中国海洋大学计算机科学与技术系

实验报告

姓名：岳宇轩　　年级：2019　　专业：19 慧与

科目：计算机组成原理　　题目：ALU

实验时间: 2021 年 5 月 6 日　　实验教师: 张巍

一、实验结果及截图分析：

（※代码挖空的部分必须截图或复制）

1. 代码补全 alu.v

```
`timescale 1ns / 1ps
//*************************************************************************
//    > 文件名: alu.v
//    > 描述   ：ALU 模块，可做 12 种操作
//    > 作者   : LOONGSON
//    > 日期   : 2016-04-14
//*************************************************************************
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
    wire alu_sll;    //逻辑左移
    wire alu_srl;    //逻辑右移
    wire alu_sra;    //算数右移
```

```
wire alu_lui;     //高位加载

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];
assign alu_sll    = alu_control[ 3];
assign alu_srl    = alu_control[ 2];
assign alu_sra    = alu_control[ 1];
assign alu_lui    = alu_control[ 0];

wire [31:0] add_sub_result; //加减结果，减法用加法来实现
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 [31:0] sll_result;
wire [31:0] srl_result;
wire [31:0] sra_result;
wire [31:0] lui_result;

wire signed [31:0] temp_src1;     //带符号数的临时变量
assign temp_src1 = alu_src1;      //方便后面对 alu_src1 进行算数右移

assign and_result = alu_src1 & alu_src2;             //按位与
assign or_result    = alu_src1 | alu_src2;           //按位或
assign nor_result = ~or_result;                      //或非
assign xor_result = alu_src1 ^ alu_src2;             //异或
assign lui_result = {alu_src2[15:0], 16'd0};      //高位加载，第二个操作数的低十六位加载到高十六位上
assign sll_result = alu_src1 << alu_src2;            //逻辑左移
assign srl_result = alu_src1 >> alu_src2;            //逻辑右移
assign slt_result = adder_result[31] ? 1'b1 : 1'b0;     // 带符号数小于置位
assign sltu_result = adder_cout ? 1'b0 : 1'b1;        //无符号数小于置位
assign sra_result = temp_src1 >>> alu_src2;        //算数右移

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;      //默认进行减法，为 slt 和 sltu 服务
    assign adder_cin      = ~alu_add;
    adder adder_module(      //调用实验一中的 adder.v 加法模块
    .operand1(adder_operand1),
    .operand2(adder_operand2),
    .cin      (adder_cin     ),
    .result   (adder_result  ),
    .cout     (adder_cout    )
    );

//代码补全部分
assign add_sub_result = adder_result;  //先给加减法结果赋值

    always@(*)
    begin

        if(alu_add | alu_sub)  //如果是加法或者减法操作，则把加减法结果 add_sub_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= add_sub_result;
            else if(alu_slt)  //如果是有符号比较，则把有符号比较结果 slt_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= slt_result;
            else if(alu_sltu)  //如果是无符号比较，则把无符号比较结果 sltu_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= sltu_result;
            else if(alu_and)  //如果是按位与操作，则把按位与操作结果 and_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= and_result;
            else if(alu_nor)  //如果是按位或非操作，则把按位或非操作结果 nor_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= nor_result;
            else if(alu_or)  //如果是按位或操作，则把按位或操作结果 or_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= or_result;
            else if(alu_xor)  //如果是按位异或操作，则把按位异或操作结果 xor_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= xor_result;
            else if(alu_sll)  //如果是逻辑左移操作，则把逻辑左移的结果 alu_sll 放入 alu 最终的运算结果 alu_result 中
            alu_result <= sll_result;
            else if(alu_srl)  //如果是逻辑右移操作，则把逻辑右移的结果 alu_srl 放入 alu 最终的运算结果 alu_result 中
            alu_result <= srl_result;
            else if(alu_sra)  //如果是算术右移操作，则把算术右移的结果 alu_sra 放入 alu 最终的运算结果 alu_result 中
            alu_result <= sra_result;
            else if(alu_lui)  //如果是高位加载操作，则把高位加载操作的结果 lui_result 放入 alu 最终的运算结果 alu_result 中
            alu_result <= lui_result;

    end
```

