2020-9-20

数字电路实验二 加法器与 ALU

秦嘉余

191220088

1348288404@qq.com

目录

1.实验目的

本次实验的目的是实现加法器和 ALU 这两种数字系统中的核心部件，学习简单 ALU 的设计方式，在 quartus 环境下利用 verilog 语言编写相关模拟程序，模拟仿真测试后写入 DE-10 开发板上。

2.实验原理

多位加法器可以利用移位加法器级联而成，而我们在 verilog 语言中无需关注这种基本的真值表达情况，只需要定义多个向量执行赋值语句即可，语言本身已经包含了处理的内容。
第一个器件加减法器中，加法直接相加即可，而减法需要将减数取反加一变成加法运算。同时根据一定的规则设定标志位即可。
第二个器件 ALU 的功能更加多样，包含了加法，减法，取反，位与，位或，位异或，大于判断，等于判断。除了加减法外，取反，位与，位或，位异或，等于判断操作均可直接赋值完成，语言本身已经可以处理。而大于判断由于我们输入的值代表为补码值，而我们定义的向量为无符号数，所以需要单独处理，不能直接比较大小。其中一种比较方法为把大于判断转化为减法判断是否大于 0 来判断。
经过上述的实验原理，我们已经可以设置加法器和 ALU 部件了

3.实验环境/器材

本次实验的环境为 Quartus17.1 版本
本次实验的器材为 DE10 Standard 开发板

4.程序代码

本次实验需要实现加减法器以及 ALU 部件，由于 ALU 部件中已经包含加减法器的内容，所以以下内容仅展示 ALU 的部分。

输入有四位输入 A,B，选择功能端 Op 为 3 位，共有八种功能，如下图：

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;

四位输出 Result，一位标志位 Carry，Overflow

代码在 ALU.v 文件中，由于篇幅问题，不在此处全文展示

主要结构为 case 选择，对于每种情况执行不同操作，其中需要注意 Op 对于的 Buttom 为低电平有效，所以 Op 为反值

```
3'b111:begin
    {Carry,Result}=A+B;
    Overflow=(A[3]==B[3])&&(Result[3]!=A[3]);
    end
```

其中加法直接执行加法，利用连接符判断 Carry 是否进位，利用 Overflow 的符号位判断是否溢出。

```
3'b110:begin
    t_no_Cin={4{Op[0]}}^B;
    {Carry,Result}=A+t_no_Cin+Op[0];
    Overflow=(A[3]==t_no_Cin[3])&&(Result[3]!=A[3]);
    end
```

减法则将减数取反加一变为加法，其余相同

```
3'b101:begin
    Result=~A;
    Carry=0;
    Overflow=0;
    end
```

中间的取反，位与，位或，位异或操作均直接操作即可

```
3'b001:begin
    if(A[3]^B[3])
        Result=B[3];
    else if(A==B)
        Result=0;
    else
        begin
        t_no_Cin={4{Op[0]}}^B;
        {Carry,Result}=A+t_no_Cin+Op[0];
        Result=!Result[3];
        end
    Carry=0;
    Overflow=0;
    end
```

大于判断，如果符号不同可以直接出结果，如果相等也可以直接出结果，如果符号相同则需要转化为减法，然后判断结果与 0 的大小

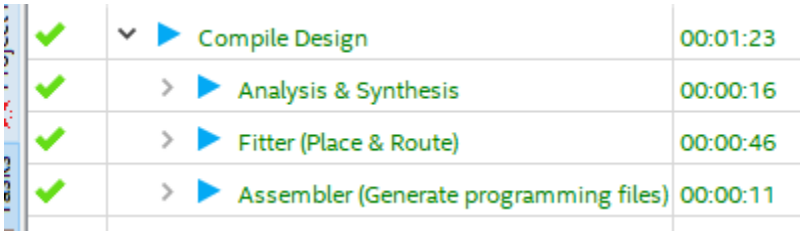
```
3'b000:begin
    Result=(A==B);
    Carry=0;
    Overflow=0;
    end
```

