南京大学

数电实验三：加法器和 ALU

课程名称：数字逻辑与计算机组成实验

姓名：孙文博

学号：201830210

班级：数电一班

邮箱：201830210@smail.nju.edu.cn

实验时间：2022.3.18

一、 实验目的

1. 复习数字电路中加法器的相关知识；

2. 了解机器运算中的各种溢出信号；

3. 熟悉 ALU 各种运算的过程；

4. 设计一个带有逻辑运算的简单 ALU；

二、 实验环境

设计\编译环境：Quartus (Quartus Prime 17.1) Lite Edition

开发平台：DE10-Standard

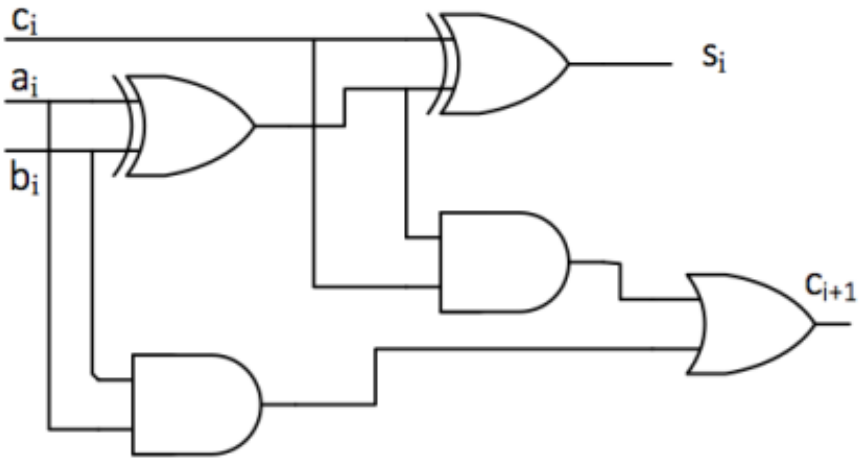
FPGA 芯片：Cyclone II 5CSXFC6D6

三、 实验原理

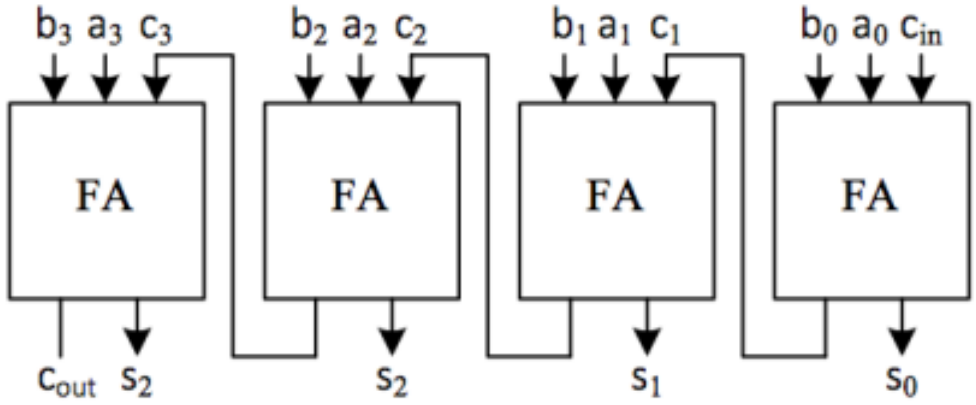
1. 加法器

两个二进制数之间的算术运算，无论是加减乘除，目前在数字计算机（数字系统）中都是化作若干次加法运算进行的。因此，加法器是构成算术运算器的基本单元，也是计算机中最“繁忙”的部件。

加法器根据是否保留进位可以分为半加器和全加器，根据计算方法可以分为串行进位加法器和先行进位加法器。我们考虑一个简单的 4 位串行加法器，它是由四个全加器串联而成的，每个全加器的逻辑也很容易理解，这里不过多阐述。



图表 1：1 位全加器电路



图表 2：四位串行全加器

2. Verilog 实现简单加减法运算器

现在我们考虑用 Verilog HDL 硬件语言模拟加减法运算器，对于指定的输入 in_x，in_y 和输出 out_s，我们可以这样计算：

$$out\_s = in\_x + in\_y;$$

$$out\_s = in\_x - in\_y;$$

但在机器运行过程中，会产生一些标志位如溢出信号 OF，进借位信号 CF，零标志 ZF，符号标志 SF，以及判断 1 的个数奇偶的 PF（i386 架构中），这里我们需要模拟实现前三个。