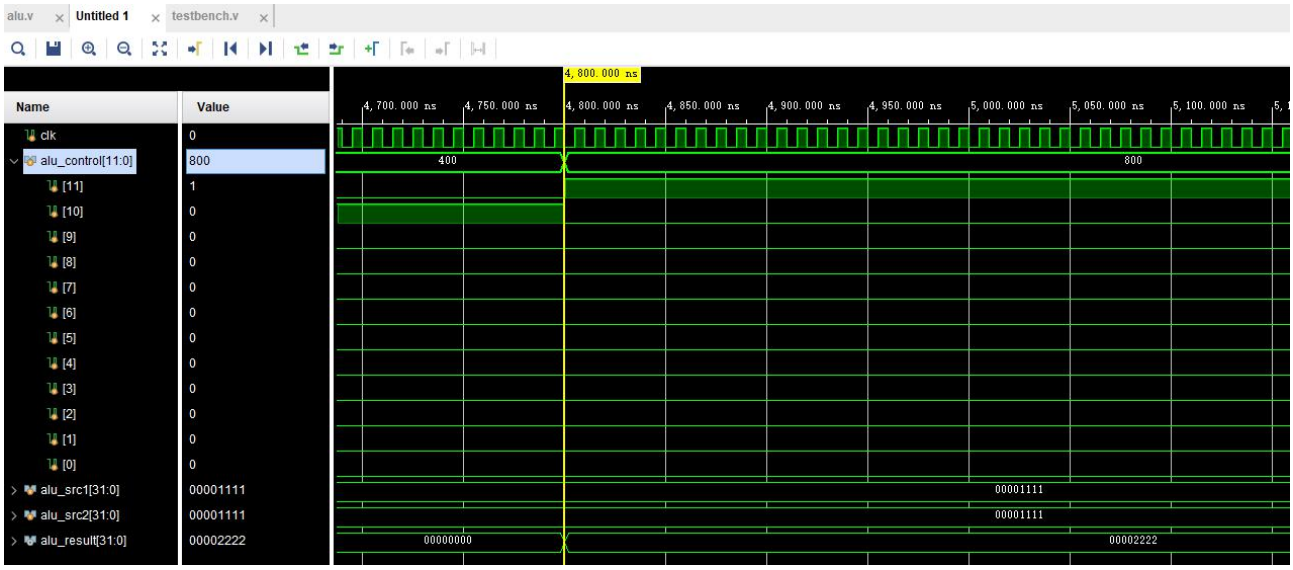
endmodule

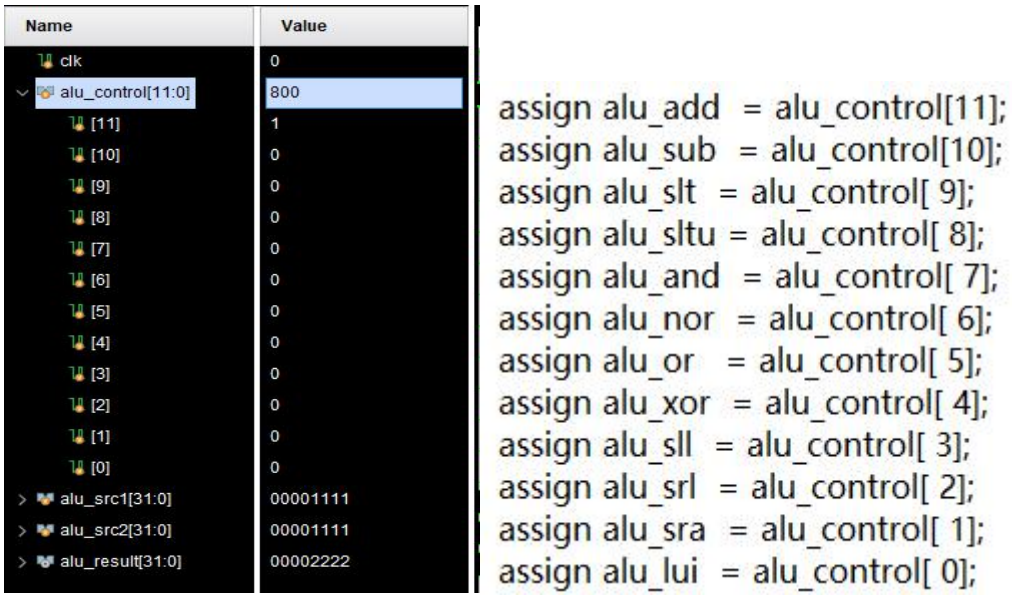
2.添加 adder.v　adder_display.v　alu_display.v　testbench.v

之前实验用到过的文件，这里不再分析解释了

3. 完成仿真，生成仿真波形图像

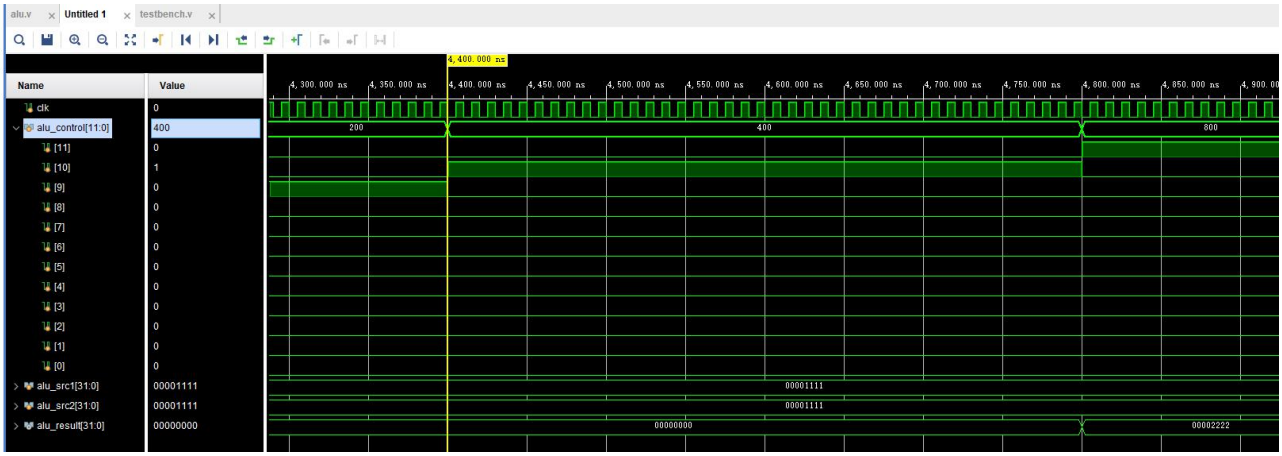
3.1 加法

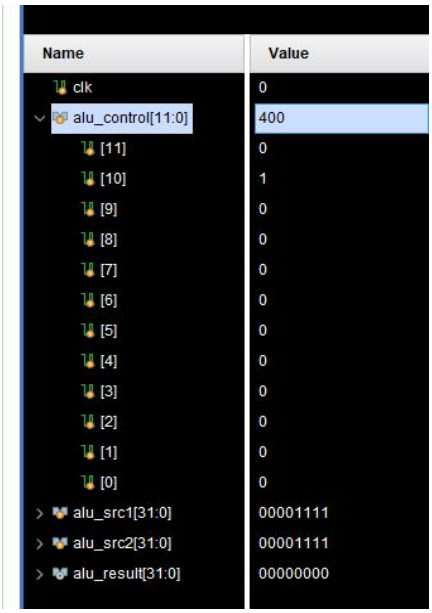




可以看到，alu 控制信号为 1000 0000 0000B，对照上方右侧定义，alu 操作为加法操作，两个操作数的值分别为 1111H 和 1111H，加法运算后的结果为 2222H。

