**实验11**

**组合逻辑电路（下）**

# **实验介绍**

在本实验中，我们将为大家介绍一些常见的组合逻辑电路，包括加法器、编码器、译码器等。这些电路在数字电路中应用极为广泛，是数字电路设计的基础。 此外，我们还会介绍一些使用组合逻辑电路解决的问题，从需求出发，一步步设计组合逻辑电路，并最终编程实现。

# **实验内容**

## 1 组合逻辑元件

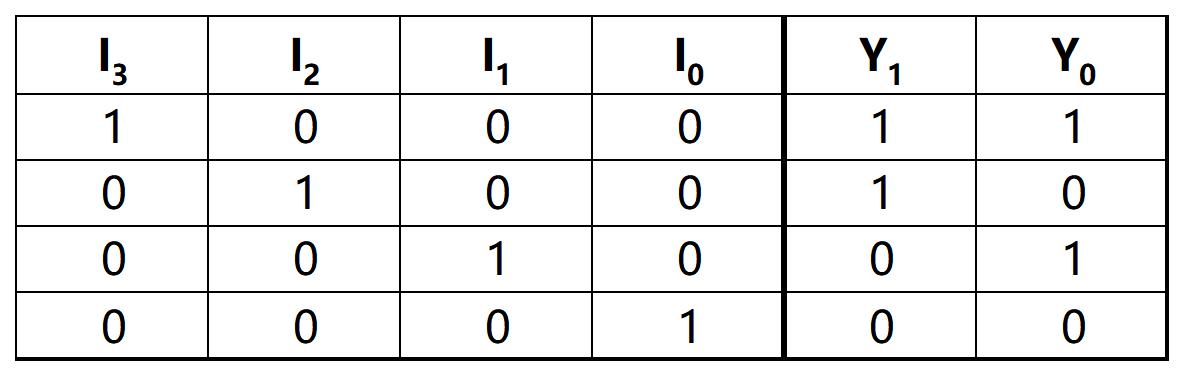
### ****1.1 编码器****

#### 1.1.1 简单编码器

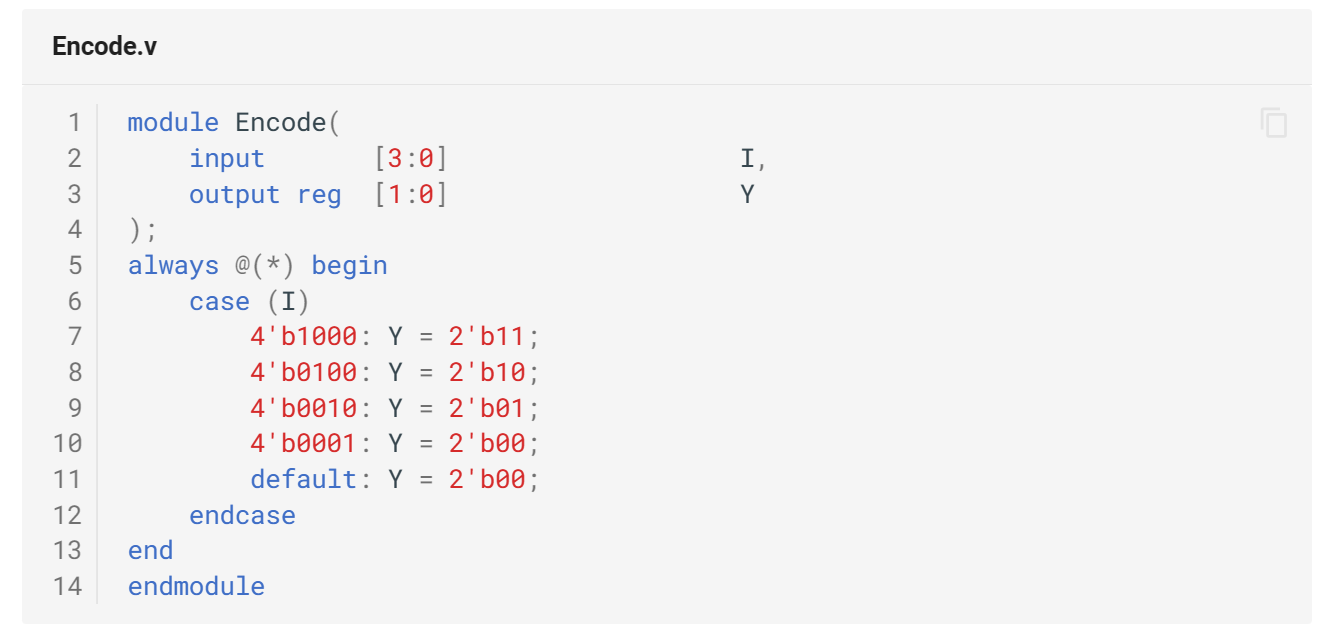
编码，是信息从一种形式或格式转换为另一种形式的过程，简单来讲就是语言的翻译过程。在计算机领域，我们需要将日常使用的自然语言翻译成计算机能够理解的二进制编码，这就是编码过程。能够实现该功能的数字电路我们称之为编码器。

某位国王想要与自己王国的四个区域建立电话联系，于是便为这四个区域各拉了一条电话线，它们汇总在皇宫的信号接收机。我们假定同一时间至多只有一个区域与国王通话。现在国王的需求是：当有电话进来时，接收机能够显示当前通话的区域编号。

我们把上面的场景形式化描述一下：输入信号有四位，分别记为 I3~I0；为了区分四个不同的区域，我们需要 2 bits 的二进制编码，也就是说输出信号有两位，分别记为 Y1~Y0。我们可以列出下图所示的真值表：

[](https://soc.ustc.edu.cn/Digital/lab4/figs/encode.jpg)

