数字逻辑与计算机组成实验

（组合逻辑电路设计）lab03:

加法器与 ALU

姓名：郑凯琳

学号：205220025

邮箱：205220025@smail.nju.edu.cn

（一） 实验目的

实现一个带有逻辑运算的简单 ALU

设计一个能实现如下功能的 4 位带符号位的**补码** ALU：

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;

ALU 进行加减运算时，需要能够判断结果是否为 0，是否溢出，是否有进位等。这里，输入的操作数 A 和 B 都已经是补码。比较大小请按带符号数的方式设置。

执行逻辑操作时不需要考虑 overflow 和溢出。

（二） 实验原理

全加器：二进制补码 – 简化 ALU 加法和减法的运算

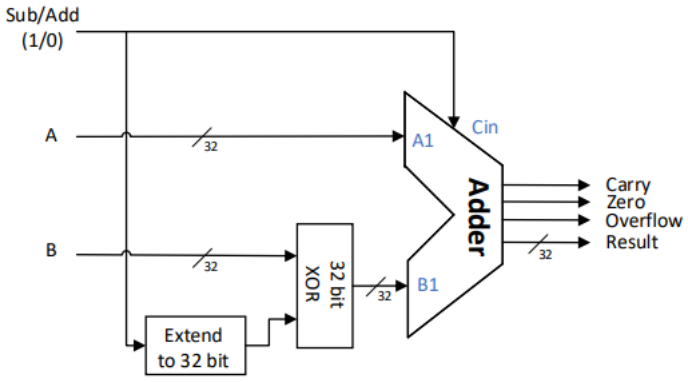


图 3-2: 简单加减 ALU

（三） 实验环境/器材等

硬件器材：Nexys A7-100T 开发板

软件平台：Vivado 开发平台

（四） 实验过程

设计思路：case 语句 – 实现各个功能

1. 加法：

   {cf, F} = A + B; //cf 为进位信号，F 为结果

   //溢出信号：若 A 和 B 符号相同但 F 与之相异，则发生溢出

2. 减法：

   B_com = ~B + 1; //对减数 B 取其负数的补码

   {cf, F} = A + B_com; //cf 为进位信号，F 为结果

   //溢出信号：若 A 和 B_com 符号相同但 F 与之相异，则发生溢出

3. 比较大小：

   由于需要考虑符号位，故将输入的 2 个二进制补码操作数转换为整型数比较。

设计代码：

```
module lab03(
    input [3:0] A,
    input [3:0] B,
    input [2:0] ALUctr,
    output reg [3:0] F,
    output reg cf,
    output reg zero,
    output reg of
    );

    integer a;
    integer b;
    reg [3:0] B_com;
```

```
always @ (*)
begin cf = 0; of = 0; F = 0;
    case(ALUctr)

        3'b000: //A+B
        begin
            {cf, F} = A + B;
            of = (A[3] == B[3]) && (F[3] != A[3]);
        end

        3'b001: //A-B
        begin
            B_com = ~B + 1;
            {cf, F} = A + B_com;
            of = (A[3] != B[3]) && (F[3] != A[3]);
        end

        3'b010: //Not A
        begin
            F = ~A;
        end

        3'b011: //A and B
        begin
            F = A & B;
        end

        3'b100: //A or B
        begin
            F = A | B;
        end

        3'b101: //A xor B
        begin
            F = A ^ B;
        end

        3'b110: //比较大小
        begin
            a = -A[3]*8 + A[2]*4 + A[1]*2 + A[0];
            b = -B[3]*8 + B[2]*4 + B[1]*2 + B[0];
            F = a > b;
        end

        3'b111: //判断相等
        begin
            F = (A == B);
        end
```

测试代码:

利用 task 功能函数

```
task check;

    input [3:0] true_F;
    input true_cf, true_zero, true_of;

    begin
        if(F != true_F)
            $display("Error: A = %h, B = %h. F should be %h", A, B, true_F);
        if(cf != true_cf)
            $display("Error: cf = %d. cf should be %d", cf, true_cf);
        if(zero != true_zero)
            $display("Error: zero = %d. zero should be %d", zero, true_zero);
        if(of != true_of)
            $display("Error: of = %d. of should be %d", of, true_of);
        else
            $display("CORRECT");
    end

endtask
```