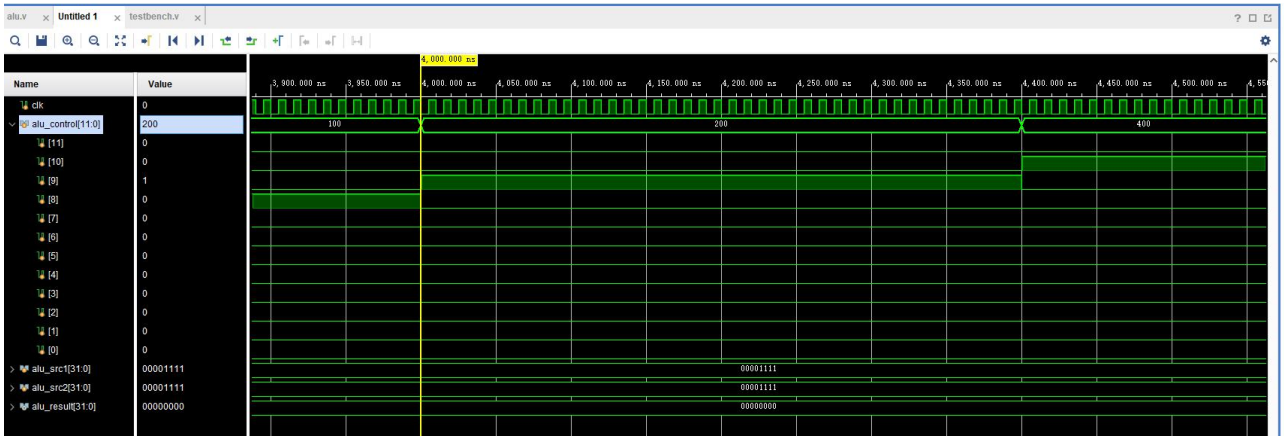
3.2 减法操作

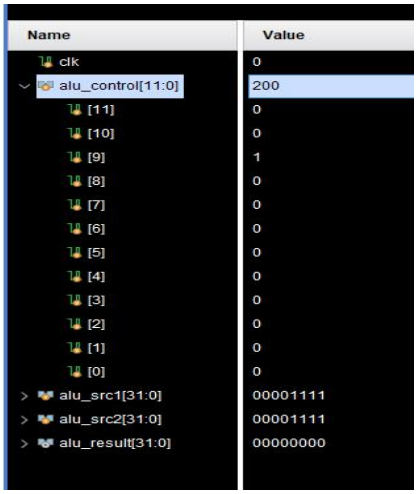




可以看到，alu 控制器值为 0100 0000 0000B，选择了减法操作，1111H-1111H = 0

3.3 有符号比较操作

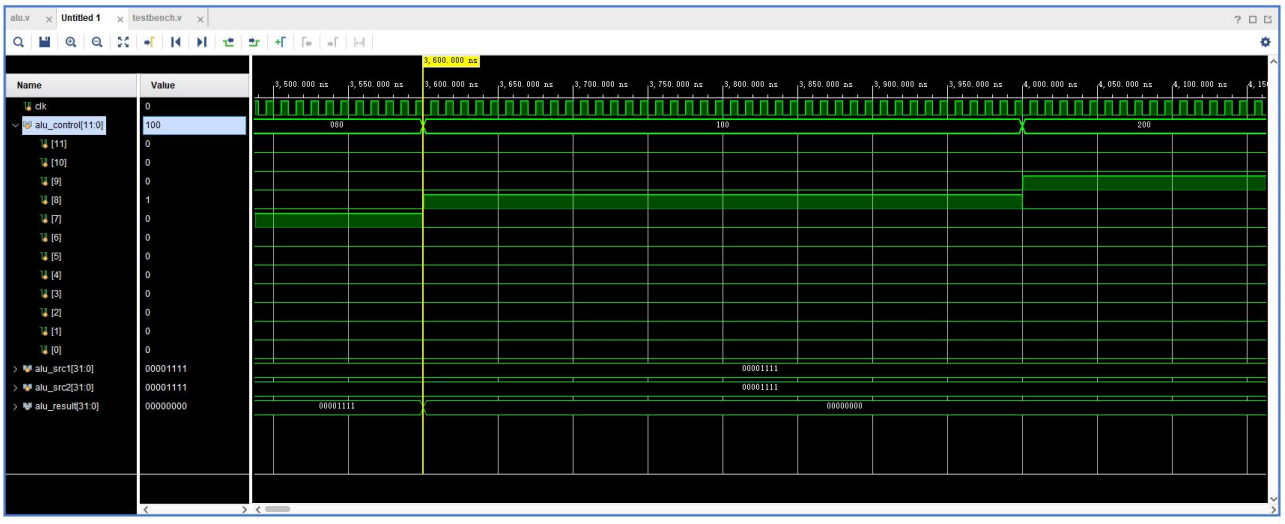


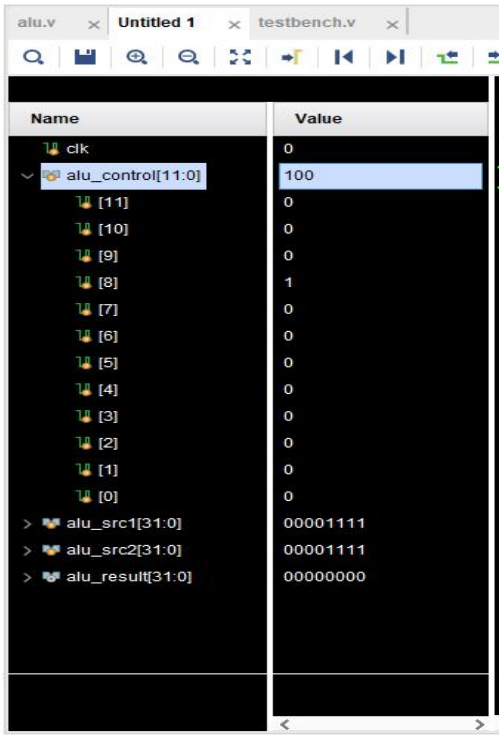


alu 控制器值为 0010 0000 0000B，选择了有符号数比较操作。

两操作数的机器码相同，按照有符号数比较结果是相同的。