如何用 Verilog 语言描述编码器呢？我们给出如下的行为级描述代码。这里我们假定对于真值表列出情况之外的输入，译码器统一输出 0。



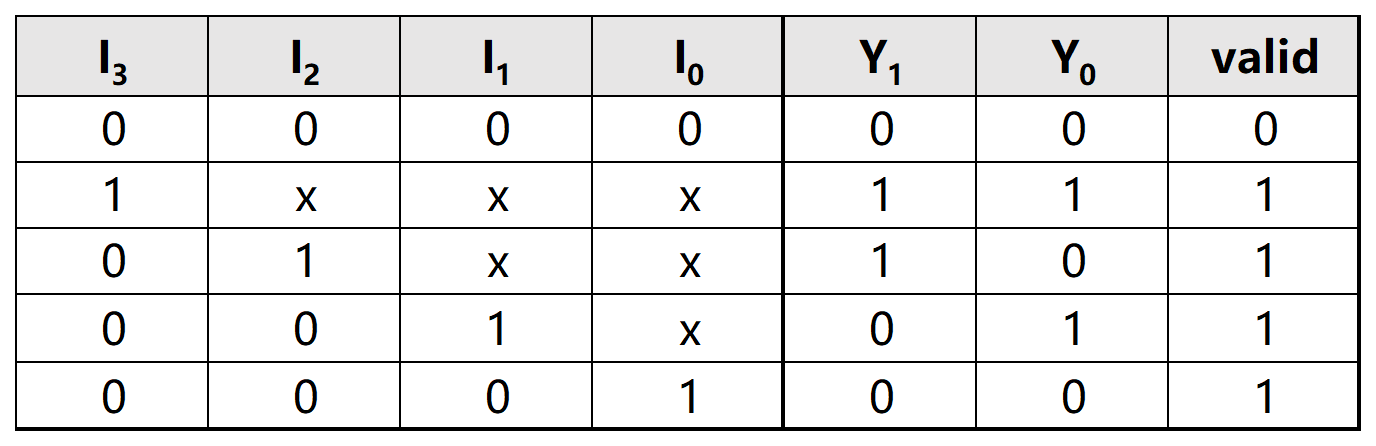
#### 1.1.2 优先编码器

现在，国王与四个地区的联系更为频繁了，同一时间与国王通话的区域数目没有限制，此时我们必须要有手段处理电话同时接入的情况。国王希望按照地区编号由大到小的顺序设定优先级，当多个区域同时接入电话时，国王会选择接听编号最大的那一个。

普通编码器虽然能实现编码的功能，但它仍有不少局限性，其中之一就表现为：普通编码器的输入端只能同时存在一个高电平信号，当我们不小心输入了多个高电平信号，比如输入 I = 4'b1111，根据代码编码器输出的结果为 2'b00, 与正常输入 I = 4'b0001 的输出结果相同，但我们无法判断此时输入了一个错误的信号。

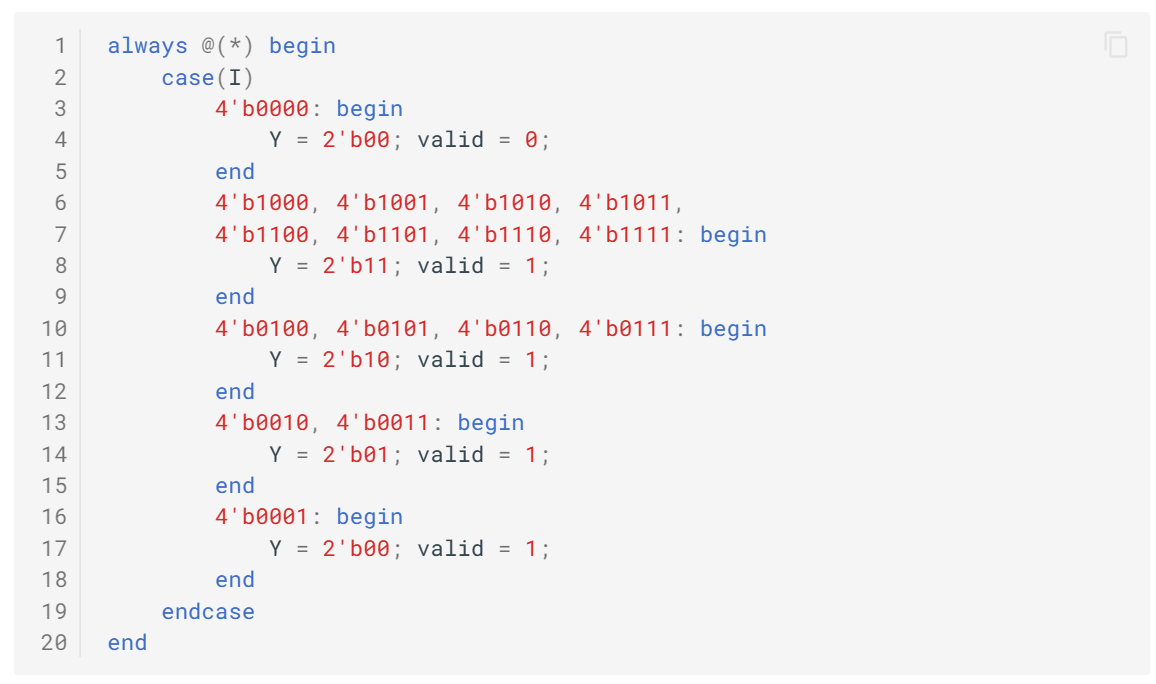
为了消除这种弊端，我们设想一种新的编码器：它的每个输入端有着不同的重要程度（更专业地说，有着不同的优先级），只要更重要的输入端输入了有效信号，我们就不再考虑来自其他输入端的信号。例如：当 𝐼3 输入有效信号时，就不再考虑来自 I2~I0 输入的信号，而在输出端直接输出 2'b11。仅有 𝐼3 输入无效信号（为 0）时才会检查 I2~I0 的内容。这就是优先编码器的思想。

