中山大学数据科学与计算机学院本科生实验报告

（2019 学年秋季学期）

课程名称：**计算机组成原理实验**　　任课教师：　　助教：

年级&班级	19 计科超算	专业(方向)	
学号	18324034	姓名	林天皓
电话		Email	linth5@mail2.sysu.edu.cn
开始日期	2020.10.23	完成日期	2020.11.6

一、 实验题目

实验 7 设计实现 ALU 功能并数码管显示

二、实验目的

⑴ 了解运算器的组成结构。

⑵ 掌握运算器的工作原理。

⑶ 掌握数码管的工作原理与使用方法，学会 IP 核封装调用。

三、实验内容与

1 实验原理

首先了解 vivado 开发板上七段四位数码管的连接方式，使用 verilog 语言描述将输入的由四条信号线组成的 0-9 的数字表示和表示要显示哪一位的四条信号线，通过语句转换为七段共阳极数码管上的具体显示数码，接下来通过 vivado 中的封装为 IP 核的功能，将 display 模块封装为一个 ip 核以供之后设计 alu 时候调用。接下来实现 alu 模块，然后实现顶层模块 top 模块，除了例化 alu 模块以外，还需要添加之前实现的 display IP 核调用和例化，即可完成 top 模块的实现，最后还要完成符号扩展模块 signext，然后使用仿真文件测试，并添加约束文件在开发板上测试。

2. 实验步骤

第一部分：实现 display IP 核。

根据 basys3 开发板上的数码管如下

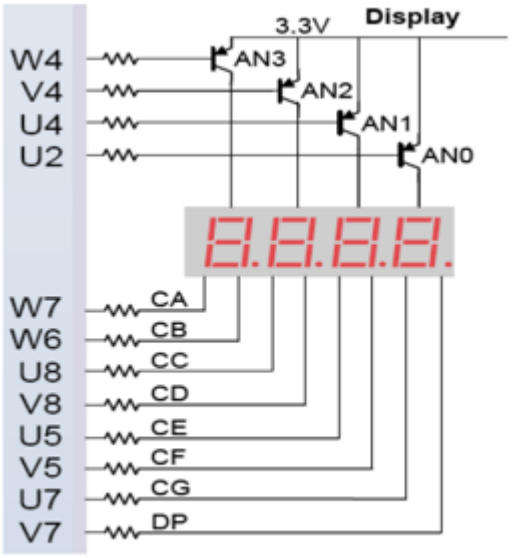

图 1-实验板上 LED 引脚连接图

为四位七段共阳极数码管，即高电平情况下为熄灭状态，低电平为点亮。所以在 display 模块中使用一个 16 位的输入 data，data 采用 BCD 码输入 4 位十六进制数字，输出给数码管的为一个 4 位的 sm_wei 信号控制是哪一位目前点亮，一个 7 位的 sm_duan 信号控制需要点亮的那一位数字具体哪一些段点亮。在具体的实现中，分为三个子部分。

1．分频模块

将原始 100MHz 的信号降低为 500Hz（注：老师的文件上标注的是 400Hz，实际上这个部分应该为 500Hz），实现部分代码如下：

```
//----------------------------------------------------//分频
integer clk_cnt;
reg clk_500Hz;
always @(posedge clk)
    if(clk_cnt==32'd100000)
    begin clk_cnt <= 1'b0; clk_500Hz <= ~clk_500Hz;
    end else clk_cnt <= clk_cnt + 1'b1;
```

该部分通过 100000 个时钟周期翻转 clk_cnt 一次，将输出的信号周期为原始时钟信号的 200000 倍，即周期为 500Hz。

2.位控制

输入为第一部分得到的 500Hz 信号，在信号的上升沿的情况下进行循环向左位移，轮流显示每一位。代码如下：

```
//-------------------------------------------------//位控制
reg [3:0] wei_ctrl=4'b1110;
always @(posedge clk_500Hz)
    wei_ctrl <= {wei_ctrl[2:0],wei_ctrl[3]}; //段控制
reg [3:0]duan_ctrl;//选择显示数据的位，4 位一组
always @(wei_ctrl)
case(wei_ctrl)
    4'b1110:duan_ctrl=data[3:0];
    4'b1101:duan_ctrl=data[7:4];
    4'b1011:duan_ctrl=data[11:8];
    4'b0111:duan_ctrl=data[15:12];
    default:duan_ctrl=4'hf;
endcase
```

