《数字电路与数字系统实验》实验报告

第 4 次实验：　　加法器与 ALU　　

姓名：　张逸凯　

学号：　171840708　

院系:　物理　学院

邮箱：　645064582@qq.com　

电话：　　18051988316　　

实验时间:　2019 年 3 月 24 日　

O. 预习部分

加法器

多位加法器可以由一位加法器级联而成，图 3-1(a)是一位全加器真值表，输入为 a_i、b_i 和 c_i，输出为 s_i 和 c_{i+1}；图 3-1(b)是一位加法器电路图，图3 1(d)是四位行波进位加法器框图。输入为 a（a_0-a_3）、b（b_0-b_3）和 c_{in}，输出为 s（s_0-s_3）和 c_{out}；

一位全加器的设计相对简单，请同学们根据电路图自行思考如何设计一个串行进位加法器电路。串行加法器速度很慢，因为进位必须从最低位传至最高位。要想构建速度较快的加法器，就要利用附加逻辑，提前算出进位信息，这就是先行进位加法器的设计思想，先行进位加法有几种常用的算法，感兴趣的同学可以查找资料自行学习。

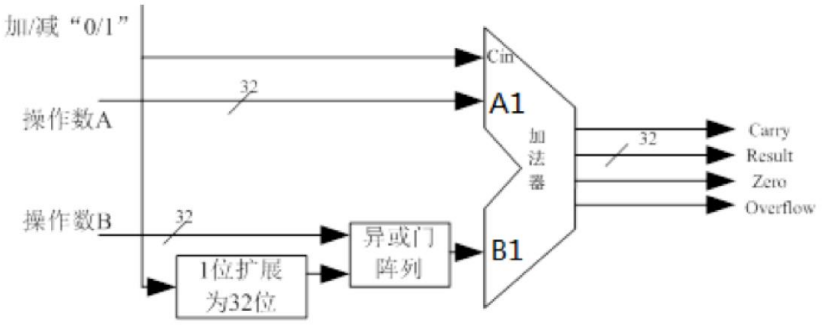


图 3-2: 简单加减 ALU

通过预习了解了本次实验之一是要求我们设计一个***具有加减法运算功能的 ALU,*** 复习之前学习的知识可以知道加减法都是在一个运算器上实现的，做加法的时候 Cin 为 0， 两个数直接相加， 做减法的时候 Cin 为 1， ***被减数取反， 当成加法来做， 结果是一样的.*** 同时还要输出高位的一位进位和一位表示溢出与否的溢出位以及类似 ZF 的判断结果是否为 0 的一位输出. 同样的通过复习的知识我们可以得出溢出位的表示， 以及判断结果是否为 0 的表示.

首先可以根据 pdf 中的表达式：

```
input [n-1:0] in_x, in_y;
output [n-1:0] out_s;
//算术赋值语句
out_s = in_x + in_y;
```

就可以实现 n 位加法器了。

通过这样来获得加法的结果以及进位的结果，当是做减法的时候我们将减数取法再加上 Cin 即可当成加法来做，同样可以获得高位的进位和相加的结果.

在位运算层面，取负就是 ~A + 1 即可，下面给出一些位运算技巧:

- `x | (~x)` 最高位一定是 1
- `~!x + 1` 在 `x == 0` 和 `x != 0` **的情况下可以化为很特殊的形式**，分别是 `0xffffffff` 和 `0x00..00`，**想消除掉某个数，`& / |` 上全零或全 1 是很好用的**

溢出判断

同时利用这个结果就可以判断两个数相加是否溢出了，这也是利用了补码

$$Overflow = (in\_x_{n-1} == in\_y_{n-1})\&\&(out\_s_{n-1}! = in\_x_{n-1}) \qquad (3\text{-}1)$$

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。当然，还有其他的判断溢出位的方式，请大家参照相关资料，了解其他判断运算是否溢出的方法。

相加的结论：如果两个加数一正一负那么结果不会溢出，***只有两个正数相加结果为负数，两个负数相加结果为正数才会溢出***，这个表达式蕴含的也是这个原理.