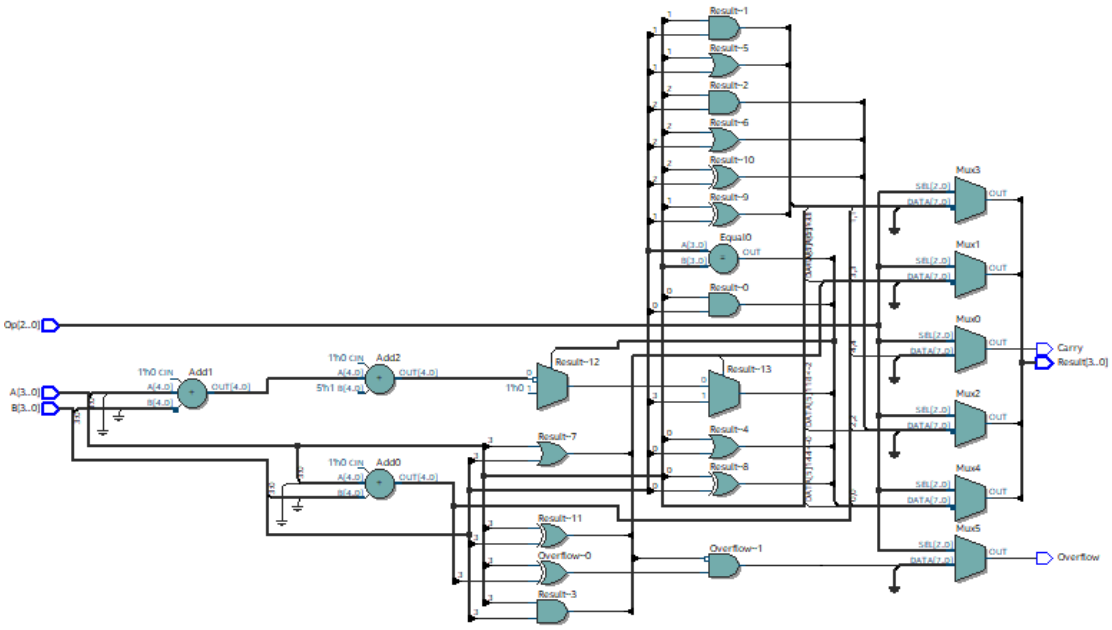
等于判断也直接操作即可

5.实验步骤

该部分主要展现是实验中各步骤

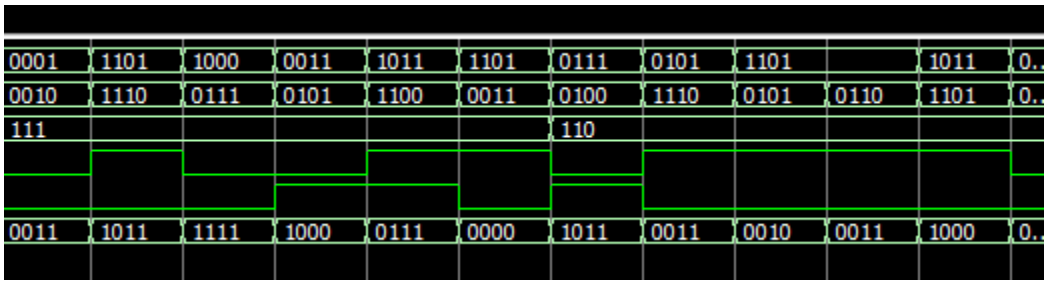
第一步：根据器件建立合适的项目文件，新建实验代码文件，键入相关代码，并且分析与综合，查看电路原理图，展示如下：





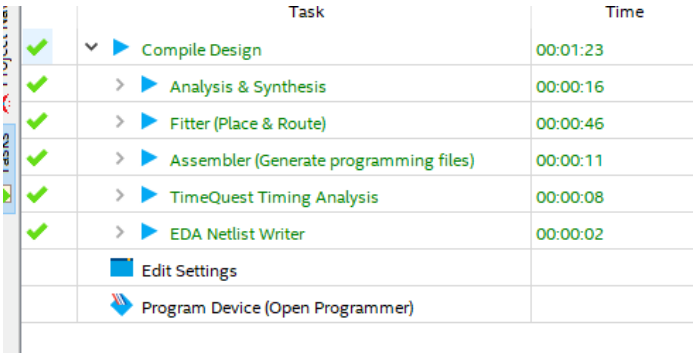
第二步：编写模拟测试代码，在 Test Benches 中链接相关模拟测试文件，最后利用 RTL Simualtion 查看波形图，测试程序是否正确，展示如下：

```
begin
// code that executes only once
// insert code here --> begin
   Op=111;  A=1;      B=2;   #10;
            A=-3;     B=-2;  #10;
            A=-8;     B=7;   #10;
            A=3;      B=5;   #10;
            A=-5;     B=-4;  #10;
            A=-3;     B=3;   #10;
   Op=110;  A=7;      B=4;   #10;
            A=5;      B=-2;  #10;
            A=-3;     B=5;   #10;
            A=-3;     B=6;   #10;
            A=-5;     B=-3;  #10;
            A=3;      B=3;   #10;
   Op=101;  A=4'b0000;       B=0;   #10;
            A=4'b1111;       B=0;   #10;
            A=4'b1010;       B=0;   #10;
            A=4'b0101;       B=0;   #10;
   Op=100;  A=4'b0000;       B=4'b1111;   #10;
            A=4'b1111;       B=4'b0000;   #10;
            A=4'b1010;       B=4'b0101;   #10;
            A=4'b1010;       B=4'b1000;   #10;
```