调用 task 功能函数：如果输出结果和预期结果不一样，利用系统调用来输出提示

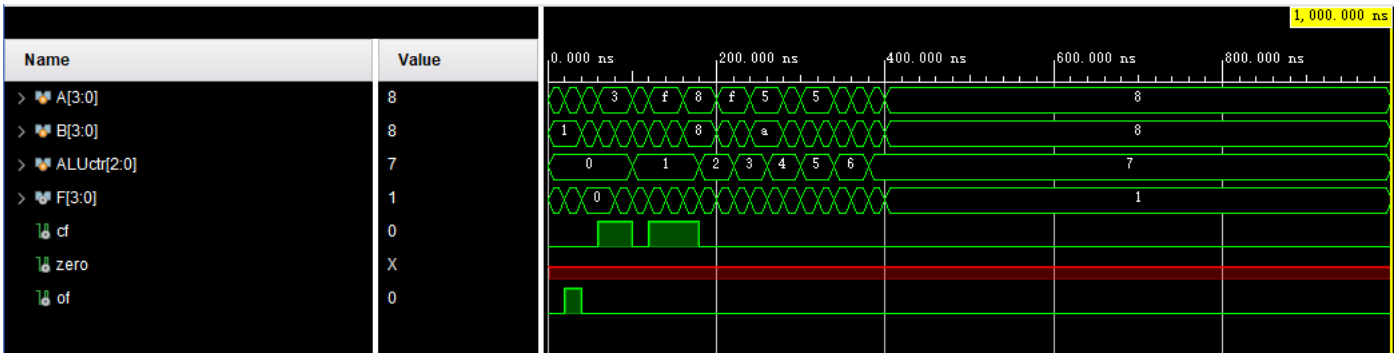
```
    initial
    begin
        ALUctr = 3'b000; A = 4'b0001; B = 4'b0001; #20; check(4'b0010, 0, 0, 0);
                         A = 4'b0111; B = 4'b0001; #20; check(4'b1000, 1, 0, 0);
                         A = 4'b0000; B = 4'b0000; #20; check(4'b0000, 0, 0, 1);
                         A = 4'b0011; B = 4'b1101; #20; check(4'b0000, 0, 1, 1);
                         A = 4'b0011; B = 4'b1110; #20; check(4'b0001, 0, 1, 0);
        ALUctr = 3'b001; A = 4'b0000; B = 4'b0111; #20; check(4'b1001, 1, 0, 0);
                         A = 4'b1111; B = 4'b1000; #20; check(4'b0111, 0, 0, 0);
                         A = 4'b1111; B = 4'b0111; #20; check(4'b1000, 0, 0, 0);
                         A = 4'b1000; B = 4'b1000; #20; check(4'b0000, 0, 1, 0);
        ALUctr = 3'b010; A = 4'b1000; B = 4'b1000; #20; check(4'b0111, 0, 0, 0);
                         A = 4'b1111; B = 4'b0000; #20; check(4'b0000, 0, 0, 0);
        ALUctr = 3'b011; A = 4'b1111; B = 4'b1000; #20; check(4'b1000, 0, 0, 0);
                         A = 4'b0101; B = 4'b1010; #20; check(4'b0000, 0, 0, 0);
        ALUctr = 3'b100; A = 4'b0101; B = 4'b1010; #20; check(4'b1111, 0, 0, 0);
                         A = 4'b0100; B = 4'b1110; #20; check(4'b1110, 0, 0, 0);
        ALUctr = 3'b101; A = 4'b0101; B = 4'b1010; #20; check(4'b1111, 0, 0, 0);
                         A = 4'b0101; B = 4'b1111; #20; check(4'b1010, 0, 0, 0);
        ALUctr = 3'b110; A = 4'b1000; B = 4'b0000; #20; check(4'b0000, 0, 0, 1);
                         A = 4'b0000; B = 4'b1111; #20; check(4'b0001, 0, 0, 0);
        ALUctr = 3'b111; A = 4'b0011; B = 4'b0001; #20; check(4'b0000, 0, 0, 0);
                         A = 4'b1000; B = 4'b1000; #20; check(4'b0001, 0, 0, 0);
    end
```

硬件实现（引脚分配）:

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (18)													
A (4)	IN			☑	14	LVCMOS33*	3.300				NONE	NONE	
A[3]	IN		U18	☑	14	LVCMOS33*	3.300				NONE	NONE	
A[2]	IN		T18	☑	14	LVCMOS33*	3.300				NONE	NONE	
A[1]	IN		R17	☑	14	LVCMOS33*	3.300				NONE	NONE	
A[0]	IN		R15	☑	14	LVCMOS33*	3.300				NONE	NONE	
ALUctr (3)	IN			☑	(Multiple)	LVCMOS33*	3.300				NONE	NONE	
ALUctr[2]	IN		M13	☑	14	LVCMOS33*	3.300				NONE	NONE	
ALUctr[1]	IN		L16	☑	14	LVCMOS33*	3.300				NONE	NONE	
ALUctr[0]	IN		J15	☑	15	LVCMOS33*	3.300				NONE	NONE	
B (4)	IN			☑	(Multiple)	LVCMOS33*	3.300				NONE	NONE	
B[3]	IN		R16	☑	14	LVCMOS33*	3.300				NONE	NONE	
B[2]	IN		U8	☑	34	LVCMOS33*	3.300				NONE	NONE	
B[1]	IN		T8	☑	34	LVCMOS33*	3.300				NONE	NONE	
B[0]	IN		R13	☑	14	LVCMOS33*	3.300				NONE	NONE	
F (4)	OUT			☑	(Multiple)	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
F[3]	OUT		N14	☑	14	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
F[2]	OUT		J13	☑	15	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
F[1]	OUT		K15	☑	15	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
F[0]	OUT		H17	☑	15	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
Scalar ports (3)													
cf	OUT		R18	☑	14	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
of	OUT		V17	☑	14	LVCMOS33*	3.300		12		NONE	FP_VTT_50	
zero	OUT		U17	☑	14	LVCMOS33*	3.300		12		NONE	FP_VTT_50	

