中山大学数据科学与计算机学院本科生实验报告

（2017 学年春季学期）

课程名称：**计算机组成原理实验**　　任课教师：郭雪梅　　助教：李声涛、王绍菊

年级&班级	15 级 18 班	专业(方向)	软件工程（移动信息工程）
学号	15352408	姓名	张镓伟
电话	13531810182	Email	709075442@qq.com
开始日期	2017.5.19	完成日期	2017.5.19

一、实验题目

设计一个 ALU 运算器主要执行 5 种操作：与，或，加，减，小于设置。其中一个为可从 basys3 板输入的 8 位操作数，输入后会扩展成 32 位，另一个操作数为常数，我设为 15。

二、实验目的

(1) 了解运算器的组成结构。

(2) 掌握运算器的工作原理。

三、实验内容

1. 实验步骤

（1）打开 vivadao，建立名为 ALU 的 RTL 项目。

（2）增加三个代码文件，依次为 signext.v,alu.v,top.v。每个文件的代码及相应功能如下：

signext.v：将 8 位输入扩展成 32 位

```
module signext(
    input [7:0] inst, // 输入8位
    output [31:0] data // 输出32位
);
//将8位输入扩展成32位输出
// assign data = inst[7:7]?{24'hffffff,inst}:{24'h000000,inst};
assign data = {24'h000000,inst};
endmodule
```

alu.v：运算器模块，根据aluCtr对两个操作数input1和input2选择相应的运算，结果存在aluRes，zero是判断结果是否为0，

```
module alu(
    input [31:0] input1,
    input [31:0] input2,
    input [3:0] aluCtr,
```

```
    output reg[31:0] aluRes,
    output reg zero
);
always @(input1 or input2 or aluCtr) // 运算数或控制码变化时操作
begin
    case(aluCtr)
     4'b0110: // 减
        begin
            aluRes = input1 - input2;
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
        end
    4'b0010: // 加
        begin
            aluRes = input1 + input2;
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
        end
    4'b0000: // 与
        begin
            aluRes = input1 & input2;
            if(aluRes == 0)
                 zero = 1;
             else
                 zero = 0;
        end
    4'b0001: // 或
        begin
            aluRes = input1 | input2;
            if(aluRes == 0)
                zero = 1;
            else
                zero = 0;
        end
    4'b1100: // 异或
        begin
            aluRes = ~(input1 | input2);
            if(aluRes == 0)
                 zero = 1;
            else
                 zero = 0;
        end
```

```
        4'b0111: // 小于设置
            begin
                if(input1<input2)
                    aluRes = 1;
                else
                    aluRes = 0;
                zero=0;
            end
        default:
            begin
            aluRes = 0;
            zero=0;
        end
    endcase
  end
endmodule
```

top.v：顶层模块，依次调用扩展数字模块和ALU模块，设置input2为常数15.其输出是ALU输出的高四位和低四位拼接，以及一个zero。

```
module top(
    input [7:0] input1,
    input [3:0] aluCtr,
    output [7:0] output1,
    output  zero
);
wire [31:0] input2;
assign input2=32'h0000_000f; //input2=15
wire [31:0] aluRes;
wire[31:0] expand;
//扩展数字模块
signext sign_expand(
            .inst(input1),
            .data(expand)
        );
// 实例化ALU模块
alu myalu(.input1(expand),
        .input2(input2),
        .aluCtr(aluCtr),
        .aluRes(aluRes),
        .zero(zero));
assign output1={aluRes[31:28],aluRes[3:0]};
endmodule
```

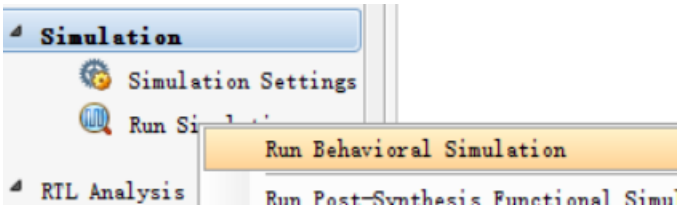
（3）在 Add Sources 里新建仿真文件 alusim。输入仿真代码。仿真代码如下：

```
`timescale 1ns / 1ps
module alusim;
// Inputs
```