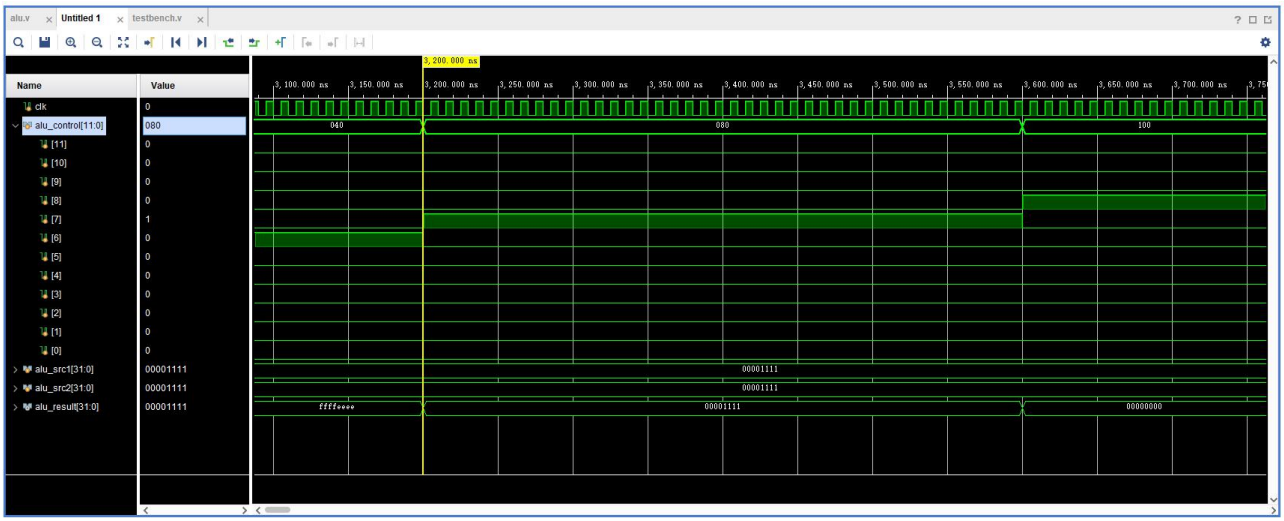
3.4 无符号比较操作

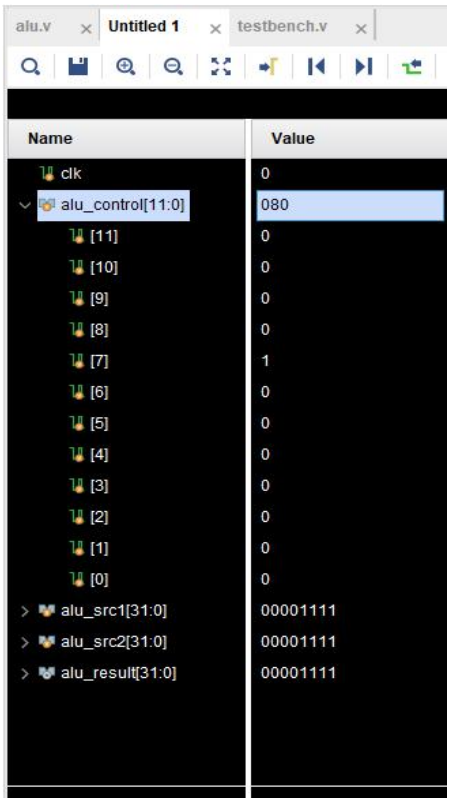




alu 控制器值为 0001 0000 0000B，选择了无符号数比较操作。
两操作数的机器码相同，按照无符号数比较结果是相同的。

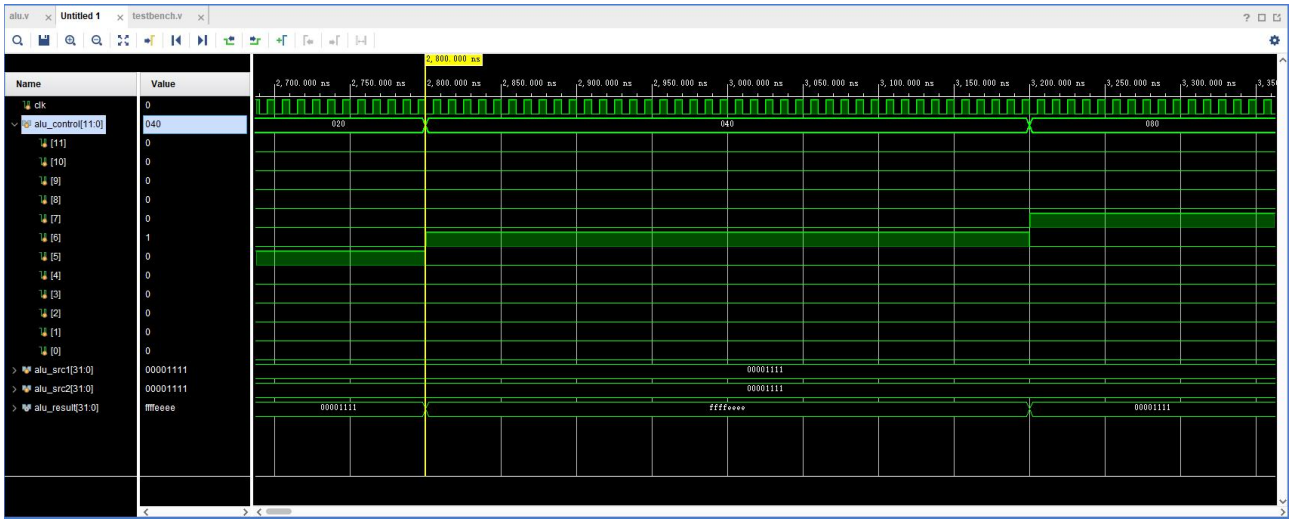
3.5 按位与操作

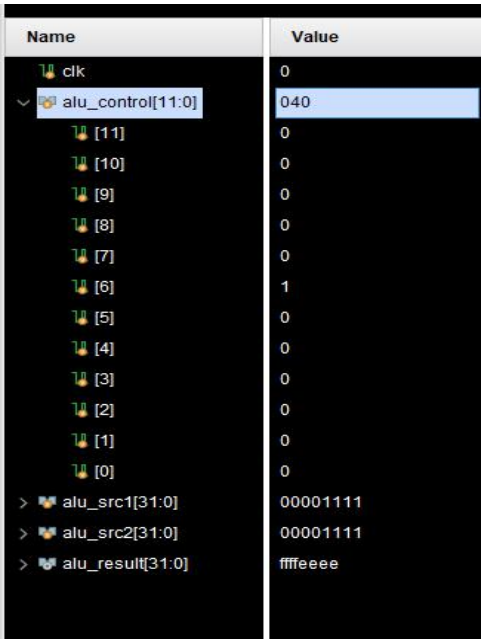




alu 控制器值为 0000 1000 0000B，选择按位与操作 1111H 与 1111H 进行按位与操作，结果为 1111H