一个符合要求的优先编码器真值表如下：

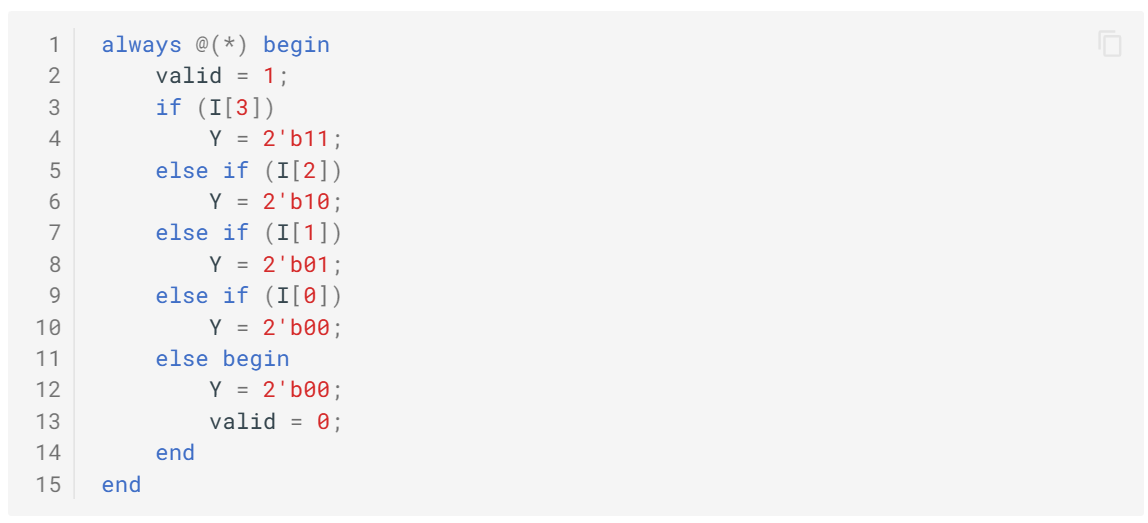
[](https://soc.ustc.edu.cn/Digital/lab4/figs/example/prio_encode.png)

这里我们使用 x 代表输入为 0/1 均可。不难看出，上面的真值表涵盖了所有 16 种可能的输入。我们引入了输入有效信号 valid。如果输入为 I = 4'b0000 则 valid 输出 0，表示当前并没有电话接入。

那么，如何使用 Verilog 语言描述呢？我们知道 case 语句是没有优先级顺序的，只能为每一种情况指定其对应的输出：



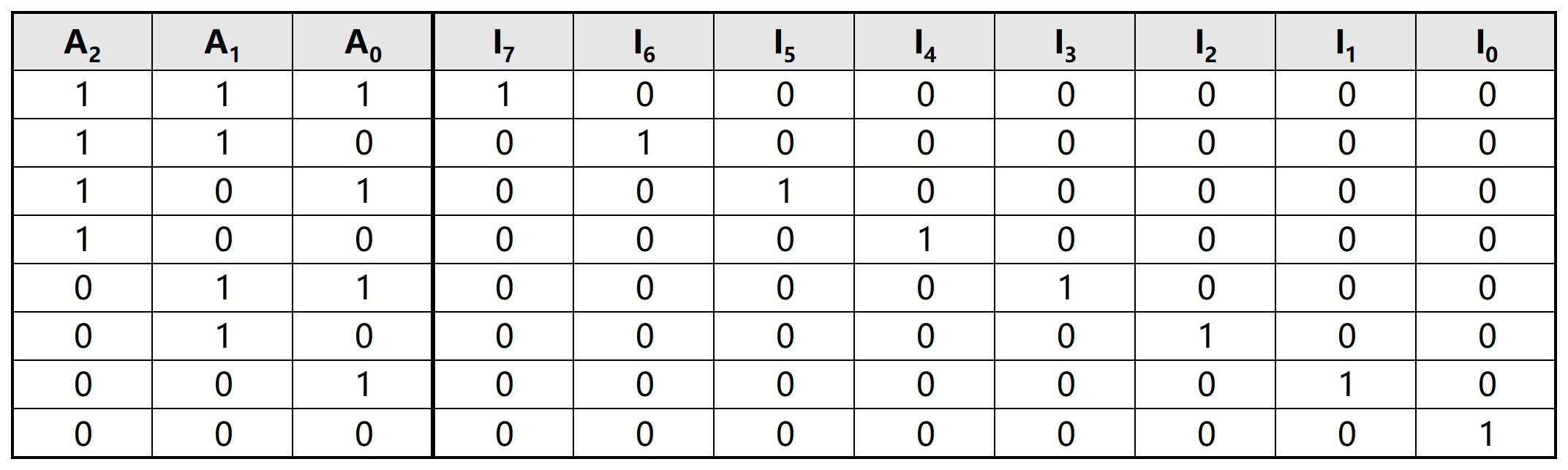
此外，我们也可以尝试使用 if-else-if 结构完成模块描述。



### ****1.2 译码器****

译码是编码的逆过程。编码过程将自然语言「翻译」成机器能理解的二进制语言， 而译码则是将二进制代码所代表的特定含义「翻译」成对应的自然语言。

国王将自己的领土扩展到了八个区域，并连接好了对应的电话线和优先编码器。现在，国王希望你能够帮助他设计一款译码器，根据输入的 3bits 编号 A2~A0 接通对应区域的电话线。简而言之，我们需要将输入的信号 A2 ∼ A0 翻译为对应的阿拉伯数字。同样地，我们使用 I7 ~ I0 表示 8 种可能的阿拉伯数字。 因此译码器的真值表如下图所示：