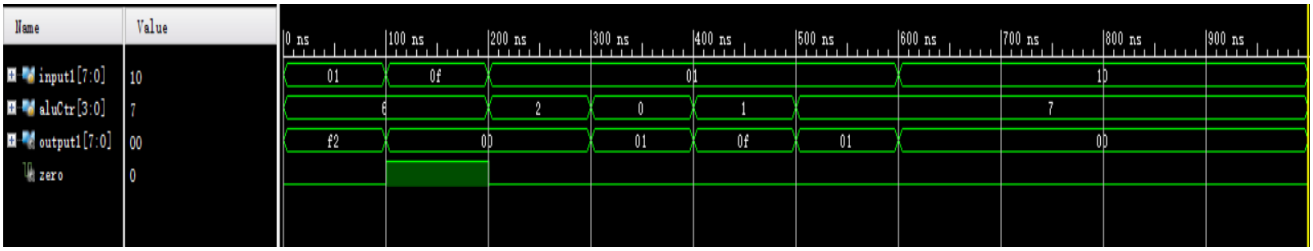
```
reg [7:0] input1;
reg [3:0] aluCtr;
wire [7:0] output1;
wire zero;
// Instantiate the Unit Under Test (UUT)
top uut (
    .input1(input1),
    .aluCtr(aluCtr),
    .output1(output1),
    .zero(zero)
);
initial begin
// Initialize Inputs
    input1 = 1;          //1-15
    aluCtr = 4'b0110;
    #100;                //15-15
        input1 = 15;
        aluCtr = 4'b0110;
    #100                 //1+15
        input1 = 1;
        aluCtr = 4'b0010;
    #100                 //1&15
        input1 = 1;
        aluCtr = 4'b0000;
    #100                 //1|15
        input1 = 1;
        aluCtr = 4'b0001;
    #100                 //1<15
        input1 = 1;
        aluCtr = 4'b0111;
    #100             //16<15
        input1 = 16;
        aluCtr = 4'b0111;
end
endmodule
```

每句时延后的功能都加了注释，方便观察波形是否正确

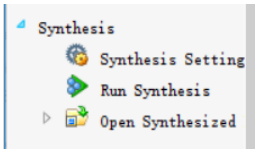
（4）点击 Run Simulation 选择 Run Behavioral Simulation 开始仿真。

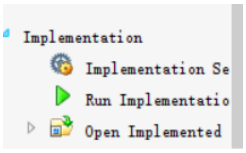


（5）与仿真代码每句的注释相比较，发现波形符合预期。

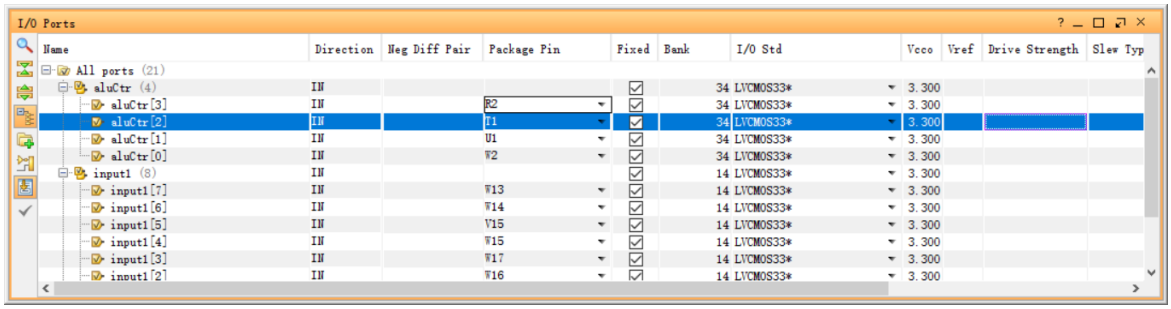


（6）仿真完成后点击 Run Synthesis 完成综合。综合完成后点击 Run Implementation 完成实现





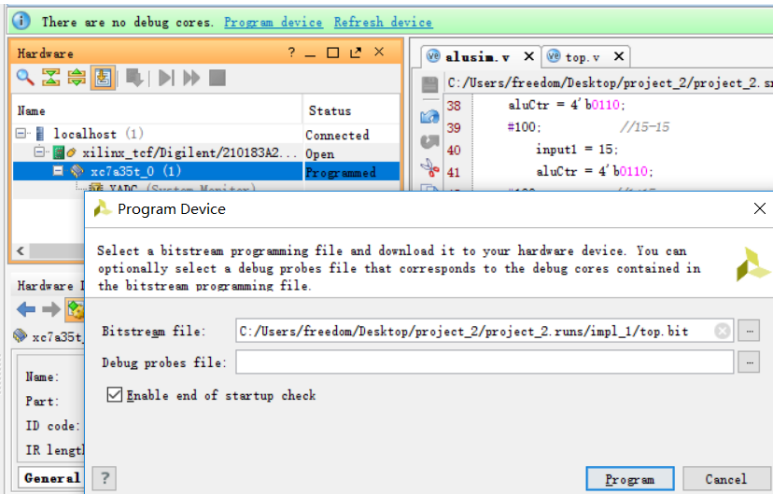
（7）实现完成之后 Open Implemented Design，完成 IO 管脚分配。其中 aluCtr[3:0]对应 SW15~SW12；input1[7:0]对应 SW7~SW0；output1[7:4]对应 led15~led12，是结果的高 4 位；output1[3:0]对应 led12~led8，是结果的低 4 位；zero 对应 led0，指示结果是否为 0。