3.6 按位或非操作

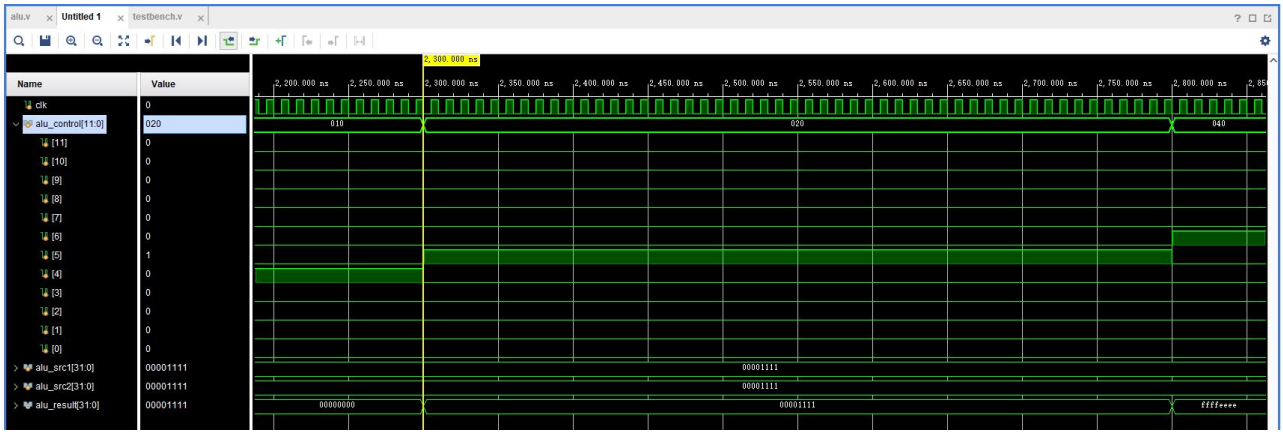


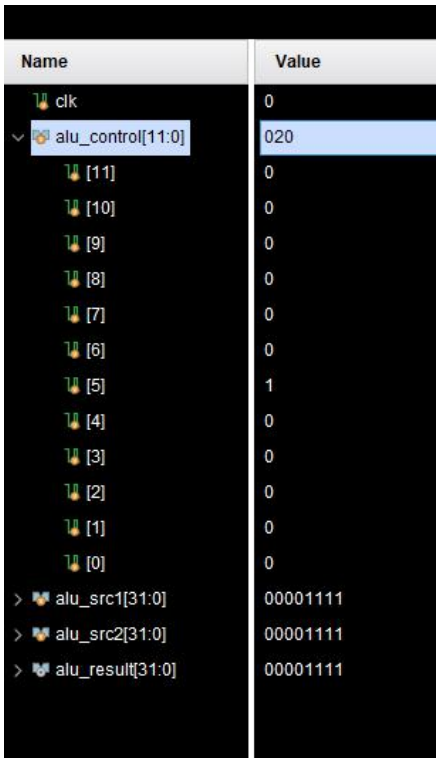


alu 控制器值为 0000 0100 0000B，选择操作为按位或非

0000 0000 0000 0000 0001 0001 0001 0001B 与 0000 0000 0000 0000 0001 0001 0001 0001B 按位或非结果为 1111 1111 1111 1111 1110 1110 1110 1110B，十六进制表示为 ffff eeeeH