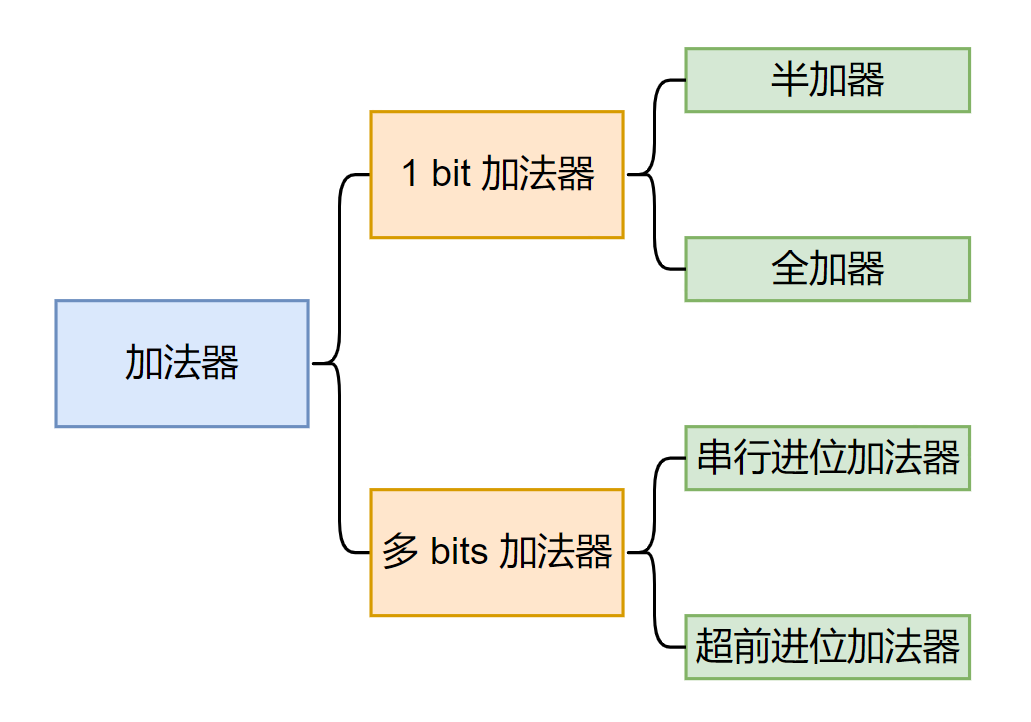
[](https://soc.ustc.edu.cn/Digital/lab4/figs/example/decode_1.png)

这里由于输入只有 3 bits，因此总可能的输入只有 8 种。译码器的Verilog代码实现如下：



### ****1.3 加法器****

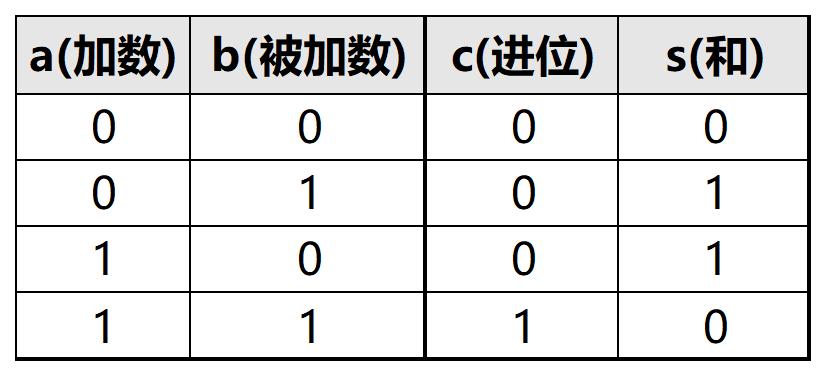
接下来的例子是关于加法器的。常用的加法器分类如下图所示。

[](https://soc.ustc.edu.cn/Digital/lab4/figs/adder.png)

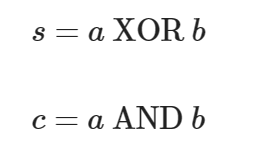
#### 1.3.1 1bit半加器

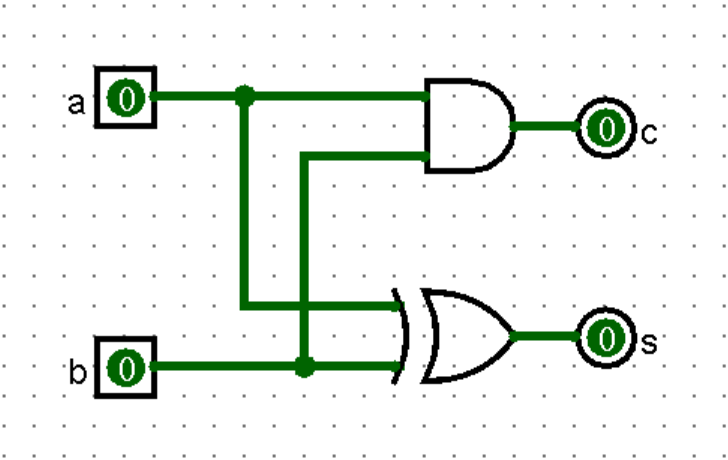
我们首先从最简单的一位半加器开始说起。半加运算是指不考虑来自低位的进位的加法，实现半加运算的电路被称之为半加器。

我们知道，二进制加法与十进制加法本质上是一致的，只是数字变成了二进制。这样的好处是每一位上可能的情况只有 22=4 种而不是 102=100 种。四种情况就是 1 bit 加法的四种可能：0+0、0+1、1+0、1+1。假定输入的两个 1bit 数为 a 和 b，由于结果可能是两位，我们引入两个 1bit 变量 s 和 c 表示结果，其中 s 表示低位，c 表示高位。据此，我们可以列出下面的真值表：

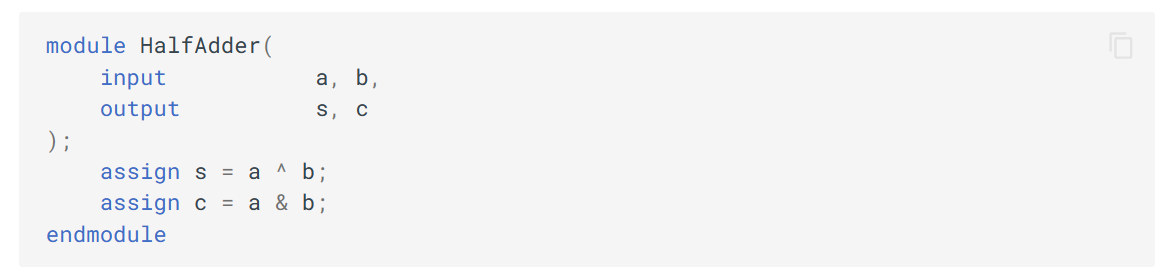
[](https://soc.ustc.edu.cn/Digital/lab4/figs/HA.jpg)