➢ 思考题

☞ 思考题：

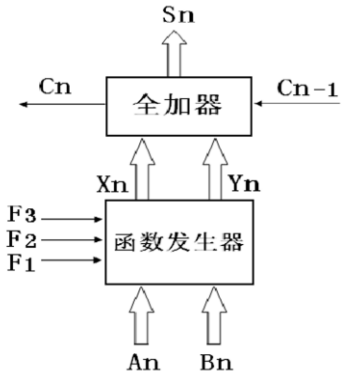
在判断输出结果是否为零的时候也有两种判断方式，一种是用 if 语句，将 Result 和 “0” 相比较，这样在硬件上会产生一个比较器。还可以使用如下语句：

```
1 assign zero = ~(| Result);
```

“| Result” 操作称为一元约简运算，这个运算在硬件上几个逻辑门就可以实现了，请查阅 Verilog 相关语法资料，了解此运算的操作过程。选择你认为好的方式来进行结果是否为 “0” 的判断。

判断结果是否为 0 正如上述所述有两种方法，***第一种***是利用 if 语句来进行判断结果是否为 0，这是最容易想到的方法，但是代码看起来不那么简洁，而且在硬件上实现起来也不方便.而***第二种方法***是利用一元约简运算，这个运算符是利用该数字的***最高位和次高位相与***，结果***再和次高位下面的一位相与***得到一个结果，***依次这样进行下去直到和最后一位相与产生的结果就是运算的结果***.这样可以和容易判断一个数是否为 0，而且硬件实现起来也比较简单，代码看起来也很舒服，第二种方法更好.

➢ ALU 的原理



实际上 ALU 的设计是在全加器的基础上，对全加器功能的扩展来实现符合要求的多种算术运算以及逻辑运算的功能.***为了实现多种功能上面的 An 和 Bn 是不能直接连到全加器上的，他们的功能必须受到变量 F3~F1 的控制.***

因此可由 An、Bn 数据和控制变量 F3~F1 组合成新的函数 Xn、Yn，然后再将 Xn 和 Yn ***和下一位进位 Cn-1*** 通过全加器以实现所需的运算功能.

- **关于任务功能的使用：**

```
task check;  //测试任务
  input [4:0] results;  //ALU的结果预期正确输出
  input resultof, resultc, resultz;  //ALU的预期溢出，进位，零位
  begin
    if(outputs!=results)  //比较预期结果和测试单元输出的outputs
      begin                    //出错时显示
        $display("Error:x=%h,y=%h,ctrl=%b,s should be %h,get  %h",
          inputa, inputb, inputaluop, results, outputs);
      end
      //自行添加溢出，进位和零位的比较
  end
endtask
```

在测试时，可以调用 check 任务来判断结果是否正确。

```
for(i=-8;i<=7;i=i+1)
  for (j=-8;j<=7;j=j+1)
    begin
       inputa=i;
       inputb=j;  //设置两个输入
       inputaluop=4'b0000; //ALU的操作码
       k = * ;    //此处自行计算正确的输出，填入*处
       of= * ;    //可分不同情况手工填写
       z = * ;
       c = * ;
       #20 check(k[3:0],of,c,z);
    end
```

在预习阶段经过查阅后发现***其实这和函数一样***，写一个任务功能函数，在测试代码中调用它，然后如果输出和自己算的结果不同，再利用系统调用来输出提示，所以在测试阶段可以用这种方法来***测试一些边界值***从而减少错误的发生.

- **关于先行进位法:**

进位的传递

$$ci+1 = ai \times bi + ai \times ci + bi \times ci$$
$$= ai \times bi + (ai + bi) \times ci$$
$$= gi + pi \times ci$$

$gi = ai \times bi$称为进位生成因子, 只要gi为1, 就有进位

$pi = ai + bi$称为进位传递因子, 只要pi为1, 就有把低位的进位向前传递

四位进位传递为例

$$c1 = g0 + (p0 \times c0)$$
$$c2 = g1 + (p1 \times g0) + (p1 \times p0 \times c0)$$
$$c3 = g2 + (p2 \times g1) + (p2 \times p1 \times g0) + (p2 \times p1 \times p0 \times c0)$$
$$c4 = g3 + (p3 \times g2) + (p3 \times p2 \times g1) + (p3 \times p2 \times p1 \times g0) + (p3 \times p2 \times p1 \times p0 \times c0)$$

只要低位有一个进位生成, 而且被传递, 则进位输出为1.

开始实验部分

一. 实验目的

理解加法器以及 ALU 的实现原理，并且掌握底层 ALU 是如何工作的，标志位是如何设置的，并且能够上网查阅资料掌握一元约简运算和熟练地进行加法器的测试，熟练运用七段数码管，最后要能用 verilog 语言实现一个简单的逻辑 ALU，数据 A、B 均为 7 位有符号输入，并将结果显示在七段数码管.