wei_ctrl 循环左移，改变要显示的位，同时将 data 中对应的 BCD 码传入 duan_ctrl 中准备进行下一步的解码。

3.段控制

该部分将第二部分中的 duan_ctrlBCD 码转换为 7 位数码管的段码，代码如下：

```
reg [6:0]duan;//段码
always@(duan_ctrl)
case(duan_ctrl) 4'h0:duan=7'b100_0000;//0
    4'h1:duan=7'b111_1001;//1
    4'h2:duan=7'b010_0100;//2
    4'h3:duan=7'b011_0000;//3
    4'h4:duan=7'b001_1001;//4
    4'h5:duan=7'b001_0010;//5
    4'h6:duan=7'b000_0010;//6
    4'h7:duan=7'b111_1000;//7
    4'h8:duan=7'b000_0000;//8
    4'h9:duan=7'b001_0000;//9
    4'ha:duan=7’b000_1000;//a
    4'hb:duan=7'b000_0011;//b
    4'hc:duan=7'b100_0110;//c
```

```
    4'hd:duan=7'b010_0001;//d
    4'he:duan=7'b000_0111;//e
    4'hf:duan=7'b000_1110;//f
    default : duan = 7'b100_0000;//0
endcase
```

display 模块总结：三个部分都使用 always 语句块，这三个部分同时是并行执行的，共同组成。

第二部分：alu 模块

alu 模块包含两个输入的操作数，一个控制信号，一个输出结果。在这里我直接实现老师在第九周要求实现的带有 ZF,CF,OF,SF,PF 的 alu。

得益于 verilog 语言已经提供了+,-,|,&,<等运算符，alu 模块实现起来很容易，使用 case 语句根据控制信号直接分为 6 种情况即可。实现代码如下

```
    case(aluCtr)
        4'b0110: // 减
        begin
            {CF,aluRes} = input1 - input2;
            OF=input1[WIDTH-1]^input2[WIDTH-1]^aluRes[WIDTH-1]^CF;
        end
        4'b0010: // 加
        begin
            {CF,aluRes} = input1 + input2;
            OF=input1[WIDTH-1]^input2[WIDTH-1]^aluRes[WIDTH-1]^CF;
        end
        4'b0000: // 与
        begin
            aluRes = input1 & input2;
            OF=0;
        end
        4'b0001: // 或
        begin
            aluRes = input1 | input2;
            OF=0;
        end
        4'b1100: // 异或
        begin
            aluRes = ~(input1 | input2);
            OF=0;
```

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

CF 为加法或者减法情况下的前面一位。其中包含了对 OF 这个溢出检测的部分，使用 OF=input1[31]^input2[31]^aluRes[31]^CF 判断同号运算后的结果是否出现了异号和进位来判断是否溢出。由于仅在加减法的情况下可能会出现溢出，所以在其他运算中我们将 OF 置 0。

最后处理 ZF，SF,PF 的部分。ZF 为判断运算结果是否为 0，SF 为符号位，PF 判断运算结果中奇偶校验，其中^aluRes 这个运算代表了将 aluRes 中的每一位按位逐一进行异或，并将结果取反，容易得到当 aluRes 中有奇数个 1 的情况下，^aluRes 的结果为 1，偶数个 1，则 PF=0。（注：这里老师给的 PF = ~^ aluRes 又是个错误，应该为 PF = ^ aluRes）实现代码如下：

```
    if(aluRes==0)
        ZF=1;
    else
        ZF=0;

    SF=aluRes[31];
    PF=~^aluRes;
```

接下来还需要在 alu 中实现符号扩展的模块，用来处理有符号数字的处理，如果符号位为 1，该数字为负数，将 16 位数字拓展为 32 位需要在前面补充 1，如果符号位为 0，该数字为正数，将 16 位数字拓展为 32 位需要在前面补充 0。实现代码如下：

```
module signext(
          input[15:0] inst,
          output[31:0] data
      );
assign data = inst[15:15]?{16'hffff,inst}:{16'h0000,inst};
```

```
endmodule
```

同时在 alu 中也需要实例化符号扩展模块

```
signext signext(.inst(inst[15:0]), .data(input1));
```

alu 模块总结：alu 模块处理了两个 32 位数字的运算，由于一个输入为 16 位我们还进行了符号扩展将 16 位的操作数正确的转换为 32 位操作数。

接下来实现把 alu 模块改成可变宽度形式，加入#(parameter WIDTH=32)语句，同时将其中的 31 的部分全部替换成为 WIDTH-1，最终完整 alu 模块代码如下：

```
module alu
#(parameter WIDTH=32) (
          input [15:0] inst,
          input [WIDTH-1:0] input2,
          input [3:0] aluCtr,
          output reg[WIDTH-1:0] aluRes,
          output reg ZF,CF,OF,SF,PF
       );
wire [WIDTH-1:0]input1;
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
            {CF,aluRes} = input1 - input2;
            OF=input1[WIDTH-1]^input2[WIDTH-1]^aluRes[WIDTH-1]^CF;
        end
        4'b0010: // 加
        begin
            {CF,aluRes} = input1 + input2;
            OF=input1[WIDTH-1]^input2[WIDTH-1]^aluRes[WIDTH-1]^CF;
        end
        4'b0000: // 与
        begin
            aluRes = input1 & input2;
            OF=0;
        end
        4'b0001: // 或
        begin
```

```
            aluRes = input1 | input2;
            OF=0;
        end
        4'b1100: // 异或
        begin
            aluRes = ~(input1 | input2);
            OF=0;
        end
        4'b0111: // 小于设置
        begin
            if(input1<input2)
                aluRes = 1;
            OF=0;
        end
        default:
            aluRes = 0;
    endcase

    if(aluRes==0)
        ZF=1;
    else
        ZF=0;

    SF=aluRes[WIDTH-1];
    PF=~^aluRes;
end
signext signext(.inst(inst[15:0]), .data(input1));
endmodule
```