（五） 实验结果

仿真结果:




```
# run 1000ns
CORRECT
Error: cf = 0. cf should be 1
Error: of = 1. of should be 0
Error: of = 0. of should be 1
Error: cf = 1. cf should be 0
Error: of = 0. of should be 1
Error: cf = 1. cf should be 0
CORRECT
Error: cf = 0. cf should be 1
CORRECT
Error: cf = 1. cf should be 0
CORRECT
Error: cf = 1. cf should be 0
CORRECT
Error: cf = 1. cf should be 0
CORRECT
CORRECT
CORRECT
CORRECT
CORRECT
CORRECT
CORRECT
CORRECT
CORRECT
Error: of = 0. of should be 1
CORRECT
CORRECT
CORRECT
```

（六） 实验中遇到的问题及解决方法

问题 1：不理解 cf（cout）和 of（overflow）

解决办法：查询资料

- of：计算结果超出限定范围（-8 至 7）时为 1
- cf：（操作数为无符号）加减运算时，若最高位再有进位为 1

问题 2：不明白 task 的用法

解决办法：在网上查了很多资料，找了很久才搞清楚

- 任务定义

```
task <task_name>
    <declaration>;
    <procedure statements>;
endtask

task check
    input [7:0] x,y;    //输入端口说明
    output [7:0] temp;  //输出端口说明

    if(x > y)
        temp = x;
    else
        temp = y;
endtask
```

- task：任务定义的开始；endtask：任务定义的结束
- <task_name>：任务名
- <declaration>：端口声明语句 & 变量声明语句（任务接收输入值 & 返回输出值）
- <procedure statements>：任务操作的过程语句
  - 如过程语句多于一条，将其放在语句块 begin-end

注意事项：

（1）第一行“task”语句中不能列出端口名称。

（2）任务的输入、输出端口和双向端口数量不受限制，甚至可以没有输入、输出以及双向端口。

（3）在任务定义的过程语句：

- YES: 延迟控制语句、"disable 中止语句"（中断正在执行的任务，继续向下执行），
→ 但不可综合。
- YES: 调用其他任务或函数，也可调用自身。
- NO: initial 和 always。

- 任务调用

```
16    task <task_name> (端口1,  端口 2, ........,  端口 N);
```

  - 参数列表的顺序必须与任务定义中的端口声明顺序一致
  - 调用语句是过程语句：所以接收返回数据的变量必须是寄存器类型 reg

注意事项：

（1）任务调用只能出现在过程块内。

（七） 思考题

1.

☞ **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：

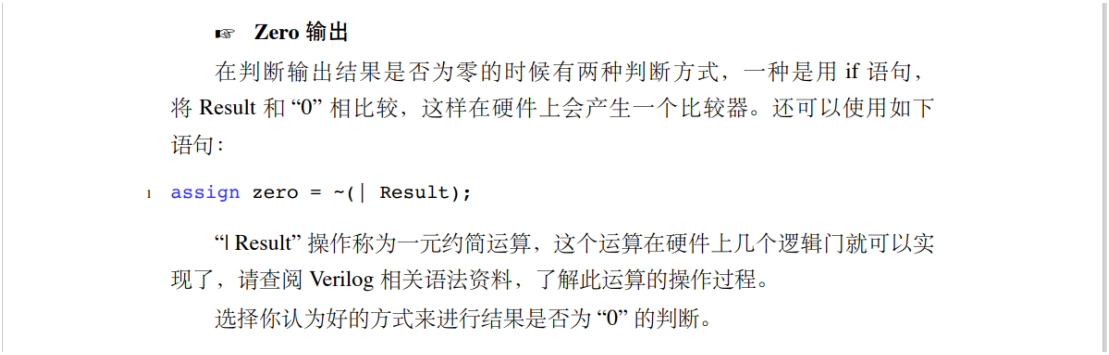
```
assign t_add_Cin =( {n{Cin}}^B )+ Cin ;  //  在这里请注意^运算和+运算的顺序
assign { Carry, Result } = A + t_add_Cin;
assign Overflow = (A[n-1] == t_add_Cin[n-1]) && (Result [n-1] != A[n-1]);
```

运算结果、进位位一样，但溢出位不一样。

因为两种方法在计算 overflow 的条件不一样：

- 方法 1：A 与 {n{cin}}^B 符号是否相同
- 方法 2：A 与 {n{cin}}^B+cin 符号是否相同

2.



☞ **Zero 输出**

在判断输出结果是否为零的时候有两种判断方式，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：

```
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。

😊 一元约简运算：代码简洁，硬件实现较方便

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，依次操作直到最后一位，得运算结果。

☹ if 语句判断结果是否为 0：代码不简洁，硬件实现不方便