从上面的真值表中，我们不难得到 s、c 关于 a、b 的逻辑表达式。

由此便可以得到半加器的逻辑电路图：

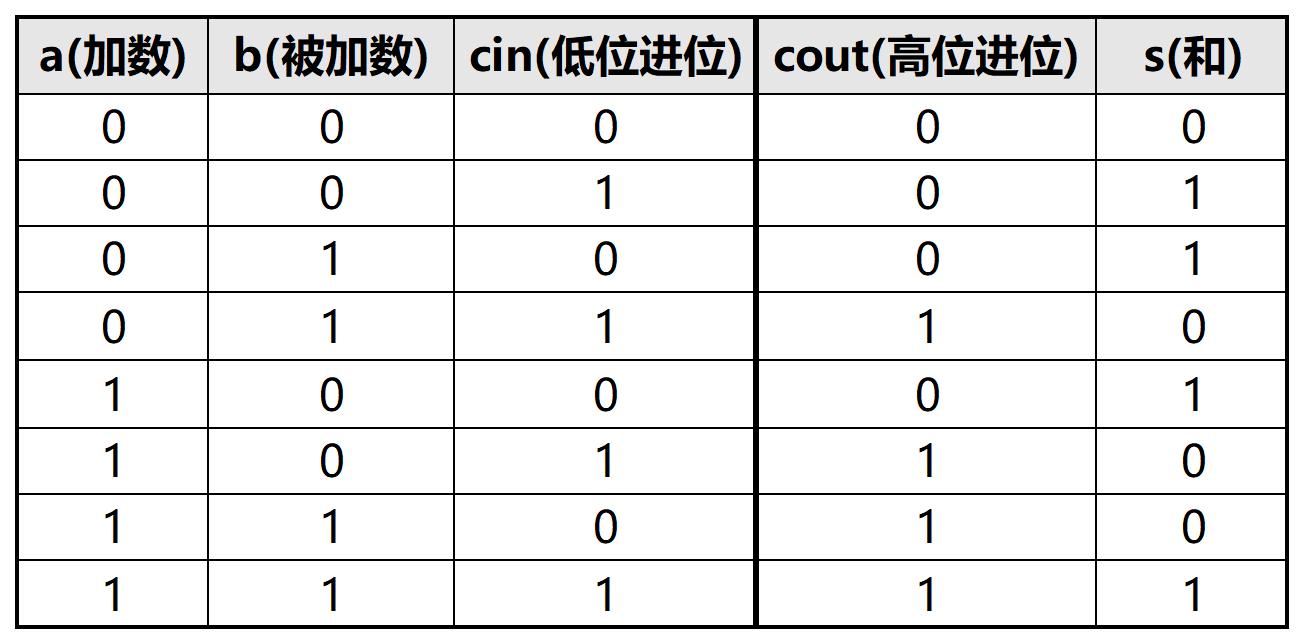
[](https://soc.ustc.edu.cn/Digital/lab4/figs/HA_circuit.png)