至此，完成了所有模块设计工作。

下面进行仿真测试。在仿真模块中，通过实例化 alu 模块进行测试。

仿真代码如下

```
module alusim;
// Inputs
reg [31:0] input2;
reg [3:0] aluCtr;
reg [15:0] inst; // 输入 16 位
// Outputs
wire [31:0] aluRes;
wire ZF,CF,OF,SF,PF;
// Instantiate the Unit Under Test (UUT)
```

```
alu uut (
.inst(inst),
.input2(input2),
.aluCtr(aluCtr),
.aluRes(aluRes),
.ZF(ZF),
.CF(CF),
.OF(OF),
.SF(SF),
.PF(PF)
);
initial begin
    $dumpfile("simalu.vcd");
    $dumpvars;
// Initialize Inputs
    $timeformat(-9, 0, " ns");
    $monitor ("%0t aluCtr=%4b inst=%16b inst=%32b aluRes=%32b ZF=%1b CF=%1b OF=%1b
 SF=%1b PF=%1b",
    $time, aluCtr,inst,inst,aluRes,ZF,CF,OF,SF,PF);
    input2 = 32'hffffffff8;
    aluCtr = 4'b0110;//减法
    inst=16'b0000_0000_0000_0001; //符号扩展的输入
    #100;
    input2 = 1;
    aluCtr = 4'b0110;//减法
    inst=16'b1000_0000_0000_0001;
    #100
    input2 = 1;
    aluCtr = 4'b0010;//加法
    #100
    input2 = 0;
    aluCtr = 4'b0000;//与
    #100
    input2 = 0;
    aluCtr = 4'b0001;//或
    #100
    input2 = 0;
    aluCtr = 4'b0111;//小于设置
    #100
    input2 = 1;
    aluCtr = 4'b0111;//小于设置
end
endmodule
```