3.7 按位或操作

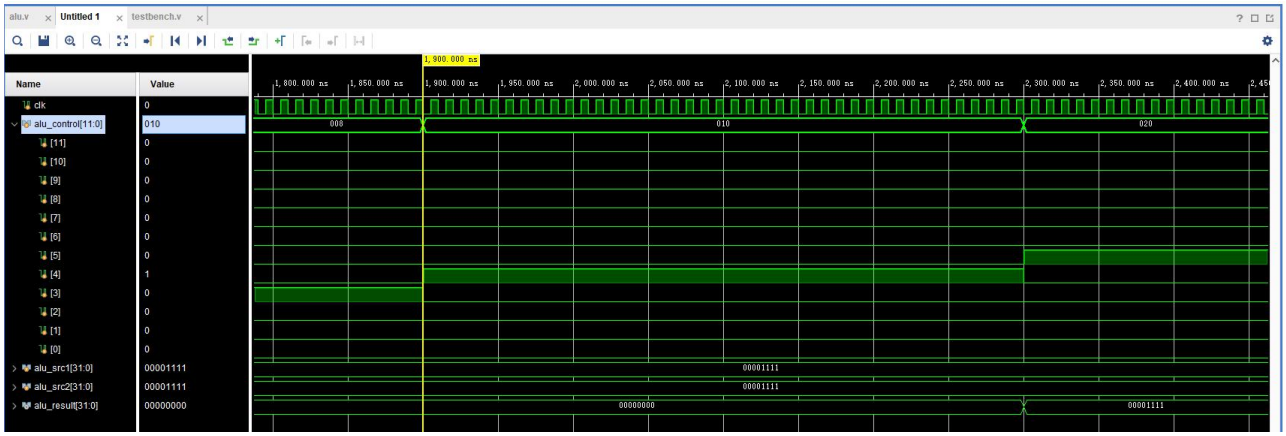


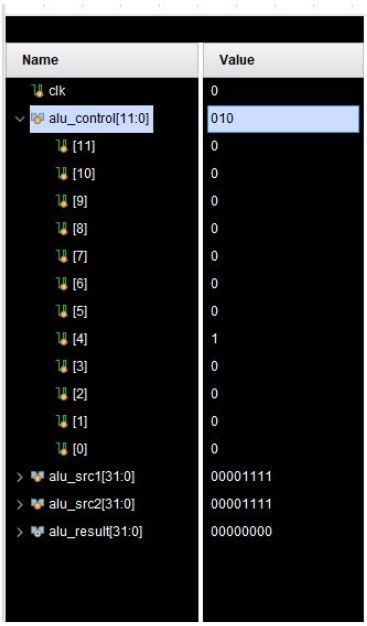


alu 控制器值为 0000 0010 0000B 选择了按位或操作

1111H 与 1111H 按位或，结果为 0001 0001 0001 0001B，十六进制表示为 1111H

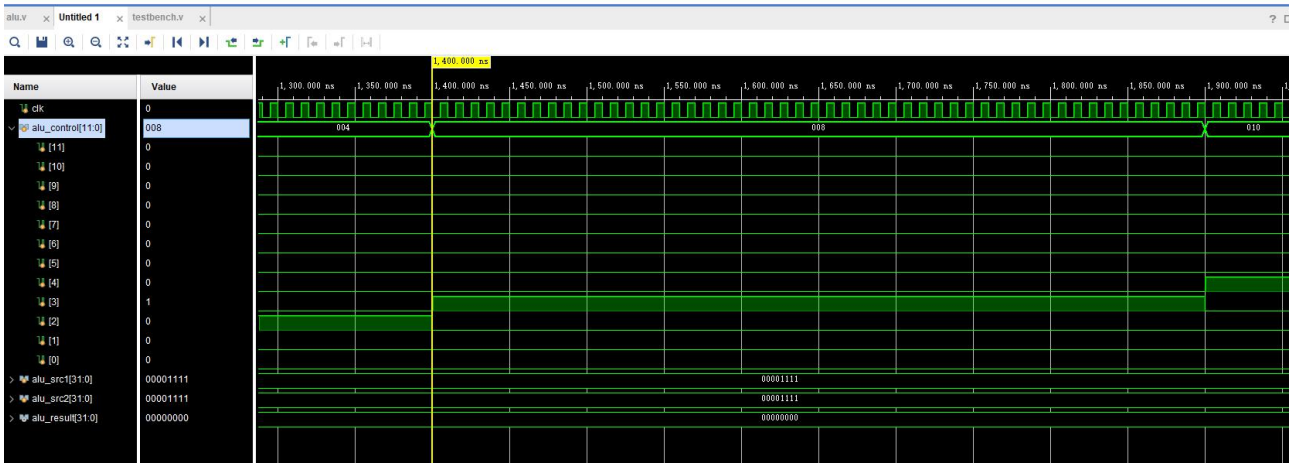
3.8 按位异或操作

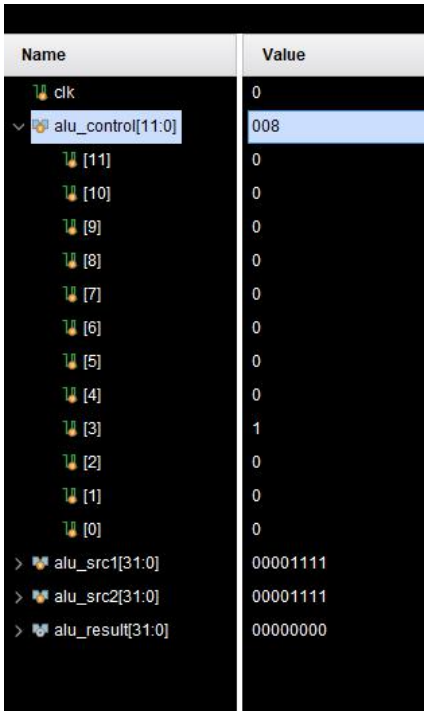




alu 控制器值为 0000 0001 0000B，选择操作为按位异或，0000 0000 0000 0000 0001 0001 0001 0001B 与 0000 0000 0000 0000 0001 0001 0001 0001B 按位异或结果为
0000 0000 0000 0000 0000 0000 0000 0000B，十六进制表示为 0000 0000H