首先，使用 Verilog 特有的拼接语句我们可以得到进位信号 CF：

{out_c,out_s} =in_x + in_y；

这里{}表示将两个reg型变量拼接，即结果的前32位保存在out_s中，第 33 位保存在 out_c 中。但这里的进位信号仅用于表示在加法运算过程中，操作数的最高位是否对外有进位，和 X86 体系中借位 CF 的概念在减法操作中是相反的，即 X86 中的 CF = out_c ⊕ cin，其中 cin 在减法时置 1。

溢出信号 OF 则主要由两个操作数的符号和结果的符号决定，其判断原理是：如果两个参加加法运算的变量符号相同，而运算结果的符号与其不相同，则运算结果不准确，产生溢出。即两个正数相加结果为负数，或者两个负数相加结果为正数，则发生了溢出，一正一负两个数相加是不会产生溢出的。（参考 ICS 课本第二章）对应的 Verilog 语言如下：

OF = (A[3]!=B[3]) && (F[3]!=A[3]);

零标志的判断只要将结果与 0 比较即可（也可以通过按位运算得到）：

ZERO = (F == 0) ? 1 : 0;

需要注意的是，在有符号的加减法中，溢出判断依据为溢出信号，进位信号不用。而在无符号数加减法中，溢出判断依据是进位信号，溢出信号不用。

四、 实验过程

根据以上分析实现一个带有逻辑运算的 ALU 器件，功能如下表：

功能选择	功能	操作
000	加法	A+B
001	减法	A-B
010	取反	Not A
011	与	A and B
100	或	A or B
101	异或	A xor B
110	比较大小	If A<B then out=1; else out=0;
111	判断相等	If A==B then out=1; else out=0;

图表 3：ALU 功能列表

其中输入 A，B 为两个 4 位二进制数的补码表示（通过八个开关模拟），选择端 ALUstr 为 3 位二进制数，表示选择的功能（通过三个按钮模拟），输出结果 F 为 4 位二进制数，并通过七段数码管显示。进行加减运算时，能够返回标志位 ZF，OF，CF（通过二极管显示）以判断结果是否为 0，是否溢出，是否有进位等；进行逻辑运算时，不需要考虑溢出和进位（即 CF=OF=0）；比较大小时默认作为有符号数比较。对应的 Verilog 代码如下：

```
//4位带符号补码ALU
module exp_3(
    input [3:0] A,
    input [3:0] B,
    input [2:0] ALUstr,
    output [3:0] F,
    output [6:0] LED,
    output CF,
    output ZERO,
    output OF
);
```

图表 4：输入端和输出端

各个功能的具体实现代码如下：

```
always @(*) begin
    case(ALUstr)
        0: // 加法
        begin
            {cf,f}=A+B;
            zero=(f==0)?1:0;
            of=(A[3]==B[3]) && (F[3]!=A[3]);
        end
        1: //减法
        begin
            {cf,f}=A-B;
            zero=(f==0)?1:0;
            of=(A[3]!=B[3]) && (F[3]!=A[3]);
        end
        2: //取反
        begin
            f=~A;
            zero=(f==0)?1:0;of=0;cf=0;
        end
        3: //与
        begin
            f=A&B;
            zero=(f==0)?1:0;of=0;cf=0;
        end
        4: //或
        begin
            f=A|B;
            zero=(f==0)?1:0;of=0;cf=0;
        end
        5: //异或
        begin
            f=A^B;
            zero=(f==0)?1:0;of=0;cf=0;
        end

         6: //比较大小
         begin
            if(A[3]==B[3])
               f=(A>B)?1:0;
            else if(A[3]==1) f=0;
            else f=1;
            zero=(f==0)?1:0;of=0;cf=0;
         end
         7: //判断相等
         begin
            f=((A-B)==0)?1:0;
            zero=(f==0)?1:0;of=0;cf=0;
         end
         default: begin
            f=0;zero=0;of=0;cf=0;
         end
     endcase
```