其中，$timeformat(-9, 0, " ns");语句控制输出时间的格式，以 ns 为单位输出，不保留小

数，使用$monitor 输出当前变量的值。

生成的 RTL 图如下：

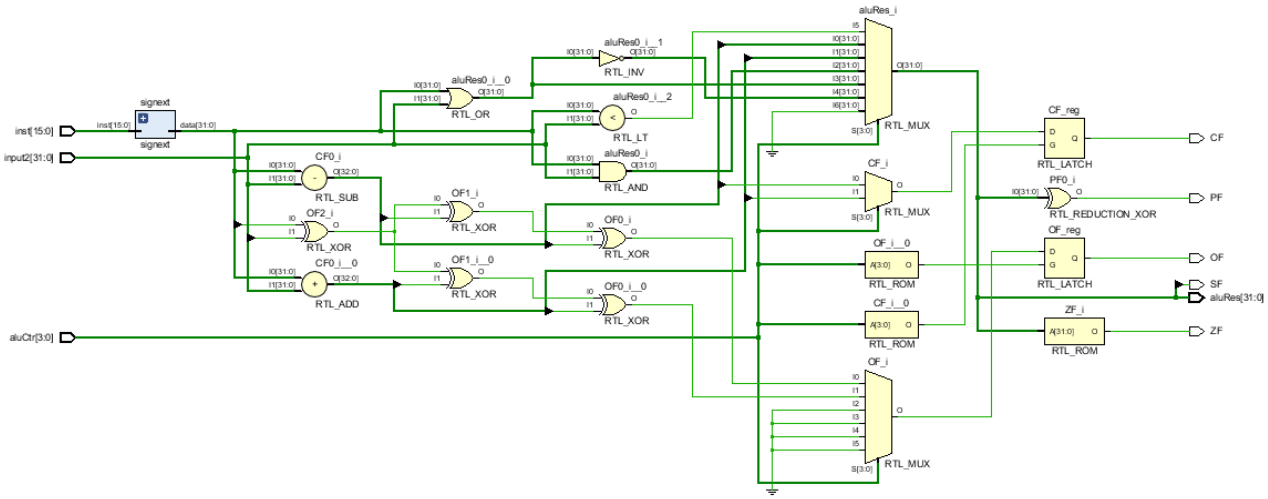

图 2-第九周 alu 模块 RTL 图

接下来按照引脚分配创建约束文件下载程序到实验板上同样用这几组数据测试，同时使用数码管测试。

实验板测试（由于实验板测试是第八周的内容，所以并没有显示标志位和符号的操作，而且两个操作数都是 16 位的）：

用 SW 模拟输入运算数据（比如 input2）以及 aluCtr 操作选择，sw0~sw7 作为 16 位 input2 的低 8 位输入，input2 的高 8 位全部接 sw8(在 top 文件中加一条语句，把高 8 位连在一起，约束文件中只连高 8 位的一位即可 input [15:0] input2, assign input2[15:8]={8{input2[8]}};)。连接数码管 an[0]-an[3]段码位选择,段码 seg[0]-seg[6].

四、实验结果

第九周中带符号扩展和标志位的如下

仿真结果如下：

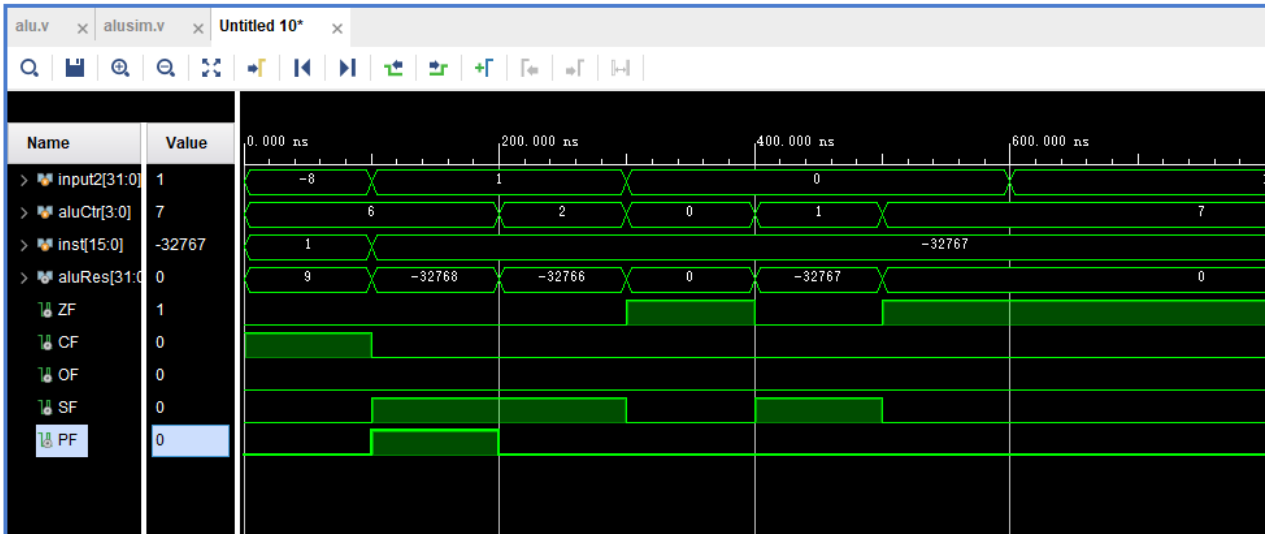

图 3-vivado 仿真波形图(有符号数表示)

vivado 输出如下：

```
# run 1000ns
0 ns aluCtr=0110 inst=0000000000000001 input2=11111111111111111111111111111000 aluRes=00000000000000000000000000001001 ZF=0 CF=1 OF=0 SF=0 PF=0
100 ns aluCtr=0110 inst=1000000000000001 input2=00000000000000000000000000000001 aluRes=11111111111111111000000000000000 ZF=0 CF=0 OF=0 SF=1 PF=1
200 ns aluCtr=0010 inst=1000000000000001 input2=00000000000000000000000000000001 aluRes=11111111111111111000000000000010 ZF=0 CF=0 OF=0 SF=1 PF=0
300 ns aluCtr=0000 inst=1000000000000001 input2=00000000000000000000000000000000 aluRes=00000000000000000000000000000000 ZF=1 CF=0 OF=0 SF=0 PF=0
400 ns aluCtr=0001 inst=1000000000000001 input2=00000000000000000000000000000000 aluRes=11111111111111111000000000000001 ZF=0 CF=0 OF=0 SF=1 PF=0
500 ns aluCtr=0111 inst=1000000000000001 input2=00000000000000000000000000000000 aluRes=00000000000000000000000000000000 ZF=1 CF=0 OF=0 SF=0 PF=0
600 ns aluCtr=0111 inst=1000000000000001 input2=00000000000000000000000000000001 aluRes=00000000000000000000000000000000 ZF=1 CF=0 OF=0 SF=0 PF=0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'alusim_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:06 . Memory (MB): peak = 2227.207 ; gain = 0.000
```