表 3-1 ALU 功能列表

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;

二. 实验原理（ 知识背景，结合理论课总结 ）

➢ 基本定义：

前面预习报告已经讲得很清楚了，这里再补充一点

ALU 必须与数字电路的其他部分使用同样的格式来进行数字处理.对现代处理器而言，*数值一律使用二进制补码表示*.早期的计算机曾使用过很多种数字系统，包括反码、符号数值码，甚至是十进制码，每一位用十个管子. 以上这每一种数字系统所对应的 ALU 都有不同的设计，而这也影响了当前对二进制补码的优先选择，因为*二进制补码能简化 ALU 加法和减法的运算*. 一个简单的能进行与或非和加运算的 2 位 ALU.

ALU 的输入是要进行操作的数据（称为操作数）以及来自控制单元的指令代码，用来指示进行哪种运算.它的输出即为运算结果. 在许多设计中 ALU 也接收或发出输入或输出条件代码到（或来自）状态寄存器.这些代码用来指示一些情况，比如进位或借位、溢出、除数为零等

三. 实验设备环境

硬件器材：FPGA 开发板.
软件平台：Qaurtus 开发平台.

四. 实验步骤 / 过程（设计思路、设计代码、测试代码、仿真结果和硬件实现等的截图代码等）

➢ 设计思路:

基于 PPT 例子（ 这次实验的例子是真的少哇 ）

```
input [n-1:0] in_x, in_y;
output [n-1:0] out_s;
//算术赋值语句
out_s = in_x + in_y;
```

就可以实现 n 位加法器了。

$$Overflow = (in\_x_{n-1} == in\_y_{n-1})\&\&(out\_s_{n-1}! = in\_x_{n-1})$$

利用上述预习过程中的结果也可以比较容易地实现加减法的功能，其他那么多功能，***可以利用case语句可以很清晰的实现，***所以现在的思路就是将上述预习结果运用起来，利用case语句实现ALU.至于各种符号位的设置，***对于操作数的每位提取出来一一判断，比如我们要把负数区分开来，***并用一个out变量来表示这个数是负数，可以用下面的代码:

```
if (result[6] == 1)

    result = ((~result) + 1);
```

其他也没有什么困难的，需求怎么来怎么实现就可以了.

难就难在开关不够了...(手动滑稽)，所以我们需要用下面的代码来绑定 A, B 变量

```
assign A = (asel == 0) ? t : A;

assign B = (bsel == 0) ? t : B;
```

设计代码:

```
module ALU(choose, t, asel, bsel, out, carry, overflow, ZF, Aeg, Beg, Ceg, Deg, Eeg, Feg);
   input [2:0] choose;
   input [6:0] t;
   input asel, bsel;
   wire [6:0] B, A;
   assign A = (asel == 0) ? t : A;
   assign B = (bsel == 0) ? t : B;
   output reg out, carry, overflow, ZF;
   output reg [6:0] Aeg;
   output reg [6:0] Beg;
   output reg [6:0] Ceg;
   output reg [6:0] Deg;
   output reg [6:0] Eeg;
   output reg [6:0] Feg;
   integer i, a, b;
   reg [6:0] c;
   reg [6:0] result;
   always @(choose or A or B or t)
   case(choose)
      0: begin { carry, result }=A + B;
                out = 0;
                overflow = (A[6] == B[6] && A[6] != result[6]);
                ZF = !(|result);
```

```
    2: begin for(i = 0; i <= 6; i = i + 1) result[i] = ~A[i];out = 0; carry = 0; overflow = 0; ZF = !(|result);
    if(result[6] == 1)
          begin
          result = ((~result) + 1);
          out = 1;
          end
    case(result % 10)
```

```
    3: begin for(i = 0; i <= 6; i = i + 1) result[i] = A[i] & B[i]; out = 0; carry = 0; overflow = 0; ZF = !(|resul
if(result[6] == 1)
          begin
          result = ((~result) + 1);
          out = 1;
          end
```