（8）点击 Genreate Bitstream 生成比特流文件。

（9）点击 Hardware Manager 下的 Open Target,添加板子。

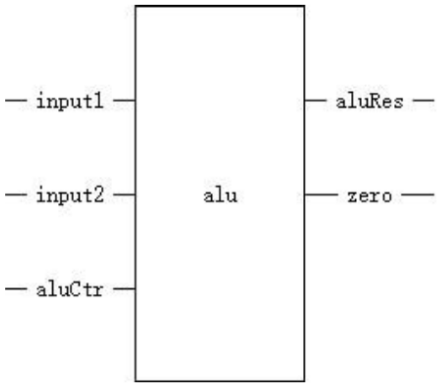
（10）点击 Hardware Manager 下的 Program Device，选择板子，烧录 bit 文件。



（11）完成烧录，就可以使用板子了。

2. 实验原理

input	input1[31:0]
input	input2[31:0]
input	aluCtr[3:0]
output	aluRes[31:0]
output	zero



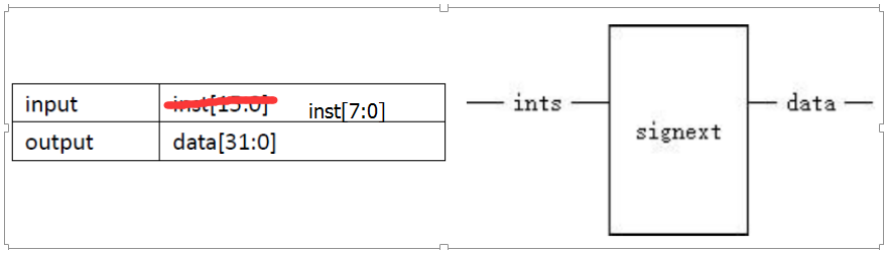
ALU 主要执行 5 种操作：与，或，加，减，小于设置。这五种操作可以使用四位的编码表示：0000，0001，0010，1110，0111。指令不同，则对应的 ALU 运算不同，所以该模块需要根据指令来控制 ALU 进行正确的运算。

根据 aluCtr 的不同进行相应的运算操作，将操作结果赋值到输出变量。具体的运算规则如下表所示：

表 1 运算规则

操作码 op	运算结果
0000	与
0001	或
0010	加
1110	减
0111	小于设置
其他	32'h00000000;

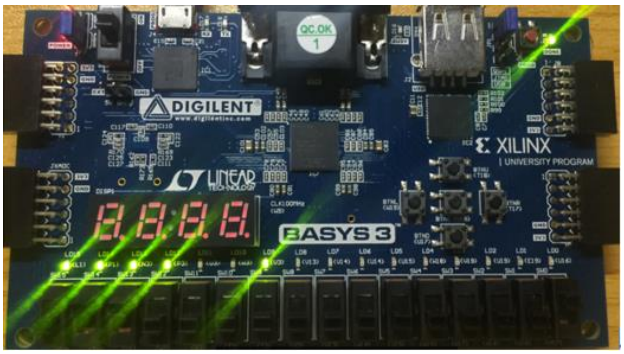
将 8 位输入扩展成 32 位，只要在前面补足 0 即可



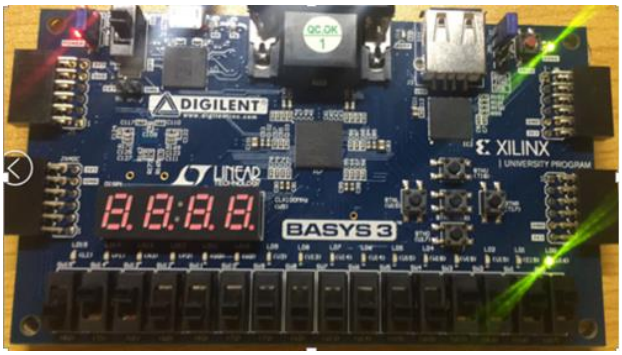
四、实验结果

下面展示6一些和仿真时一样的运算。

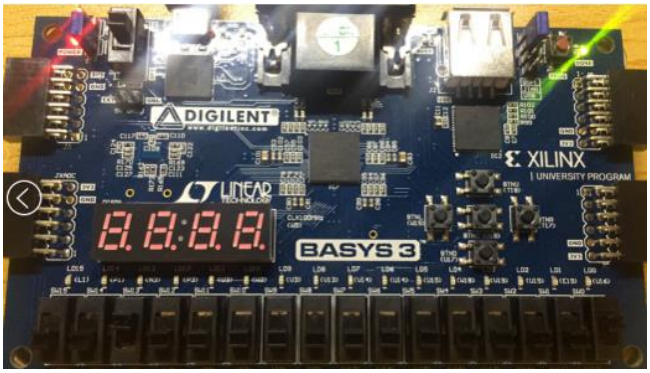
1-15=-14，可以知道-14补码为ffff fff2，高四位位1，末四位为0010，显示正确。