图 4-vivado 的数值输出

分析：

第一组数据：为减法，操作数 1 为 1，操作数 2 为-8，结果为 9，奇偶校验为 0，无溢出，符号为正数，计算结果与各个标志为均准确无误。

第二组数据：为减法，操作数 1 为-32767，操作数 2 为 1，结果为-32768，奇偶校验为 1，无溢出，符号为负数，计算结果与各个标志为均准确无误。

第三组数据：为加法，操作数 1 为-32767，操作数 2 为 1，结果为-32766，奇偶校验为 0，无溢出，符号为负数，计算结果与各个标志为均准确无误。

第四组数据：为与操作，操作数 1 为-32767，操作数 2 为 0，结果为 0，奇偶校验为

0，无溢出，符号为负数，零标志位为 1，计算结果与各个标志为均准确无误。

第五组数据：为或操作，操作数 1 为-32767，操作数 2 为 0，结果为-32767，奇偶校验为 0，无溢出，符号为负数，计算结果与各个标志为均准确无误。

第六组数据：为小于设置操作，操作数 1 为-32767，操作数 2 为 0，结果为-32767，奇偶校验为 0，无溢出，符号为负数，计算结果与各个标志为均准确无误。

第七组数据：为小于设置操作，操作数 1 为 32769（此时为无符号数），操作数 2 为 1，结果为 0，奇偶校验为 0，无溢出，符号为负数，计算结果与各个标志为均准确无误。