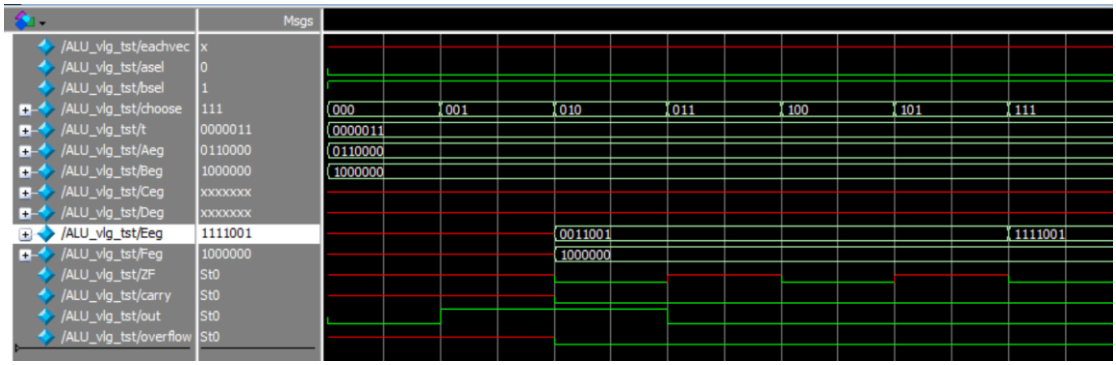
```
    4: begin for(i = 0; i <= 6; i = i + 1) result[i] = A[i] | B[i]; out = 0; carry = 0; overflow = 0; ZF = !(|resul
    if(result[6] == 1)
          begin
          result = ((~result) + 1);
          out = 1;
          end
      5: begin for(i = 0; i <= 6; i = i + 1) result[i] = A[i] ^ B[i]; out = 0; carry = 0; overflow = 0; ZF = !(|resul
      if(result[6] == 1)
            begin
            result = ((~result) + 1);
            out = 1;
            end
```

激励代码:

```
begin
// code that executes only once
// insert code here --> begin
asel = 1; t = 7'b1000001; asel = 0; bsel = 1; t =7'b0000011; choose=3'b000; #10;
                                  choose=3'b001; #10;
                                  choose=3'b010; #10;
                                  choose=3'b011; #10;
                                  choose=3'b100; #10;
                                  choose=3'b101; #10;
                                  choose=3'b111; #10;


// --> end
$display("Running testbench");
end
```

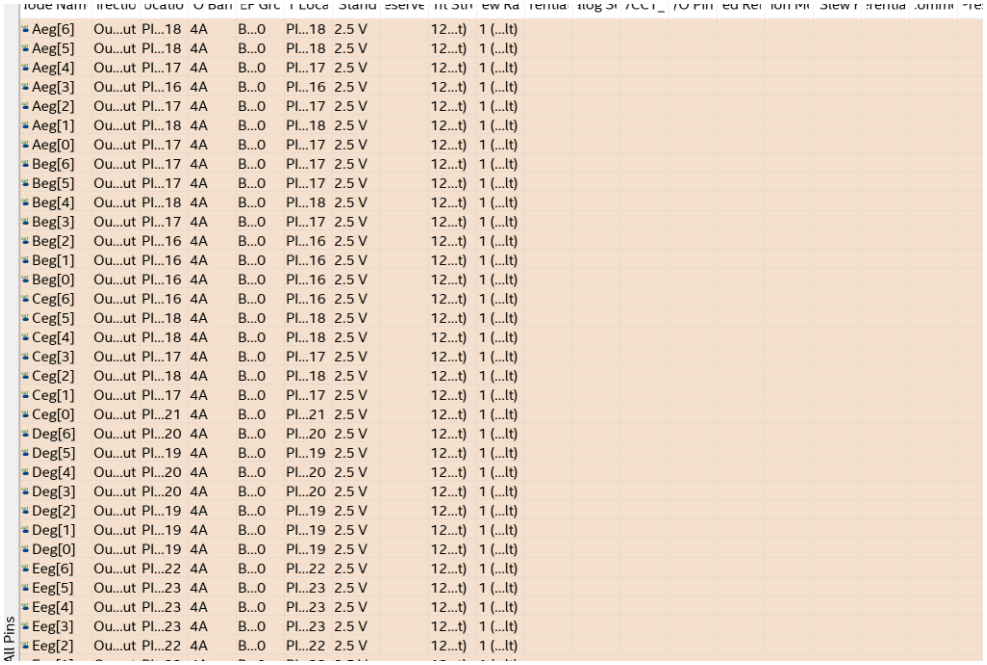
ModelSim 仿真波形:



没有测到所有情况，所以有的值没有被赋到，所以出现了红线.

➢ 硬件实现:

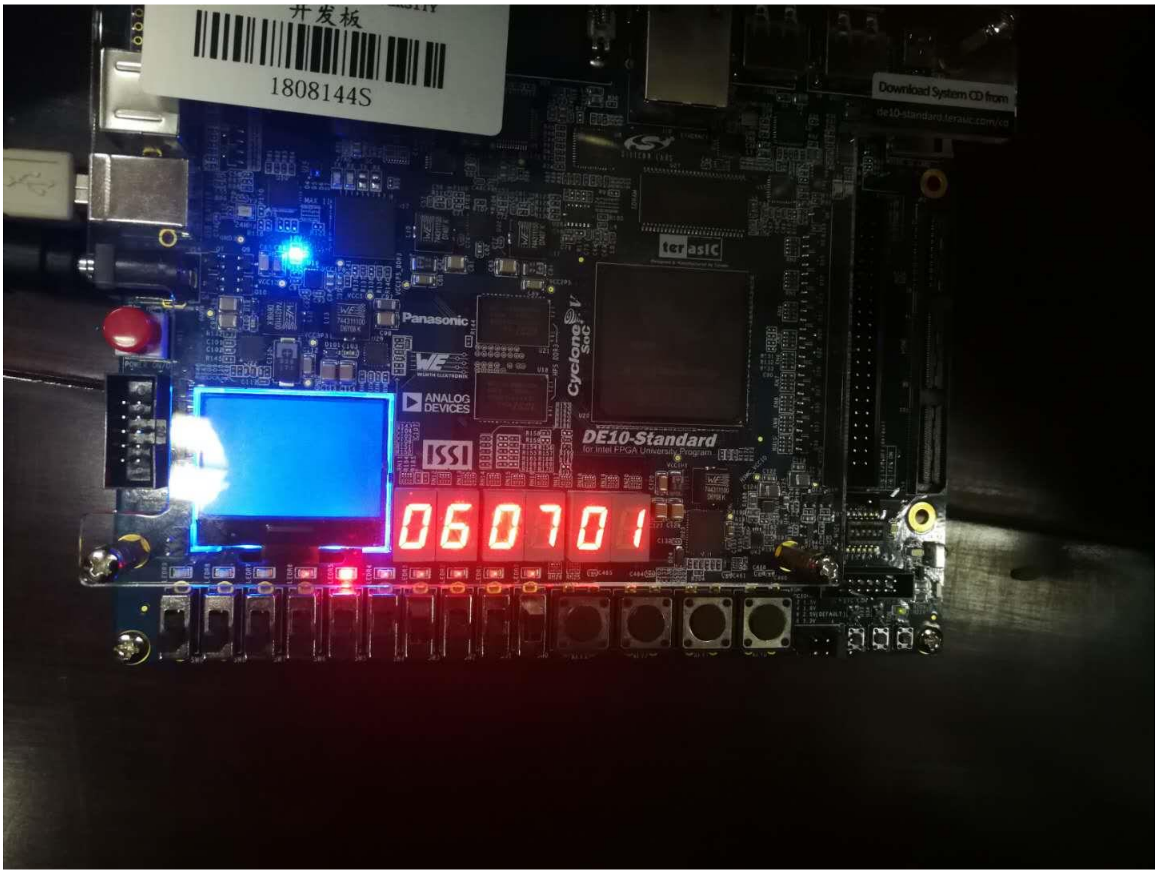
引脚分配:



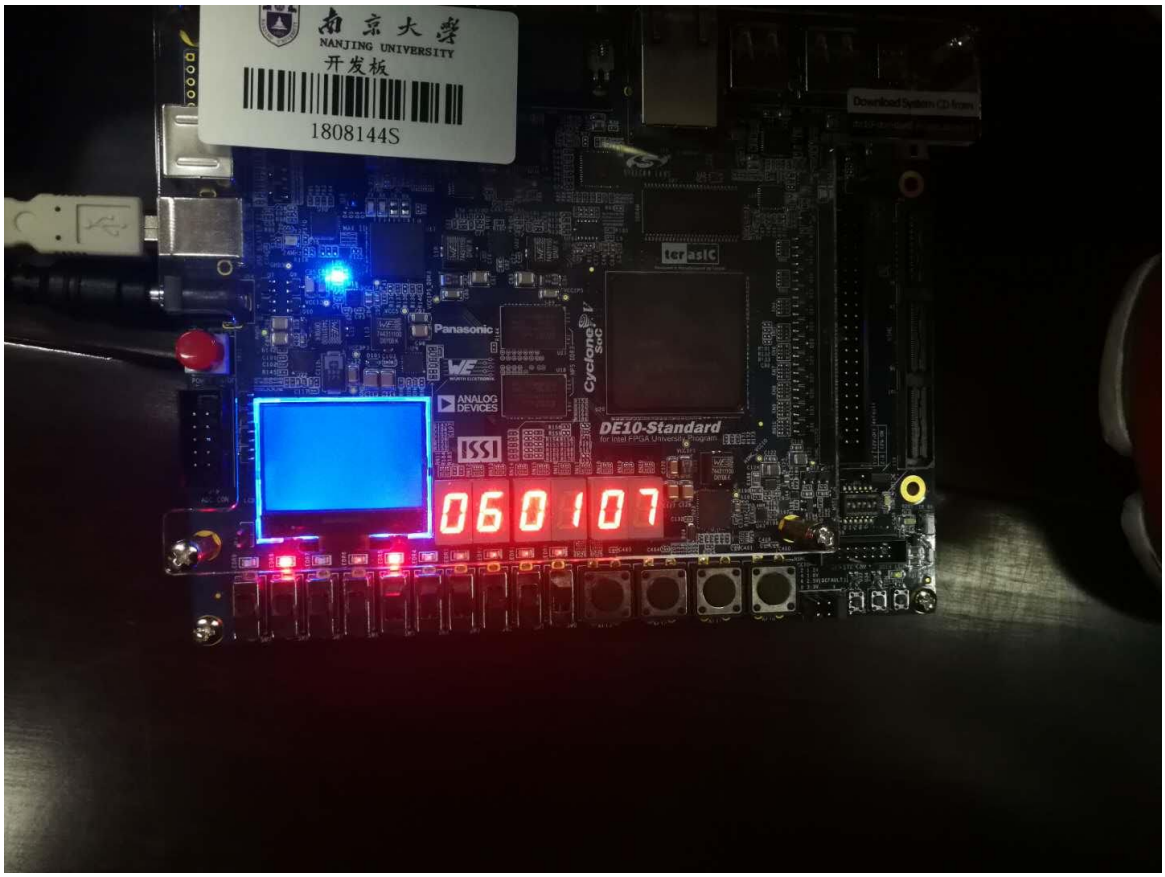
Eeg[2]	Ou...ut	PI...22	4A	B...0	PI...22	2.5 V		12...t)	1 (...lt)
Eeg[1]	Ou...ut	PI...22	4A	B...0	PI...22	2.5 V		12...t)	1 (...lt)
Eeg[0]	Ou...ut	PI...21	4A	B...0	PI...21	2.5 V		12...t)	1 (...lt)
Feg[6]	Ou...ut	PI...21	4A	B...0	PI...21	2.5 V		12...t)	1 (...lt)
Feg[5]	Ou...ut	PI...19	4A	B...0	PI...19	2.5 V		12...t)	1 (...lt)
Feg[4]	Ou...ut	PI...19	4A	B...0	PI...19	2.5 V		12...t)	1 (...lt)
Feg[3]	Ou...ut	PI...20	4A	B...0	PI...20	2.5 V		12...t)	1 (...lt)
Feg[2]	Ou...ut	PI...20	4A	B...0	PI...20	2.5 V		12...t)	1 (...lt)
Feg[1]	Ou...ut	PI...21	4A	B...0	PI...21	2.5 V		12...t)	1 (...lt)
Feg[0]	Ou...ut	PI...21	4A	B...0	PI...21	2.5 V		12...t)	1 (...lt)
ZF	Ou...ut	PI...24	4A	B...0	PI...24	2.5 V		12...t)	1 (...lt)
asel	Input	PI...15	3B	B...0	PI...15	2.5 V		12...t)	
bsel	Input	PI...14	3B	B...0	PI...14	2.5 V		12...t)	
carry	Ou...ut	PI...22	5A	B...0	PI...22	2.5 V		12...t)	1 (...lt)
cho...[2]	Input	PI...28	5B	B...0	PI...28	2.5 V		12...t)	
cho...[1]	Input	PI...27	5B	B...0	PI...27	2.5 V		12...t)	
cho...[0]	Input	PI...30	5B	B...0	PI...30	2.5 V		12...t)	
out	Ou...ut	PI...25	4A	B...0	PI...25	2.5 V		12...t)	1 (...lt)
overflow	Ou...ut	PI...22	4A	B...0	PI...22	2.5 V		12...t)	1 (...lt)
t[6]	Input	PI...30	5B	B...0	PI...30	2.5 V		12...t)	
t[5]	Input	PI...29	5B	B...0	PI...29	2.5 V		12...t)	
t[4]	Input	PI...30	5B	B...0	PI...30	2.5 V		12...t)	
t[3]	Input	PI...28	5B	B...0	PI...28	2.5 V		12...t)	
t[2]	Input	PI...25	5B	B...0	PI...25	2.5 V		12...t)	
t[1]	Input	PI...25	5B	B...0	PI...25	2.5 V		12...t)	
t[0]	Input	PI...30	5B	B...0	PI...30	2.5 V		12...t)	
<<n...e>>									