对应的 verilog 代码如下：

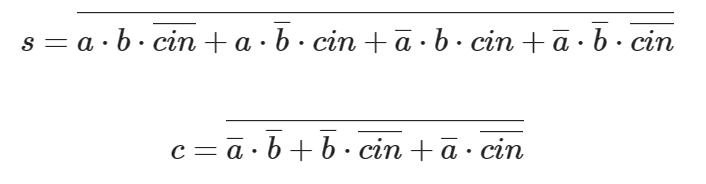


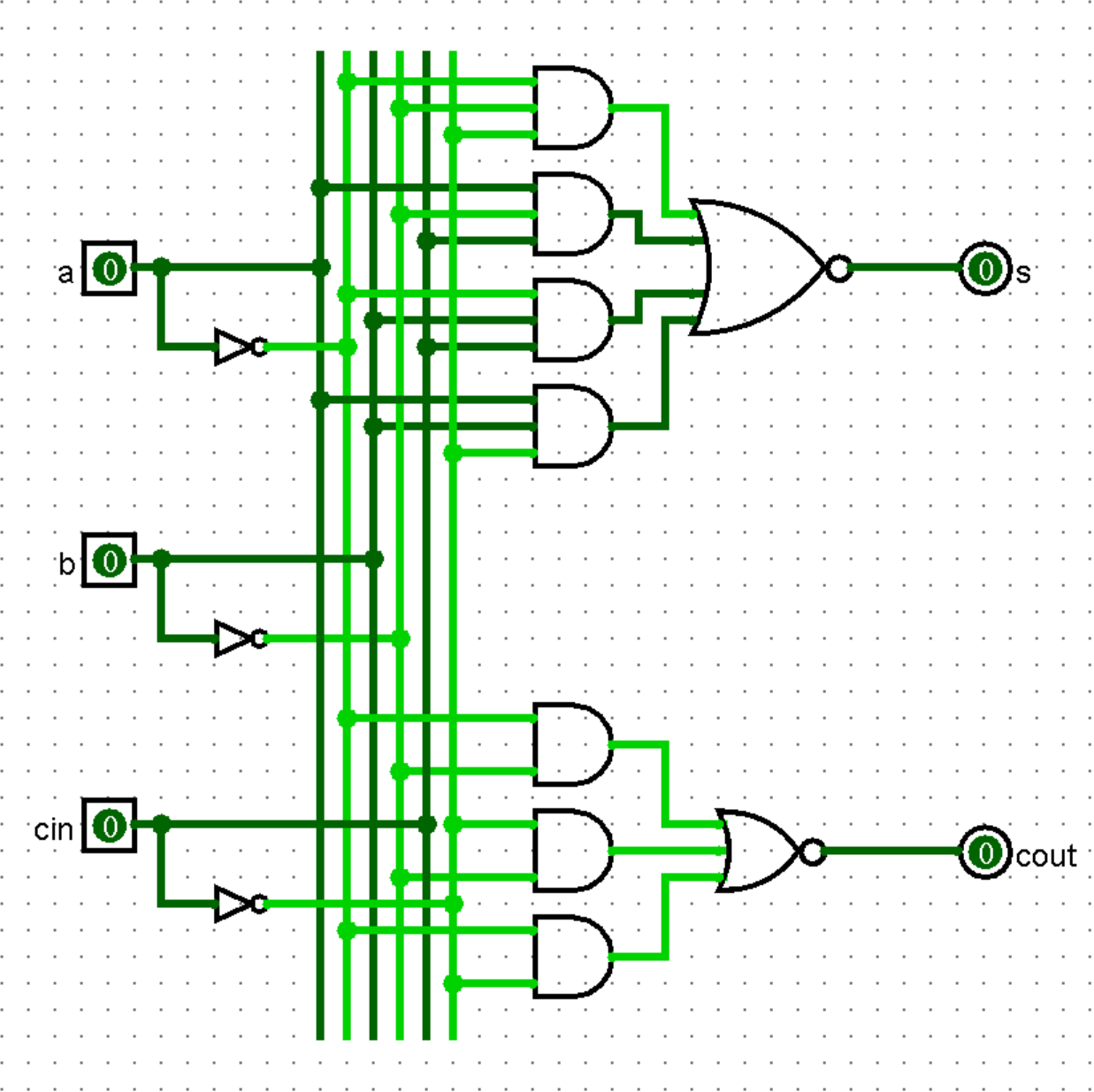
#### 1.3.2 1bit全加器

当涉及到多位加法时，除了最低位外，每一位都需要考虑来自低位的进位。因此半加器就需要进行一定的改进，支持两个加数以及低位进位三个数的想加。这种运算称为全加运算，实现电路被称为全加器。

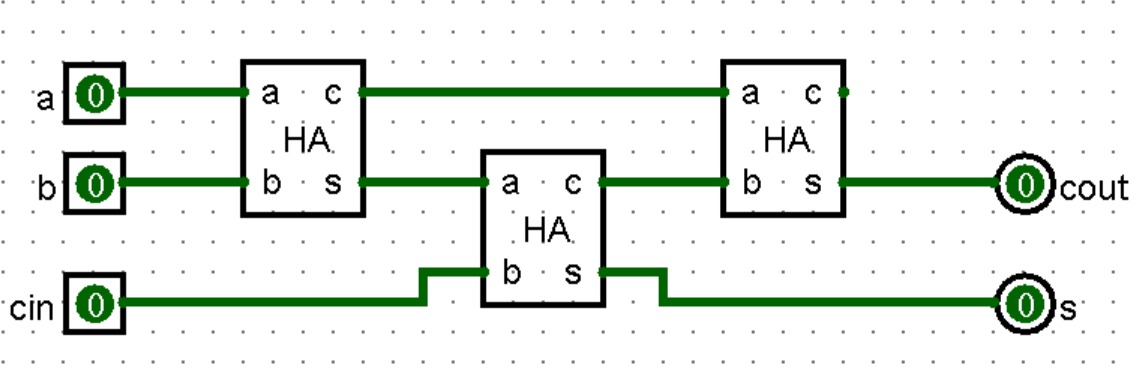
[](https://soc.ustc.edu.cn/Digital/lab4/figs/FA.jpg)