通过上述测试，alu 模块的加法，减法，与，或，小于设置均运行正常，完成了 alu 部分的设计。

实验板测试（第八周内容）

第一组：加法，input1=0x7，input2=0x10

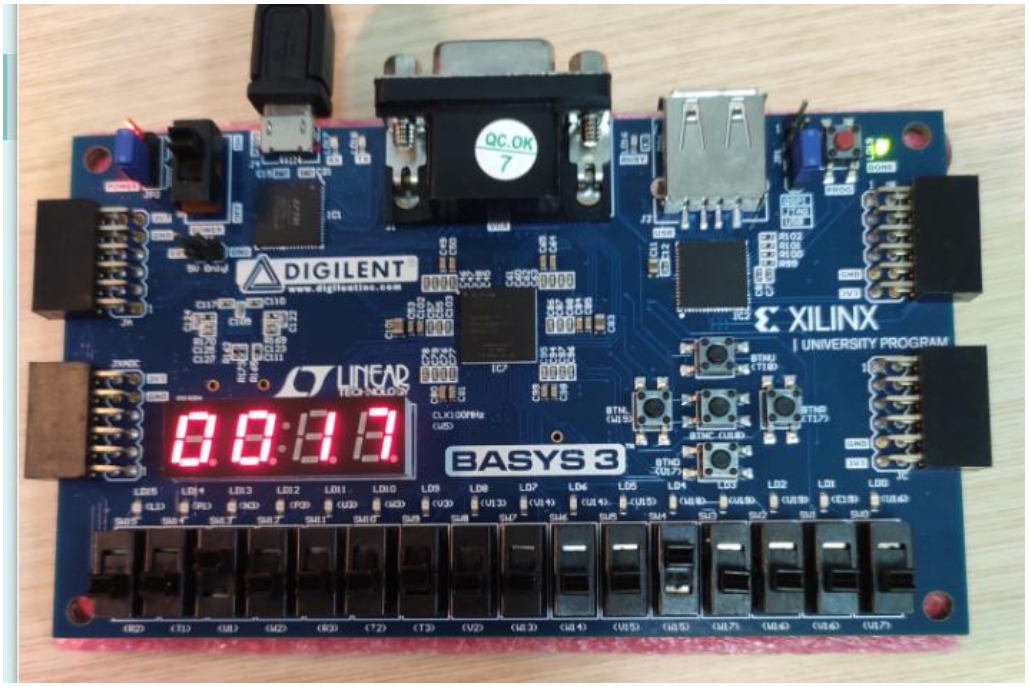

图 5-加法测试

分析：0x7+0x10=0x17,运算正确

第二组：减法，input1=0x7，input2=0x4

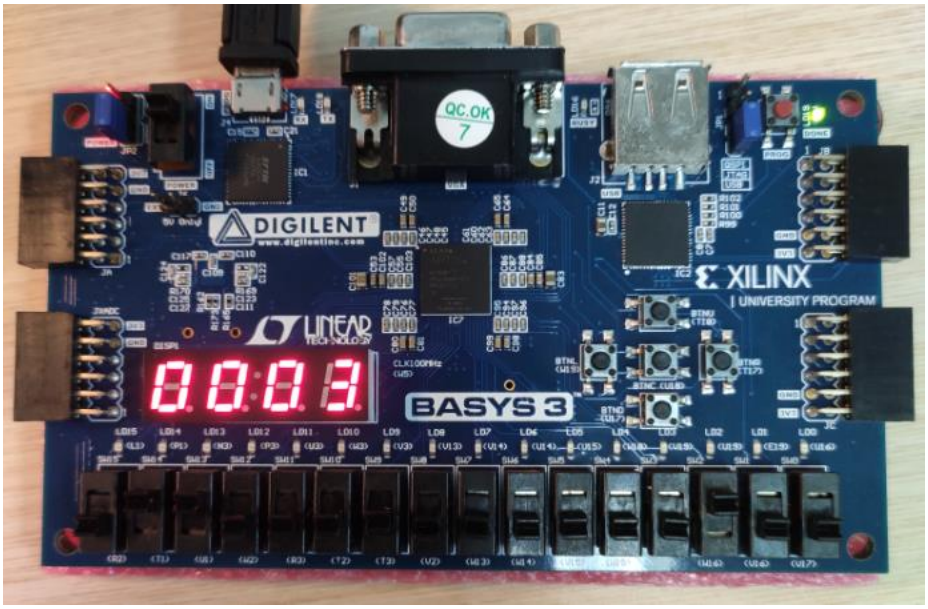

图 6-减法测试

分析：0x7-0x4=0x3,运算正确

第三组：与运算，input1=0x7，input2=0x3

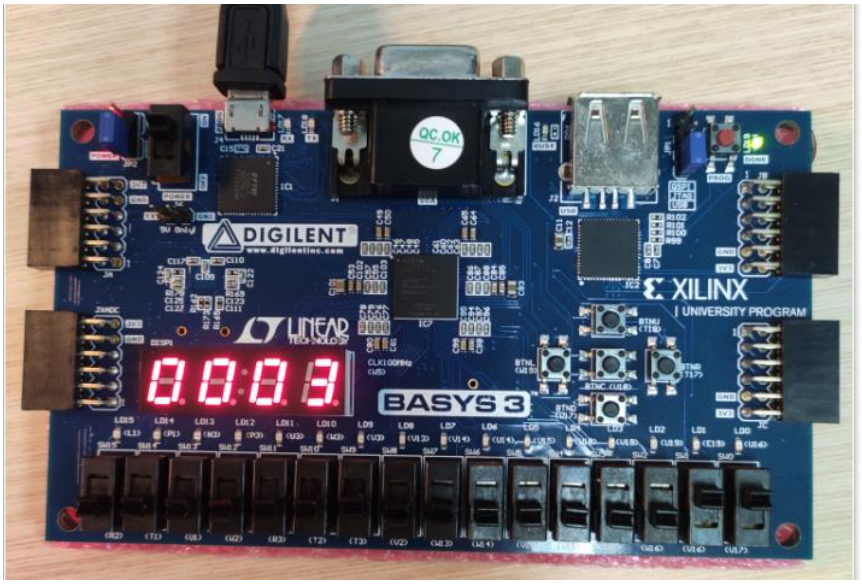

图 7-与运算测试

分析：0x7 & 0x3=0x3,运算正确

第四组：或运算，input1=0x7，input2=0x1c

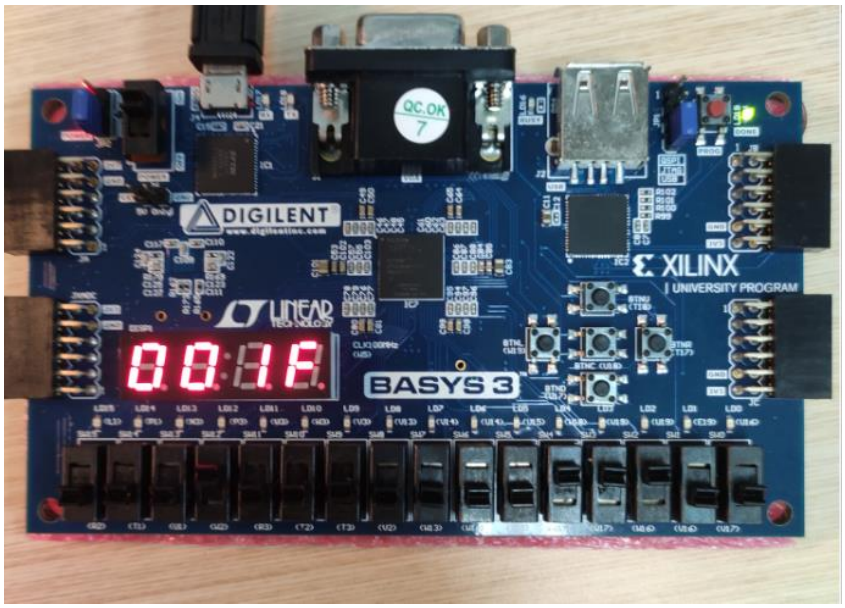

图 8-或运算测试

分析：0x7 | 0x1c=0x1f,运算正确

第一组：异或运算，input1=0x7，input2=0x0