图表 5：ALU 各功能代码

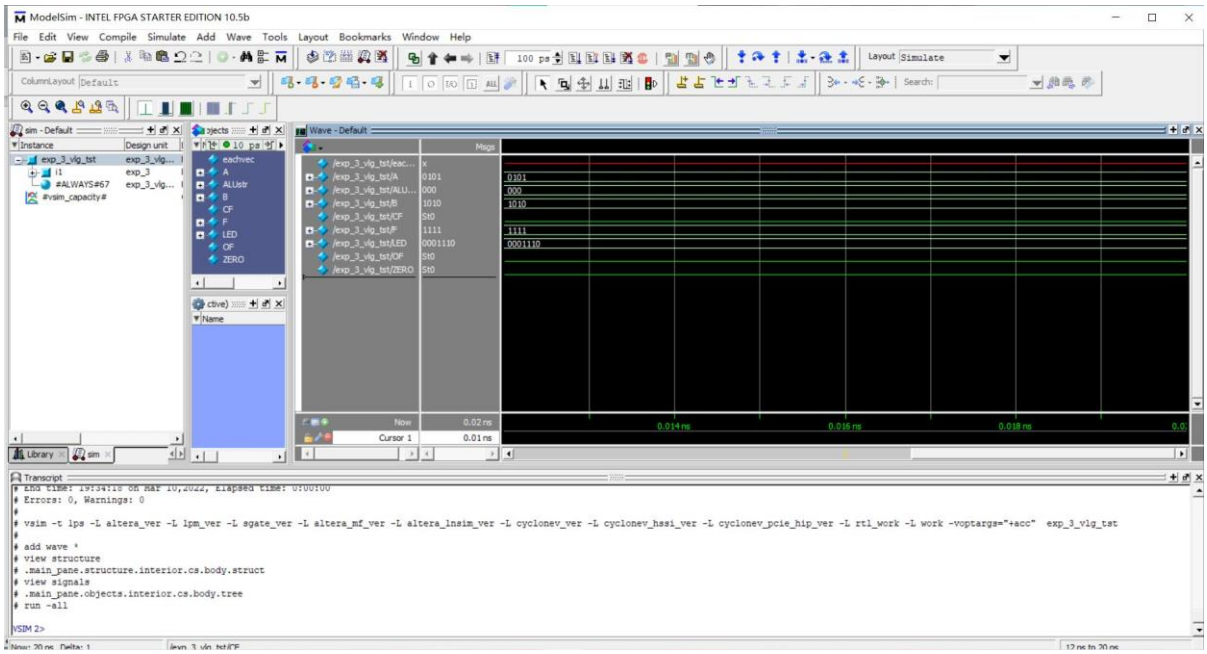
需要注意的是，比较大小时先判断两个数的正负，再进行绝对值比较，其余要点均已在上述分析中提及，不再赘述。

最后将结果表示为七段数码管形式：

```
case(f)
    0: led=7'b1000000;
    1: led=7'b1111001;
    2: led=7'b0100100;
    3: led=7'b0110000;
    4: led=7'b0011001;
    5: led=7'b0010010;
    6: led=7'b0000010;
    7: led=7'b1111000;
    8: led=7'b0000000;
    9: led=7'b0010000;
    10: led=7'b0001000;
    11: led=7'b0000011;
    12: led=7'b1000110;
    13: led=7'b0100001;
    14: led=7'b0000110;
    15: led=7'b0001110;
endcase
```