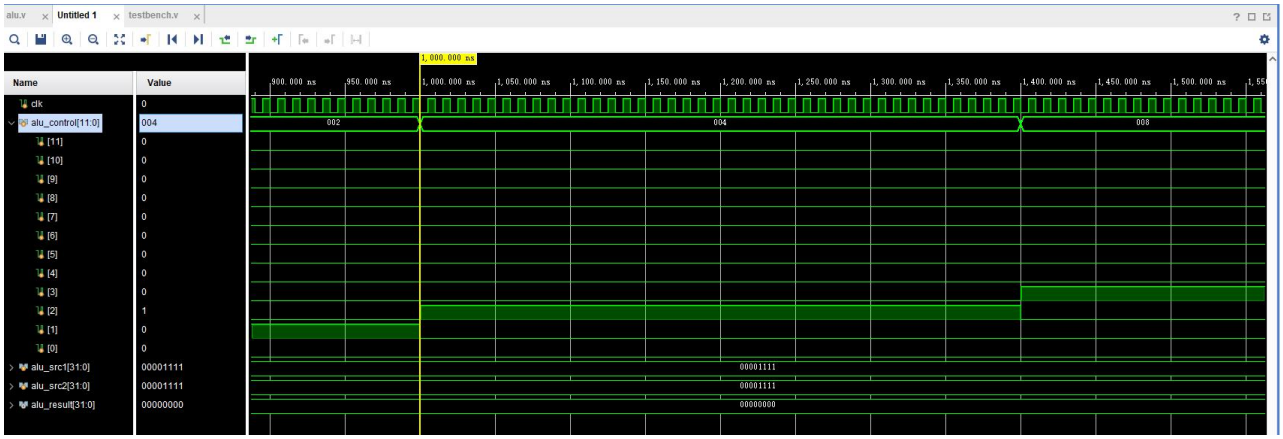
3.9 逻辑左移操作

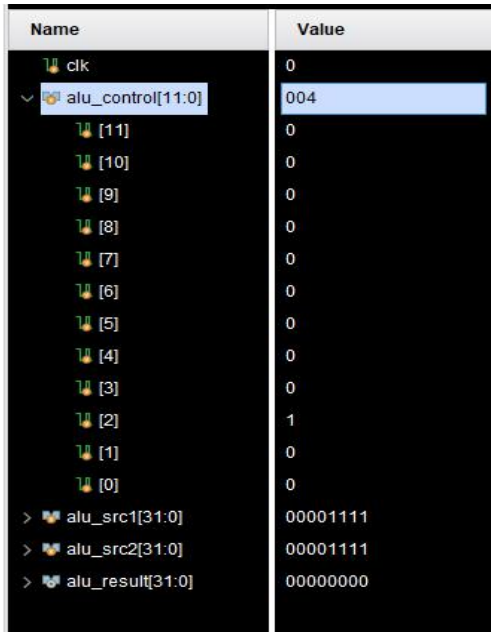




alu控制器值为0000 0000 1000B，选择操作为逻辑左移，高位移出，高位补零，0000 0000 0000 0000 0001 0001 0001 0001B 逻辑左移 0000 0000 0000 0000 0001 0001 0001 0001B 位结果为
0000 0000 0000 0000 0000 0000 0000 0000B，十六进制表示为 0000 0000H