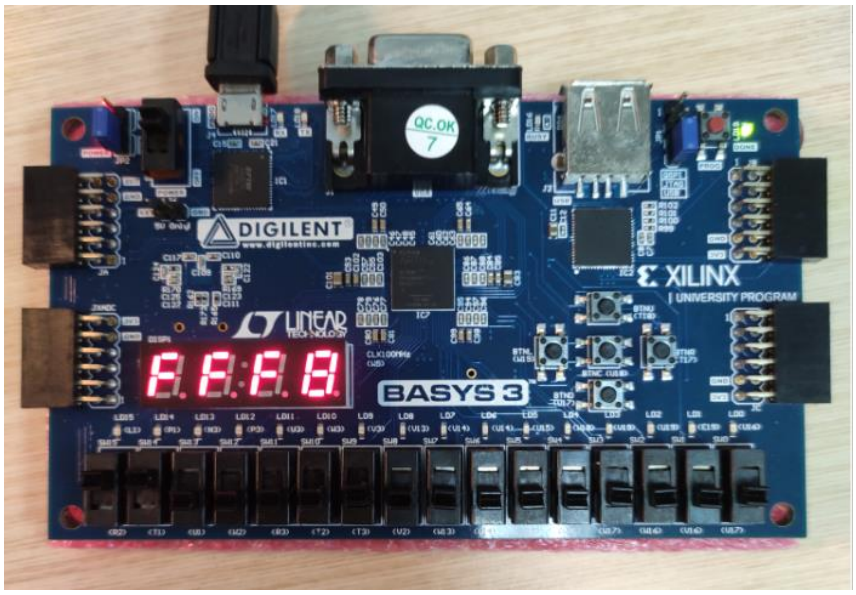

图 9-异或运算测试

分析：0x7 ^ 0x0=0xfff8,运算正确

第六组：小于设置，input1=0x7，input2=0x10

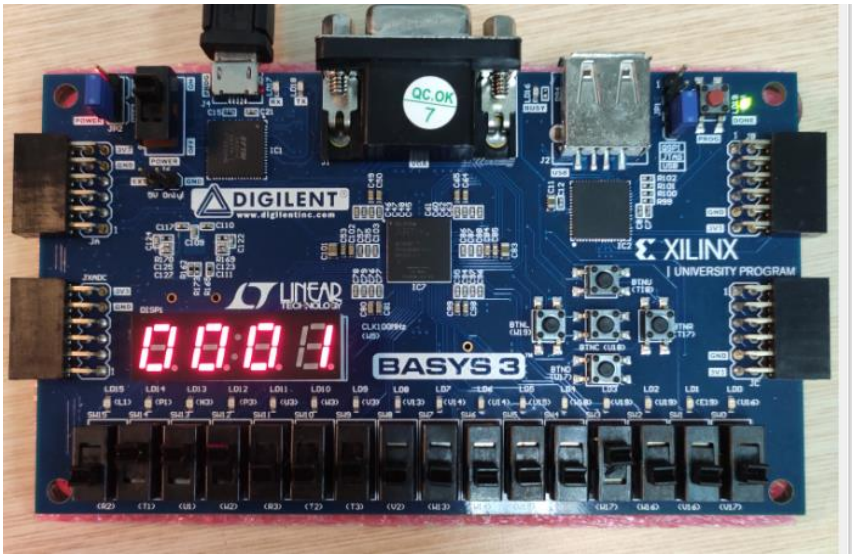

图 10-小于设置测试

分析：0x7 < 0x10,结果为 1,运算正确。

经过更多的实验，该 alu 模块的加，减，或，异或运算正确，完成了 alu 模块和数码管显示模块的设计实验。

五、实验感想

本次实验在以前的基础上，进一步完成 alu 的设计，包含了更多的标志位，同时要求自行封装数码管的 IP 核，并且使用数码管显示，在整个实现过程中，进一步理解了 verilog 语言的使用方法，例如在顶层文件中，需要连接到实验板的端口，必须写在顶层文件的输入端口中，如果在后续定义则为临时变量。同时，assign 也隐式地可以定义新变量，

在本次关于 IP 核的使用过程中，实际上仅仅将代码封装，并没有像官方提供的 IP 核那样可以自行调整一些参数，并没有发挥 IP 核的真正实力，如果要做一个通用性强的 IP 核需要做好各种参数的设置，根据使用情况可以灵活调整参数的 IP 核，更能被广泛运用。