➢ 开发板实现

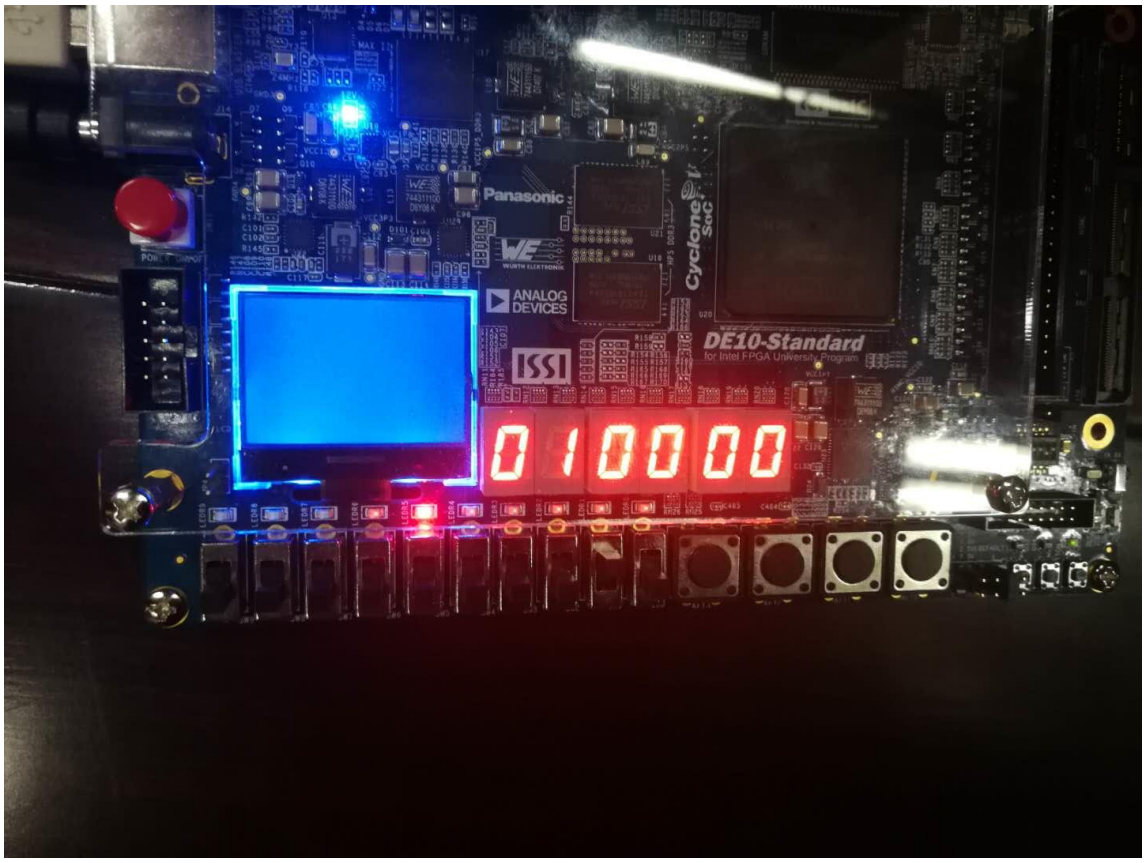
减法之减出正数：



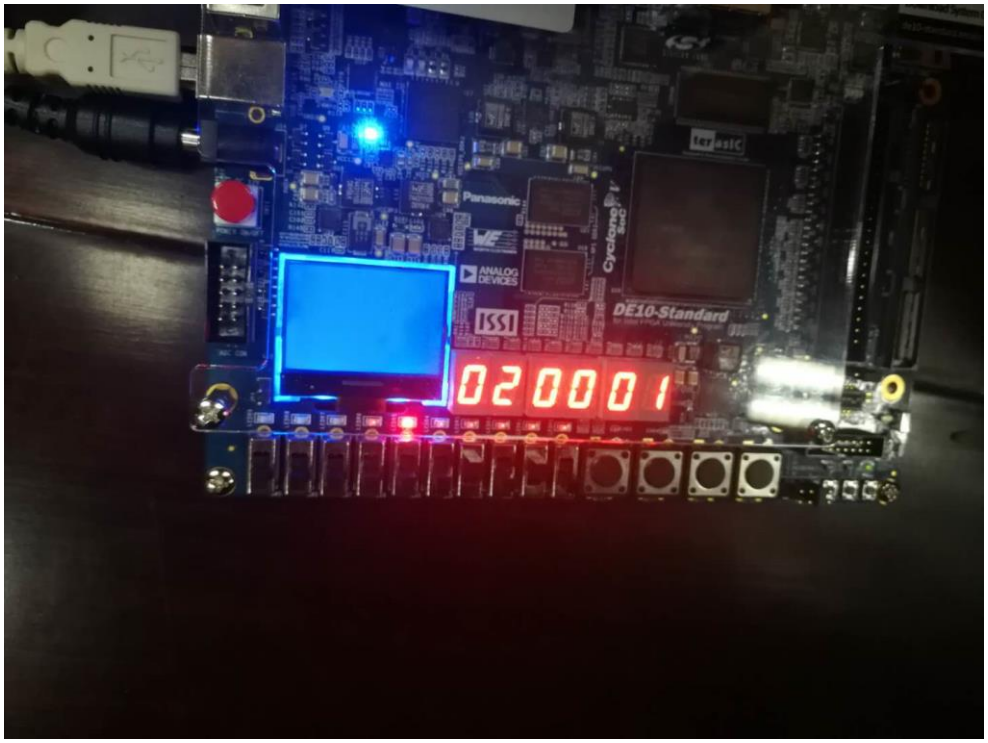
减法之减出负数:



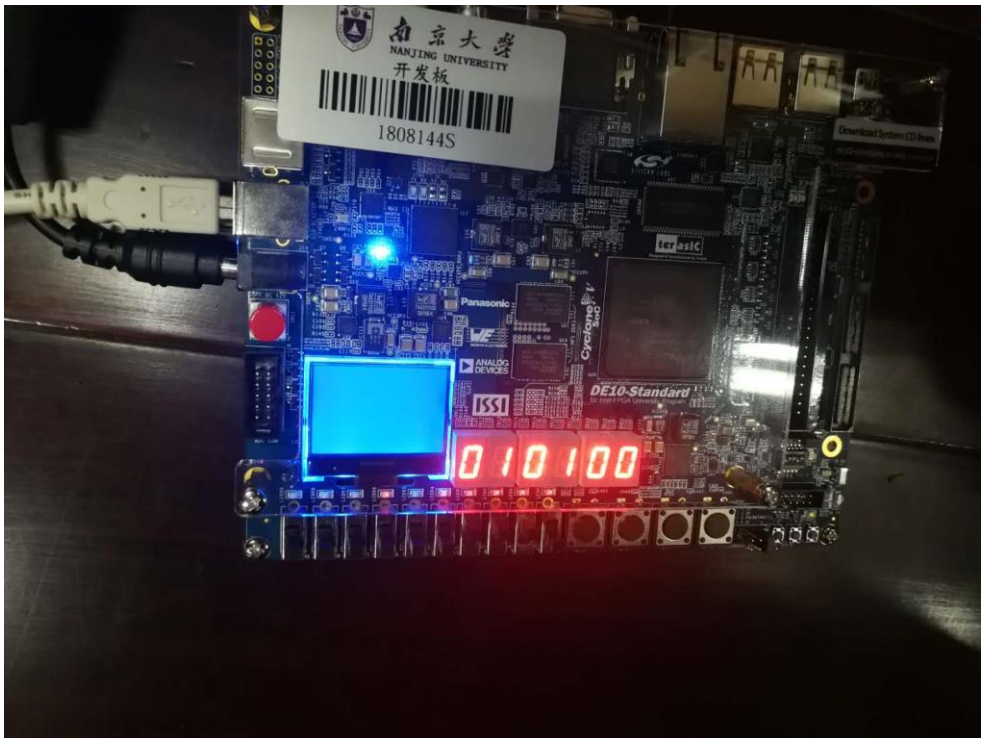
比较大小之相等:



比较大小之后面比前面大:



判断是否相等之不相等:



因为助教哥验收过啦，其他比较简单的就不放上来了.

五.实验中遇到的问题及解决方案（请具体的描述问题和解决方法）

当我全部都实现好了，仿真也很漂亮，美滋滋地来做引脚分配的时候，发现！真•无奈 A 和 B 开关不够了！不能被分配到同一个开关上，然后就悲剧了，之前一直没有想到 assign 语句，迷迷糊糊又看不懂 PDF 的意思，然后就乱改代码，结果浪费了很多时间，一直报错，这提醒我下次实现要考虑周全想清楚实验的每一步情况，先计划在实行，这样就不会再闹这样的笑话了.

六.实验得到的启示（积极思考）

以前实验比较后面才开始做，一些 dalao 同学会把 PDF 欠缺的问题都说了，给我省了很多时间. 但是这次我还是在 assign 上花了很多时间，还是平时不够细心认真吧. 做实验时出现错误不要慌,要慢慢去寻找，这样能节约很多的时间,同时做实验要看清实验要求，这样也会少花一些功夫.

七.意见和建议等

七段数码管引脚分配有点麻烦呀，有什么工具或者方法可以教教我们吗，截止交报告前我好像只找到一个叫做 System Builder 的 tool 可以.