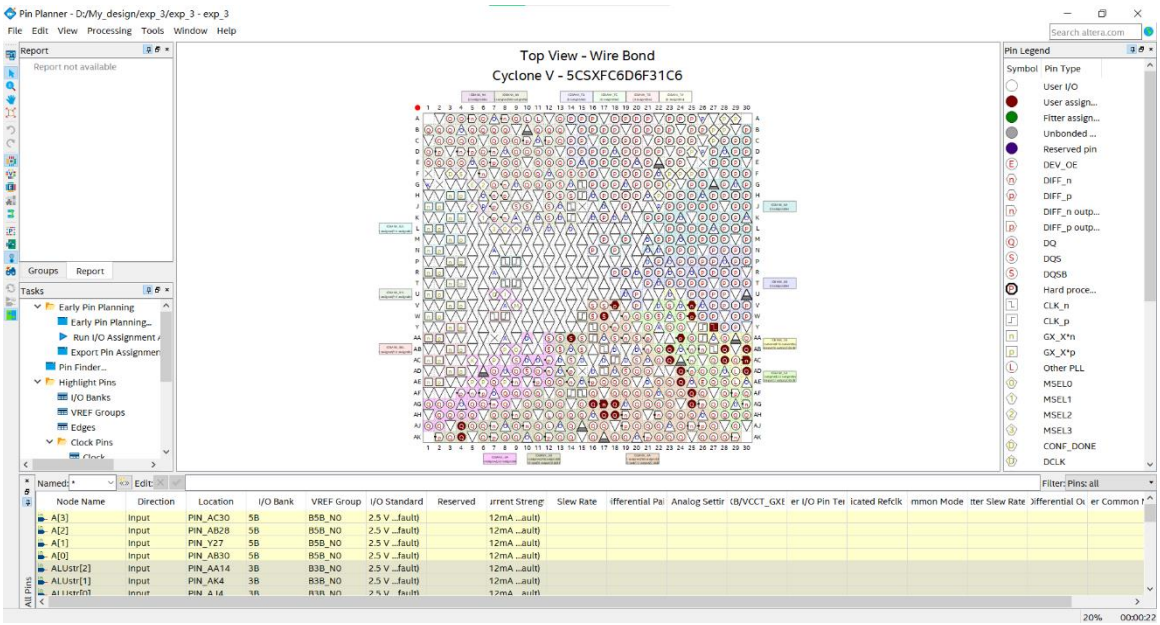
图表 6：七段数码管表示结果

代码部分结束，通过 testbench 仿真模拟如下，符合预期：



图表 7：testbench 仿真模拟

接下来配置引脚，将文件烧录到芯片中，等待上板验证。



图表 8：芯片配置

五、 实验结果

1. 思考题

考虑到最小负数，我们应先加入进位 Cin，这里方法二是正确的。

> ☞ **减法 Overflow**
>
> 虽然减法也是利用加法器实现的，但减法运算在减数是最小负数时溢出判断需要特殊处理。考虑以下两种实现，那一种是正确的?
>
> 方法一:

```
assign t_no_Cin = {n{ Cin }}^B ;
assign {Carry,Result} = A + t_no_Cin + Cin;
assign Overflow = (A[n-1] == t_no_Cin[n-1]) && (Result [n-1] != A[n-1]);
```

> 方法二:

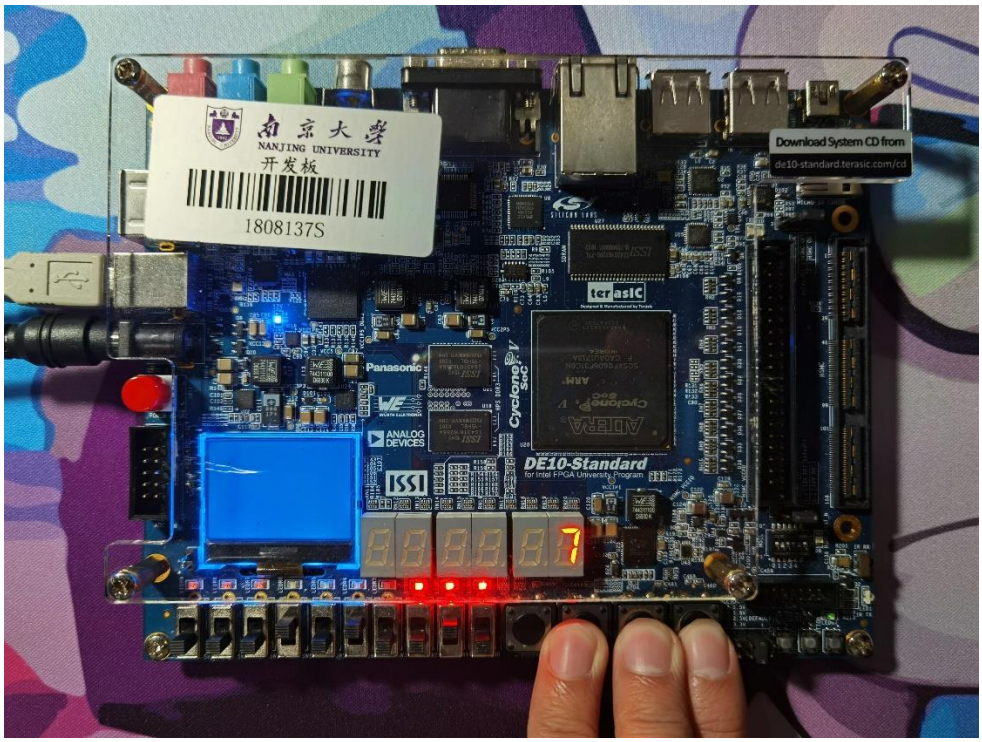
```
assign t_add_Cin =( {n{Cin}}^B )+ Cin ;  //  在这里请注意^运算和+运算的顺序
assign { Carry, Result } = A + t_add_Cin;
assign Overflow = (A[n-1] == t_add_Cin[n-1]) && (Result [n-1] != A[n-1]);
```