从真值表中得到逻辑表达式如下

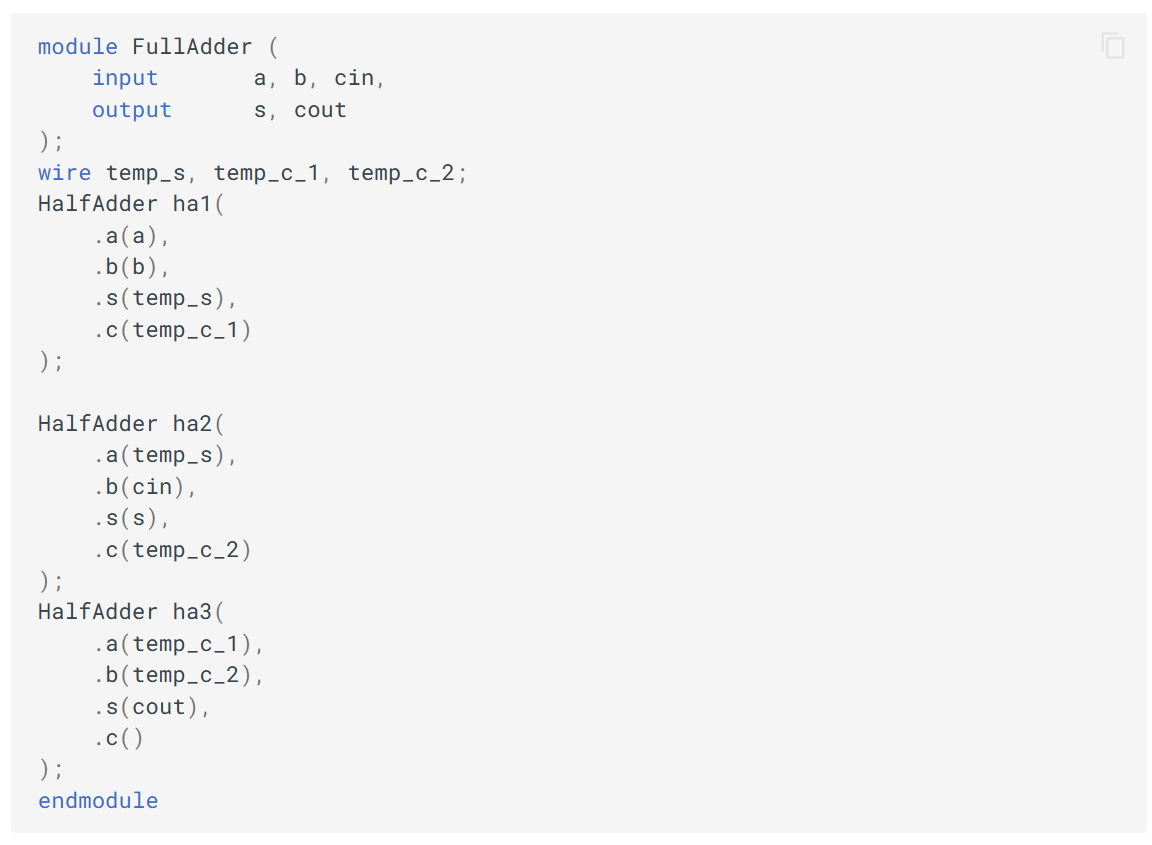
这个式子就复杂了一些，但也可以用逻辑门直接搭出。

[](https://soc.ustc.edu.cn/Digital/lab4/figs/FA_circuit_1.png)

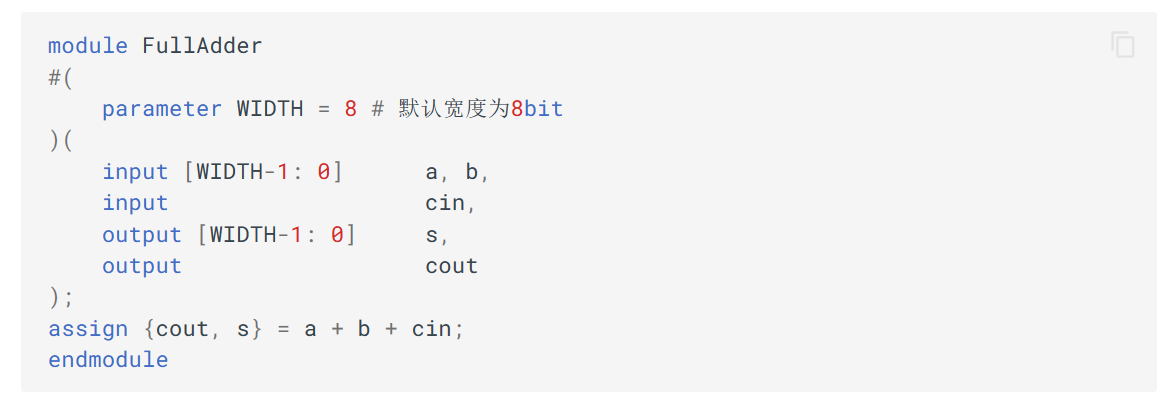
我们也可以直接使用先前的半加器搭建全加器。考虑半加器的功能是实现不考虑进位的加法，全加器在此基础上额外引入了前位进位 cin，我们可以将三个数的加法拆解为两次两个数的加法。示意图如下：

[](https://soc.ustc.edu.cn/Digital/lab4/figs/FA_circuit.png)