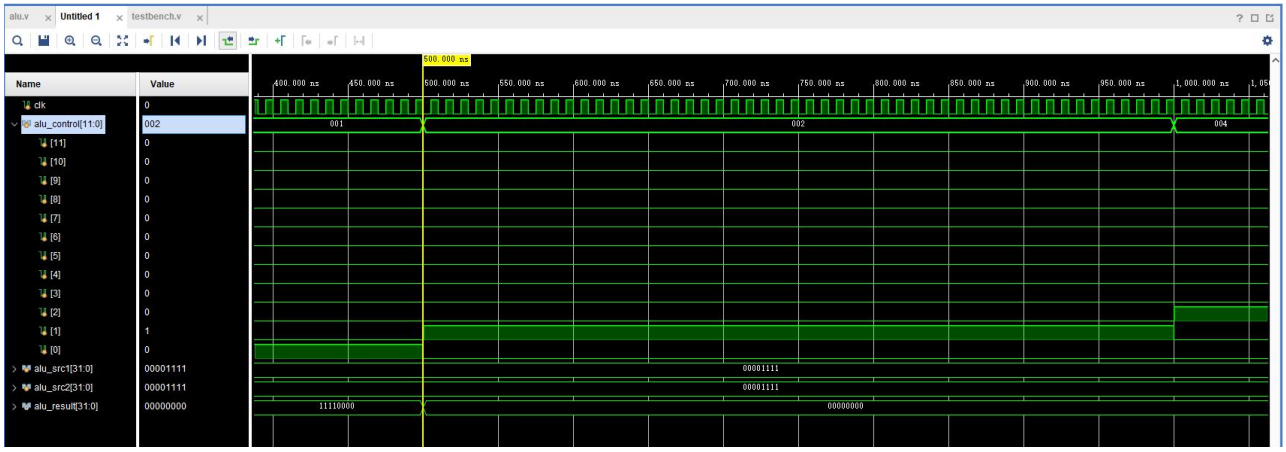
3.10 逻辑左移操作

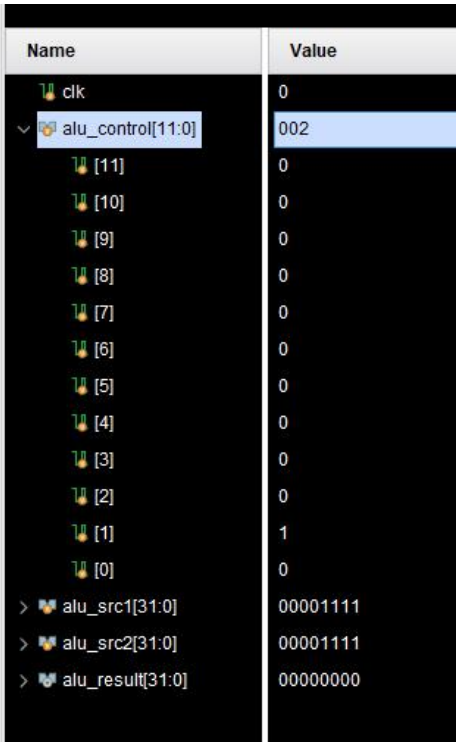




alu控制器值为0000 0000 0100B，选择操作为逻辑右移，低位移出，高位补零，0000 0000 0000 0000 0001 0001 0001 0001B 逻辑右移 0000 0000 0000 0000 0001 0001 0001 0001B 位结果为
0000 0000 0000 0000 0000 0000 0000 0000B，十六进制表示为 0000 0000H

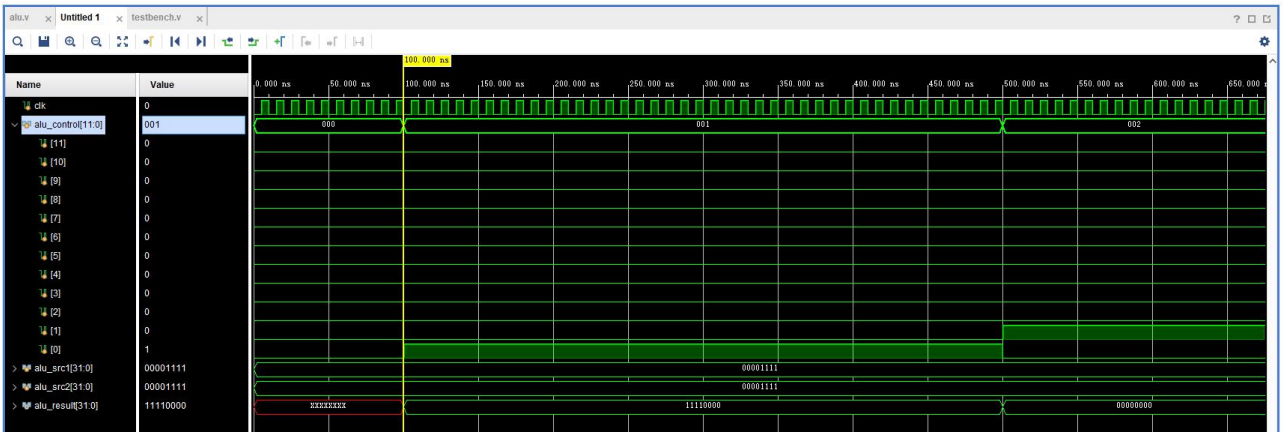
3.11 算术左移操作

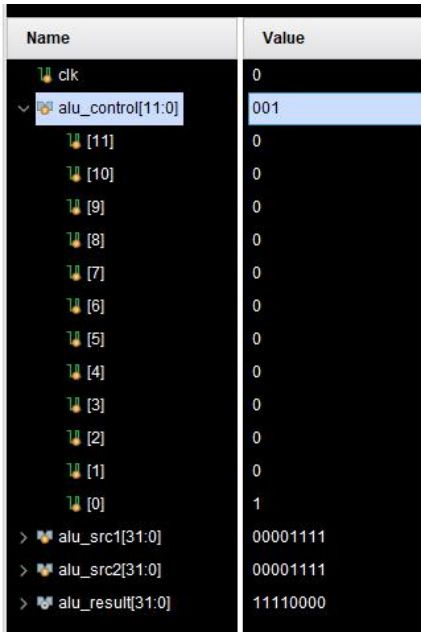




alu控制器值为0000 0000 0010B，选择操作为算术左移，高位移出，低位补零，0000 0000 0000 0000 0001 0001 0001 0001B 算术左移 0000 0000 0000 0000 0001 0001 0001 0001B 位结果为
0000 0000 0000 0000 0000 0000 0000 0000B，十六进制表示为 0000 0000H

3.12 高位加载操作





alu 控制器值为 0000 0000 0001B，选择操作为高位加载，第二个操作数的低十六位加载到高十六位上，低十六位填零。 0000 0000 0000 0000 0001 0001 0001 0001B 高位加载的结果为 0001 0001 0001 0001 0000 0000 0000 0000B，十六进制表示为 1111 0000H

二、实验总结

通过这次实验，我对于 alu 的原理有了更加清楚的认知。

首先，alu 具有多种功能，如加法，减法等，因此它需要一个控制信号，来告诉他需要进行那种操作。alu 有两个源操作数，一个目的操作数。在代码补全的过程中，实现了对 alu 控制信号的判断，凭此来决定将那种运算的结果作为 alu 运算的结果进行输出。以及，在实现每种操作的过程中，我对于各种运算操作的原理和实现过程有了更加清楚的认知。

本次实验最大的收获是对于仿真波形图像的理解。我学习了对于含有选择控制信号电路的仿真波形图像的观察和分析，控制信号不同的值代表选择了不同的输出通路。在本次实验中，两个源操作数一直不变，而控制信号是不断变化的，alu 结果也随着控制信号的改变对应输出相应运算后的结果。通过观察每

个时钟周期前后各个值的变化，能更好的理解 alu 使用过程中的原理。