2. 上板验收

ALU 部分操作模拟截图如下所示，完整版视频放在附件中。

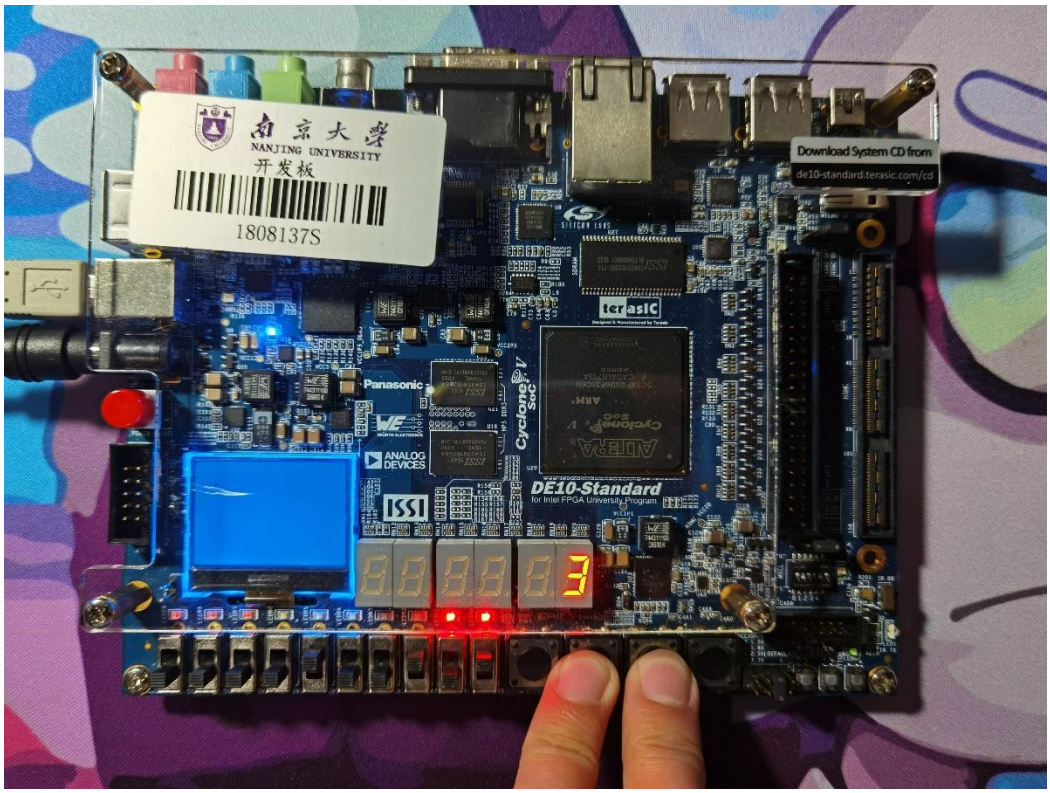
首先，我们取操作数 A 为 2（0010），操作数 B 为 5（0101），操作码取 000（按下去为 0，弹开为 1），得到的结果为 7（0111），三个