在本次实验中，还尝试了远程连接实验板进行下载比特流。本人使用的远端机器为宿

舍内的台式机，本地机器为校园网无线网环境中的笔记本电脑。

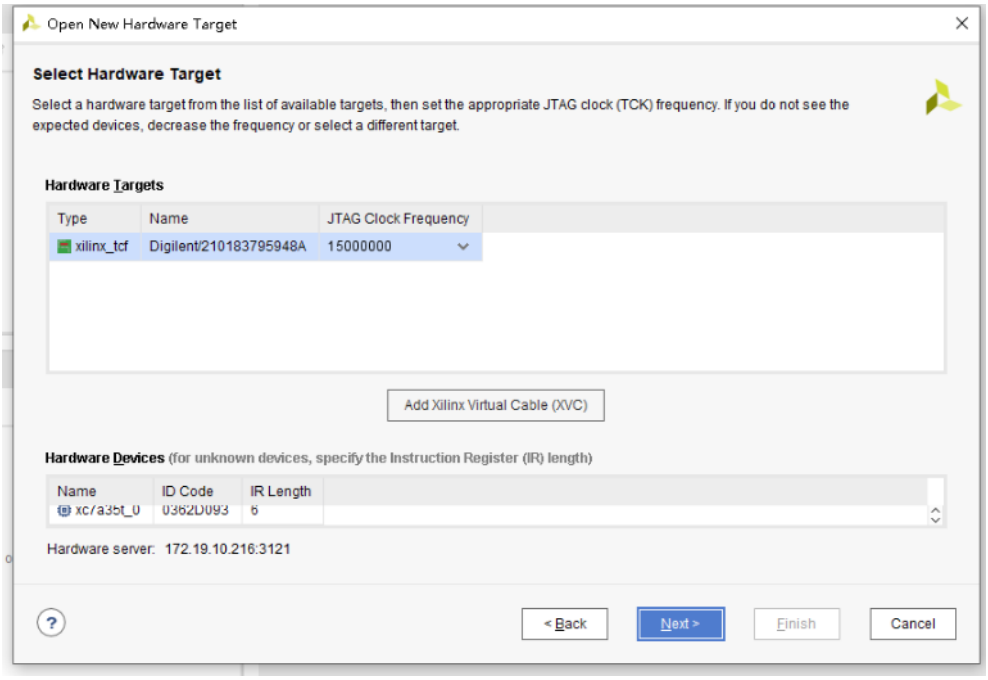

图 11-远程连接，远程端连接设置

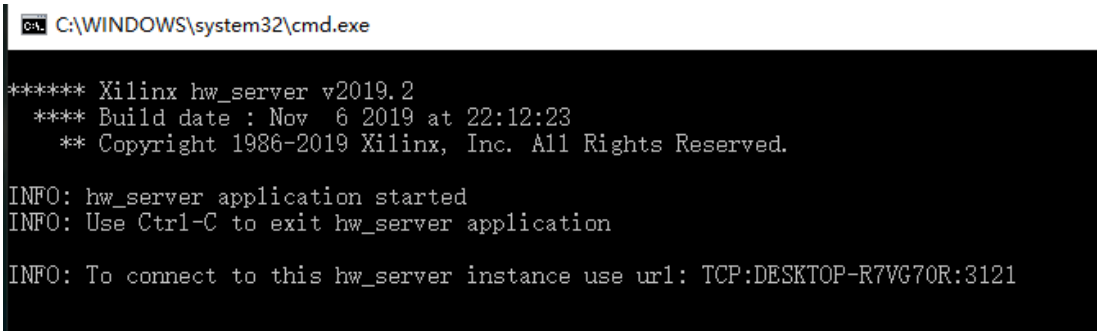

图 12-远程连接，本地端开启 hw_server

只需要在两台机器上安装 vivado，并且在本地机器上开启 bin 文件夹中的 hw_server.exe，然后就可以通过远程连接的电脑上连接本地创建的硬件服务器，就可以通过网络使用远端算力来编译文件，下载程序了。(这部分参考了 https://cnblogs.com/jiandahao/p/5734388.html)

附录（流程图，注释过的代码）：

第八周实验结构图

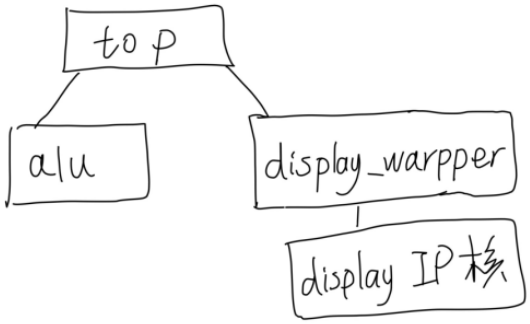

图 1-第八周测试

第九周实验结构图

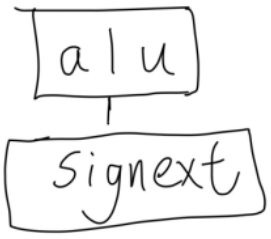

图 1-第九周测试

本次实验代码太多，均打包在压缩包内，不在此处重复。