测试仿真代码仅展示部分，全文见 ALU.vt 文件

第三步: 第二步正确后，将引脚对应到开发板的相应接口上，再编译全部项目，连接 DE10 Standard 开发板到电脑，将程序写入开发板上，展示如下:



A[3]	Input	PIN_AC30	5B
A[2]	Input	PIN_AB28	5B
A[1]	Input	PIN_Y27	5B
A[0]	Input	PIN_AB30	5B
B[3]	Input	PIN_AD30	5B

Overflow	Output	PIN_AF25	4A
Result[3]	Output	PIN_AD24	4A
Result[2]	Output	PIN_AC23	4A
Result[1]	Output	PIN_AB23	5A

第四步: 写入开发板后，测试开发板输入输出是否正确，展示如下:

自此，该实验完成

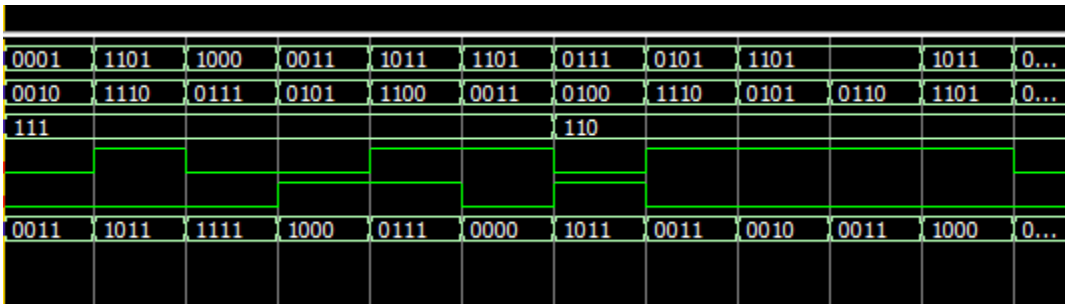
6.测试方法

本次实验采用测仿真仿真的方法， 编写的测试仿真代码如 ALU.vt，在此由于篇幅限制不再展示。

其中分 Op 从 111 到 000 测试 8 种功能，每种功能选择有代表性的组合输入，测试了大部分有意义的情况

7.实验结果

仿真结果如下：



下载运行结果如下：

实验验收完成，与仿真结果一致

8.实验中遇到的问题及解决办法

在实验开始时，中对减法的标志位判断逻辑不太理解，在对 4 位二进制数的运算进行分析后了解了标志位判断的原理。

在比较大小的功能实现时，一开始打算直接判断，后来想到使用之前实现过的减法判断更为简便

9.对实验的思考

此时，PA 也进行到有关 ALU 的部分，在对标志位已经运算本身的实现上，两者有一些区别，也由共通之处，这让人更加体会到硬件语言与普通语言的不同之处。

10.思考题

思考题 1

需要比较 A1，B1 和运算结果的符号位，因为减法变成加法时才产生了溢出，此时的溢出时加法器产生的，而加法器的操作数是 A1，B1，自然需要比较 A1， B1 与结果

思考题 2：

不完全一样，其中方法一是正确的，因为以 4 位数为例，最小值为-8，最大值为 7，这样的

情况下， 如若把-8 取反加一变为 8，是超出了表示的范围的，所以 Cin 的 1 不能提前加到 B 取反的值上，应该在加法执行时一起加上，这样，溢出的产生是在 A,B 加法时产生的，而不是 B 取反加一时产生的，对 Overflow 判断时，不会因为 t_add_Cin 已经溢出而产生错误，这样才不会影响 Overflow 的判断

思考题 3:

缩减运算时 verilog 的一种特殊语法，可以简单的对一个多位运算数执行自身的位操作，例如，|A 等价于 A[0]|A[1]|A[2]|A[3]，显然这种方法判断是否为 0 更加简单，不需要额外的比较器