用 verilog 的描述如下：



对于大多数的应用场景，Verilog 描述加法器最合适的方式是行为描述而不是门级描述。我们完全可以使用 Verilog 的 "+" 运算符描述加法。



## 2 实际应用

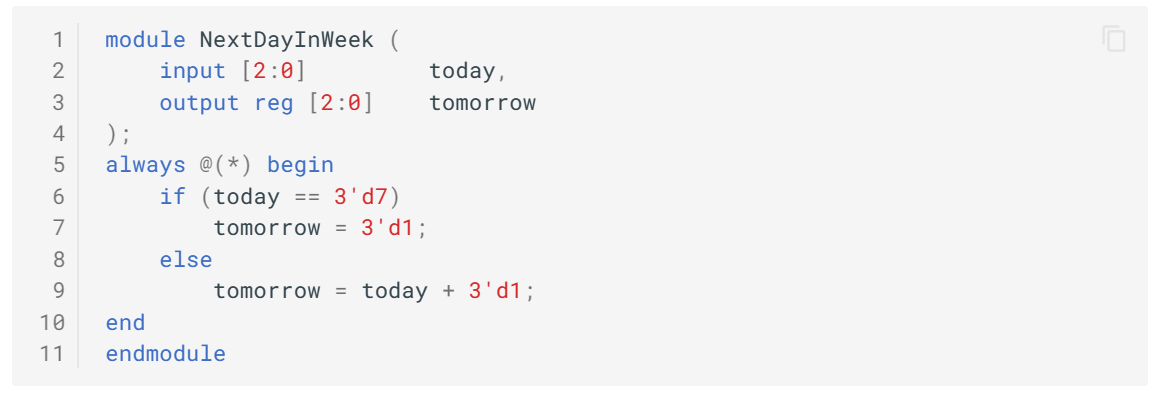
### ****2.1 日期电路****

我们考虑下面这个例子。某同学想要在开发板上搭载一个日历模块，其中一项功能需求是根据当前的日期计算明天的日期。模块的输入输出描述如下：



我们可以将功能拆解成两部分：计算日期以及计算星期，因为二者是独立的。

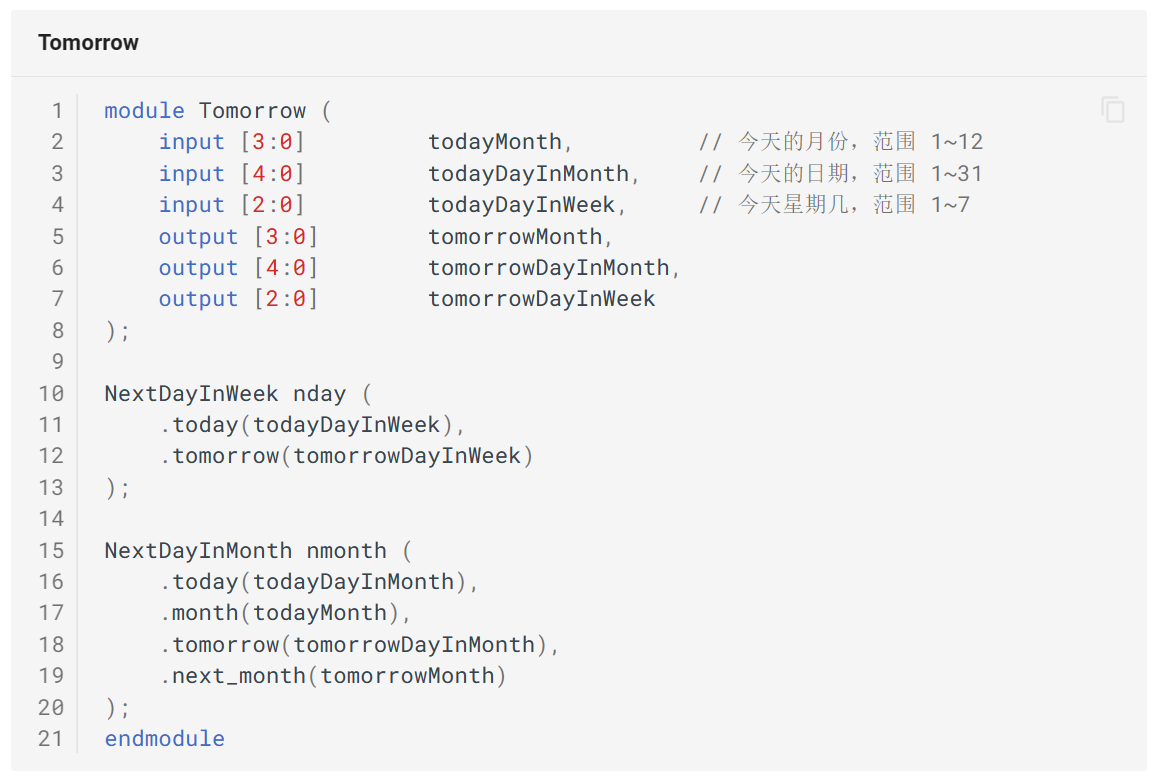
计算星期很简单，每一周都有 7 天，因此只需要判断当天是否是星期日即可。



计算日期则相对复杂一些。我们暂时先不考虑闰年的影响，每个月可能有 28、30 或 31 天，此时就需要使用分支语句进行描述



最后，将两个模块组合起来，即可得到最终的模块。



### ****2.2 6bits 5的倍数检测器****

如题，我们需要设计一个电路，检测输入的 6bits 二进制数是否为 5 的倍数。检验一个数是不是5的倍数，我们可以将其转换为十进制数，然后判断是否能被 5 整除。但是还有一个最为朴素的做法：枚举！

#### 2.2.1 暴力枚举

6bit 二进制数的取值范围为 0~63，我们可以枚举所有的情况，然后判断是否为 5 的倍数。我们可以使用 Verilog 的 case 语句来实现这一功能。



显然，上述方法在面对更大的数时就不太适用了，毕竟我们不太可能枚举所有的情况。那么我们该如何设计一个通用的电路呢？

一个自然的想法是构建一个有限状态机，通过状态机的状态来记录当前的余数，并通过结束时落到的状态确定最终余数。由于一个数除以 5 的余数只可能是 0~4 ，因此我们可以设计一个 3bits 的状态机，用于记录当前的余数。

很可惜，这并不属于组合逻辑的范畴，因为状态机需要记录当前的状态，而状态的改变是需要时间的。因此我们需要引入时钟信号，将其变成时序逻辑——这也是我们将在之后中介绍的内容。

而现在，我们只能另辟蹊径，寻找另一种自然的方法。这就是我们接下来要介绍的长除法。

#### 2.2.2 长除法

由于位数已经固定为 6bits，因此我们可以直接使用 Verilog 的除法运算符。但是这样的做法并不符合我们的初衷，我们希望能够自己设计一个电路，而不是直接使用现成的运算符。

我们以 6bits 二进制数除以 5 为例，假设我们要计算 011011 除以 101，我们可以模拟除法竖式的过程。

module LongDivision(

input [5:0] num,

output reg isMultipleOf5

);

reg [2:0] lend\_1;

reg [2:0] lend\_2;

reg [2:0] lend\_3;

reg [2:0] lend\_4;

always@(\*) begin

lend\_1 = num[5:3] >= 3'b101 ? num[5:3] - 3'b101 : num[5:3];

lend\_2 = {lend\_1, num[2]} > 3'b101 ? {lend\_1, num[2]} - 3'b101 : {lend\_1[1:0], num[2]};

lend\_3 = {lend\_2, num[1]} > 3'b101 ? {lend\_2, num[1]} - 3'b101 : {lend\_2[1:0], num[1]};

lend\_4 = {lend\_3, num[0]} > 3'b101 ? {lend\_3, num[0]} - 3'b101 : {lend\_3[1:0], num[0]};

if (lend\_4 == 3'b0)

isMultipleOf5 = 1'b1;

else

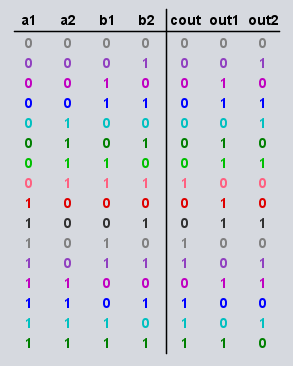
isMultipleOf5 = 1'b0;

end

endmodule

# 思考与练习

1. 我们已经介绍了一个 1bit 的全加器，现在我们希望设计一个 2bits 半加器，其真值表如下图所示：

[](https://soc.ustc.edu.cn/Digital/lab4/figs/2bitadder.png)

请将下面的 Verilog 代码补充完整，以实现预期的功能。注意：不允许直接使用加号和减号！

