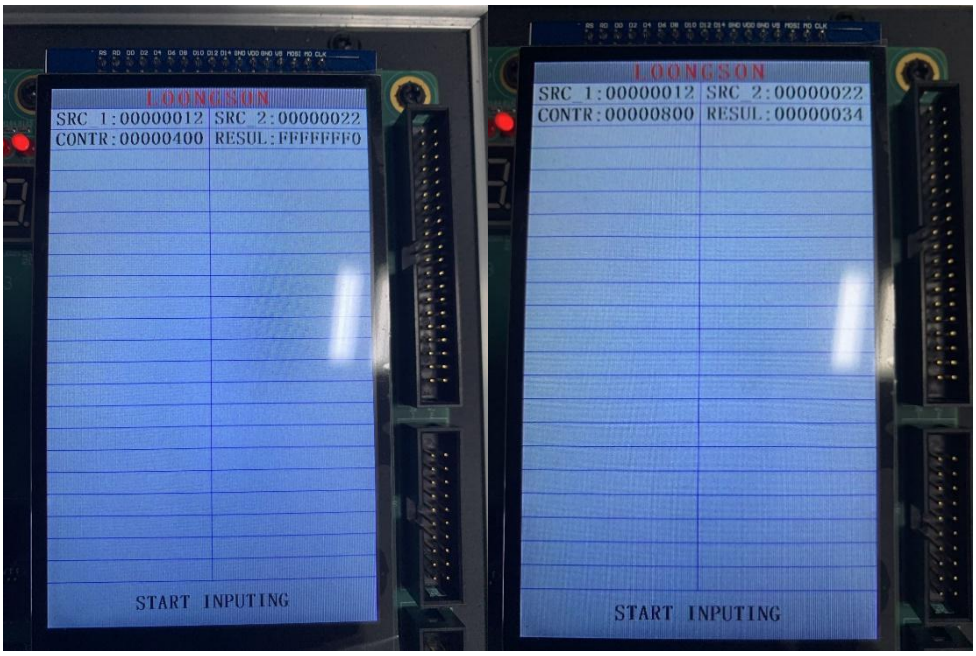
010H　按位异或
020H　按位或
040H　按位或非
080H　按位与
100H　无符号比较
200H　有符号比较
400H　减
800H　加



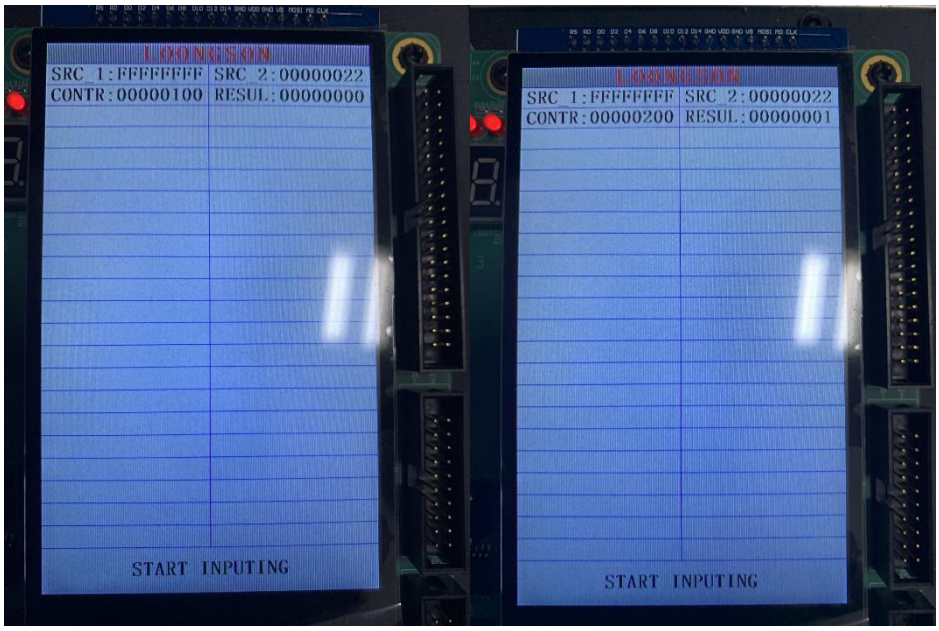
上图为运算器执行异或以及或运算的结果。CONTR 是控制信号输入，SRC_1 和 SRC_2 分别是两个操作数，RESULT 是结果。12H=00010010，22H=00100010，所以两数进行异或操作得到 00110000=30H，两数进行或操作得到 00110010=32H，结果正确。



上图为运算器执行或非以及与操作的运算结果。12H=00010010，22H=00100010，所以两数进行或非操作高位全为 1111=FH，低两位为 11001101=CDH，两数进行与操作得到 00000010=02H，结果正确。



上图为运算器执行减法及加法操作的运算结果。12H=00010010，22H=00100010，所以两数进行减操作得到负数-10H，结果以补码形式表示为FFFFFFF0H，两数进行加操作得到34H，结果正确。



上图为运算器执行无符号比较及有符号比较操作的运算结果。当进行无符号比较时 SRC_1 表示为正数，显然比 SRC_2 大，所以 RESULT 置 0；当进行有符号比较式 SRC_1 表示为负数，显示比 SRC_2 这个正数要小，所以 RESULT 置 1，结果正确。

7. 心得体会：

本次实验相较于第一次实验复杂了许多，但我们也可看到，实际上第一次实验就是在为第二次实验做准备。有第一次实验加法器的铺垫，我们就能较为容易地实现加减以及比较功能。

而且通过本次实验我更加充分地理解了计算机内部运算器的各种功能是如何控制并且进行运算的。当然我知道，真正 CPU 内部的运算器功能远不止 8 种这么少。为了满足 CPU 各种指令、各种操作的需求，运算器一定是十分复杂的。

总之，这次实验花了我不少的时间与精力，当然这可能与我实验前没有充分地预习有关，这样说明了课前预习的重要性。接下来还有两次实验，希望下两次实验能够又快又好地完成吧。