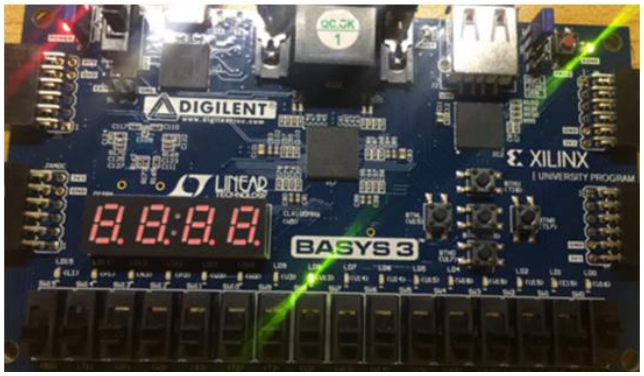
15-15=0，只有zero灯亮，显示正确。



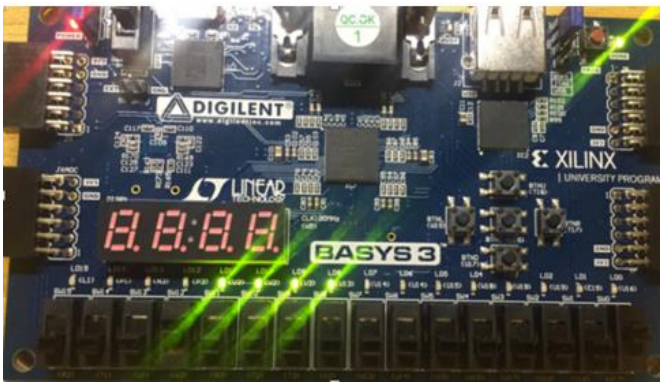
1+15=16=32’h00000010，高四位低四位都是0，所以没灯亮，显示正确。



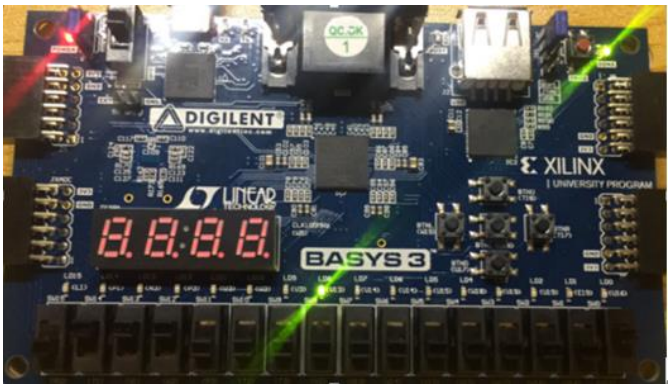
1&15=1;



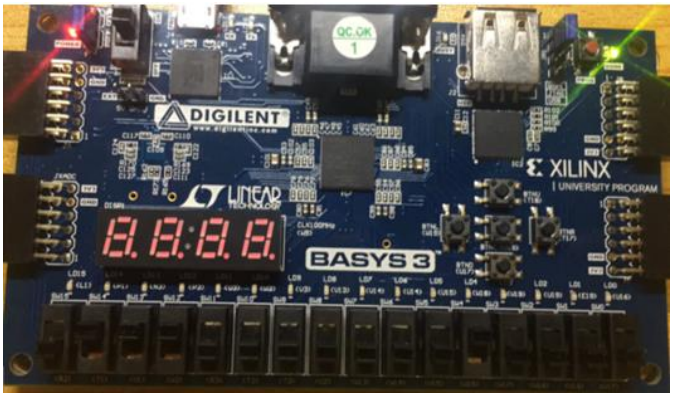
1|15=15；



1<15，符合要求，则结果为1.



16<15,不符合不等式关系，则结果为0，这里我设置了不等关系的结果为0，zero也不会亮



五、实验感想

这一次的实验是设计一个 ALU。虽然老师一如既往给出了代码，但是这次代码并不全，需要我们自行进行补充，同时还有的要做修改。这就比较好地考验了我们的动手能力以及对所学知识的掌握程度。这代码设计上我暂时没有遇到什么问题。但要注意的是随着我们的代码越来越多，适当的注释是非常有必要的，可以帮助我们快速理清代码的流程。