标志位（对应第 4，5，6 个 LED 灯）依次为 ZF=0，CF=0，OF=0。



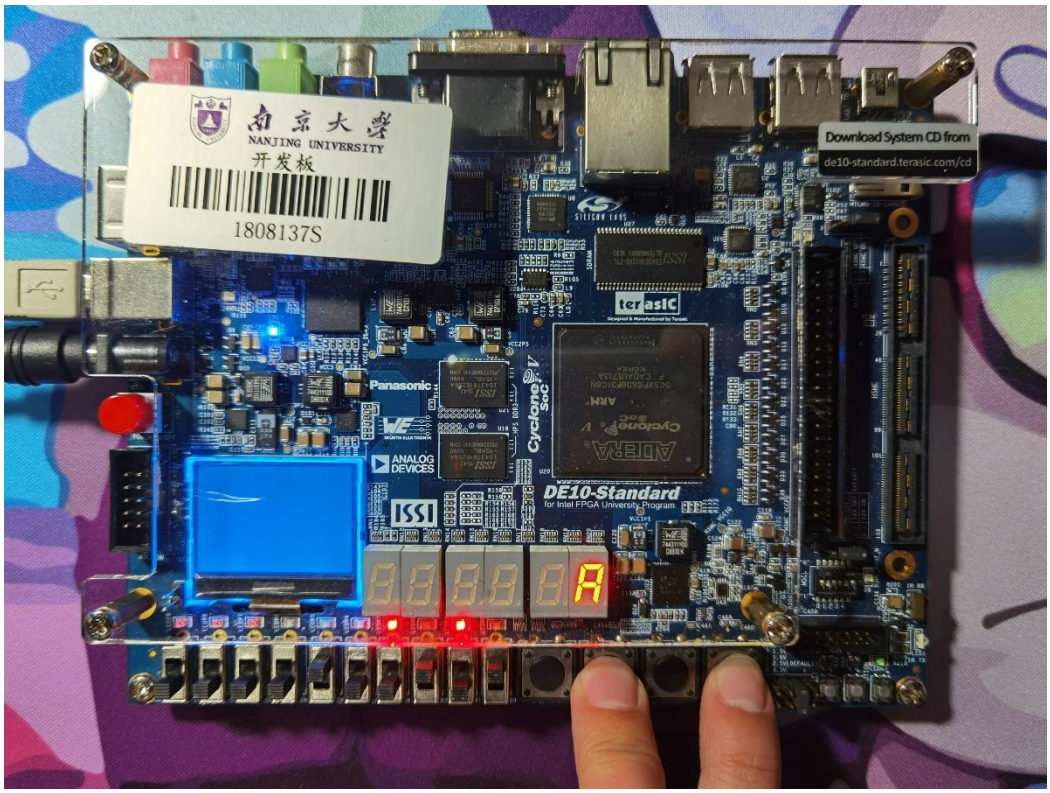
图表 9：ALU 加法

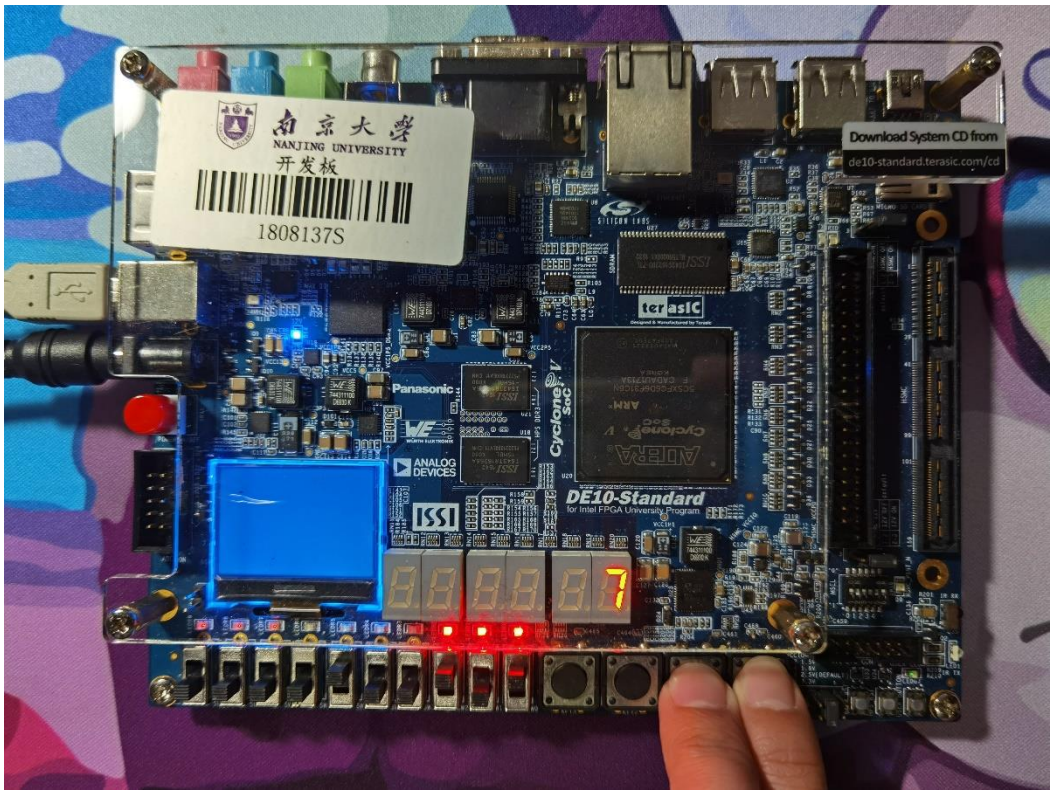
接着操作码取 001 做减法，得到的结果为 3（0110）：

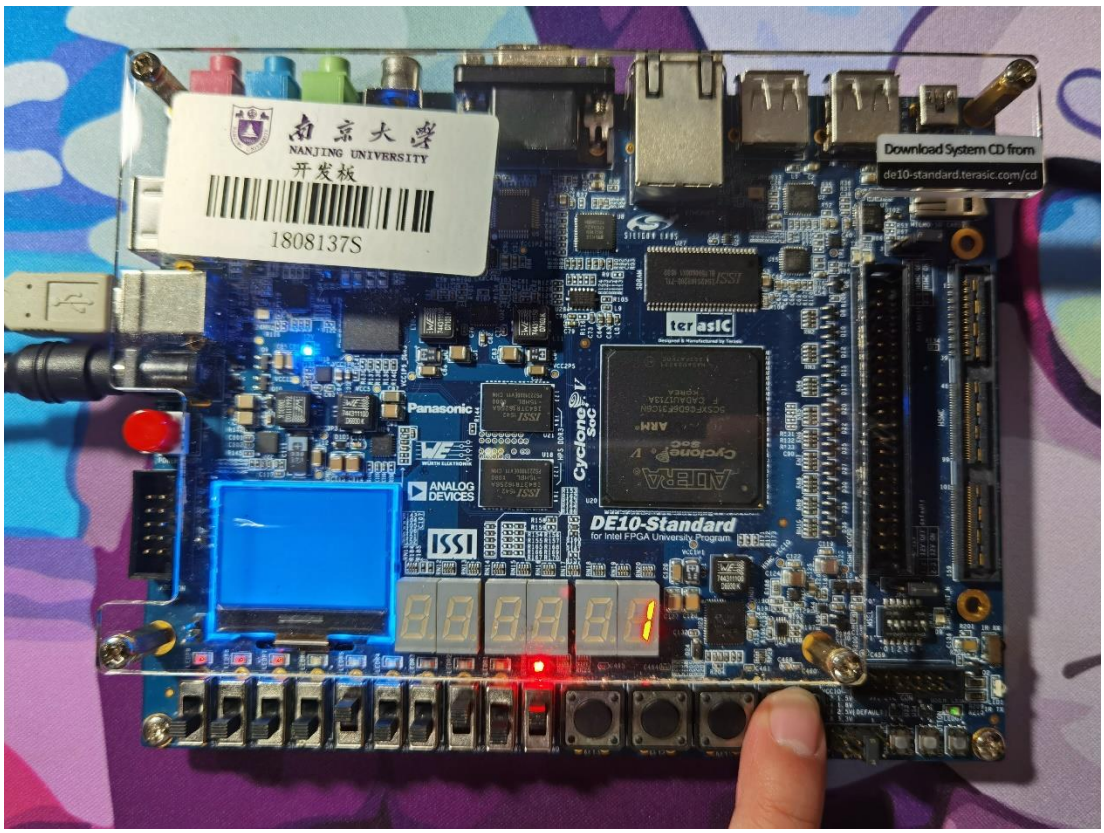


图表 10：ALU 减法

接着分别做取反（010），或（100），比较大小（110）操作，得到对应的结果为 1010（A），7（0111），1（0001）。







六、 总结与反思

本次实验中我们复习了数电及计算机组成原理中数据的存储表示和运算相关功能，实现了一个可以进行多种操作的四位补码简单 ALU。其中比较难的部分是加减法运算中各个标志位的取值方法，不过结合 PA 实验的相关内容最终还是理解并成功实现了，希望接下来的实验可以再接再厉！💪