**实验14**

**加法器**

# **实验介绍**

在之前的实验中，我们已经学会了组合逻辑电路与时序逻辑电路的设计方法与流程。本次实验中，我们将尝试使用我们所学到的知识，搭建出一个计算机系统中的重要组成部分：ALU（Arithmetic Logic Unit，算术逻辑单元），并通过对运算的不同实现方式的对比，来学习如何在设计中权衡时间性能与资源的使用。

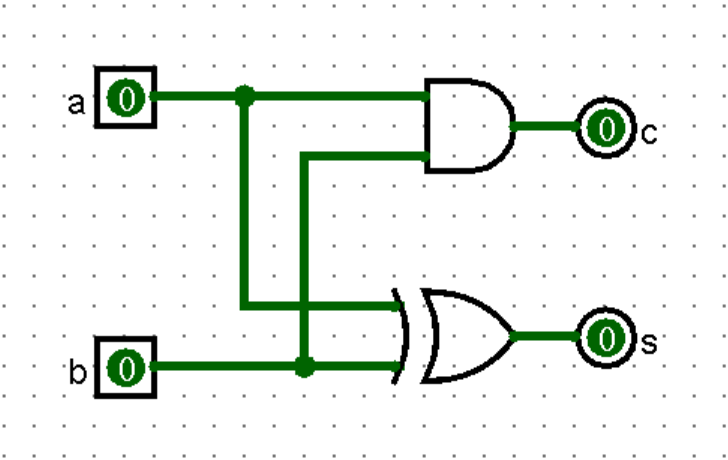
在各种运算中，加法是最简单也最基本的，它的性能高低可能会影响到各个运算的性能。我们曾向大家介绍了基础的 1bit 位宽半加器和全加器。在本次实验中，我们将介绍以半加器与全加器为基础的串行进位加法器，以及一种更高效的加法器——超前进位加法器。

# **实验内容**

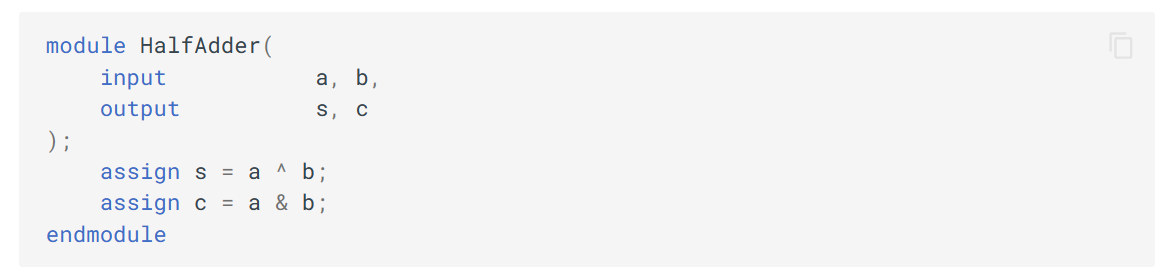
## 1 串行加法器

**1.1 半加器与全加器**

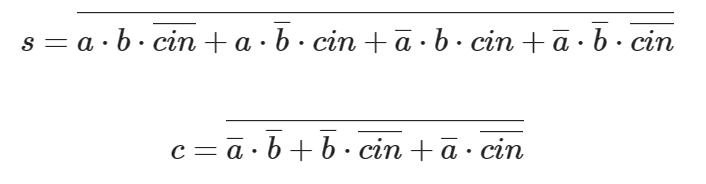
我们知道，1bit 半加器的逻辑电路图为：

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

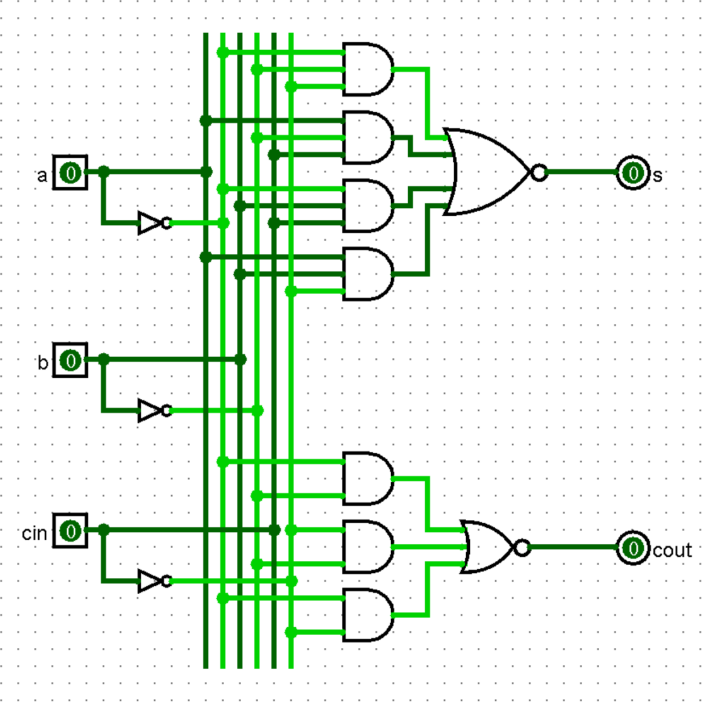
对应的 Verilog 代码如下：



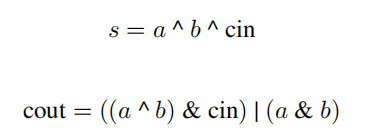
1bit 全加器从真值表中得到的逻辑表达式如下



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)

实际上，上面得到的逻辑表达式可以通过布尔代数进行等价变形。一个更为简洁的形式是：



那么，如何基于此搭建更多位宽的加法器呢？注意到加法运算实际上是从最低位依次向最高位进行的，来自低位的运算结果将参与高位的运算，并获得高位对应的进位结果。我们可以设计如下的 4bits 位宽加法器：



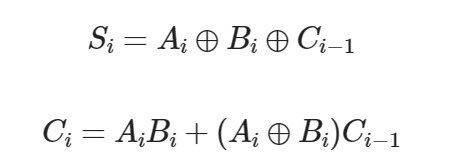
## 2 并行加法器

### ****2.1 超前进位加法器****

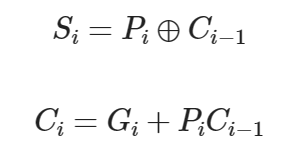
串行进位加法器十分简单，但一点也不高效。对于 32bits 位宽的输入数据，输出结果需要等待低 31 位的加法进位计算完成后才能得到，这带来了极大的电路延迟。超前进位加法器就是为了解决这一问题而设计的。

简单来说，超前进位加法器的原理就是在输入的两个数的每一位上都进行预先的进位，然后再进行加法运算。这样，每位的进位都只由加数与被加数唯一决定，而与来自低位的进位无关，最终加法器的速度不会受到进位信号的限制。

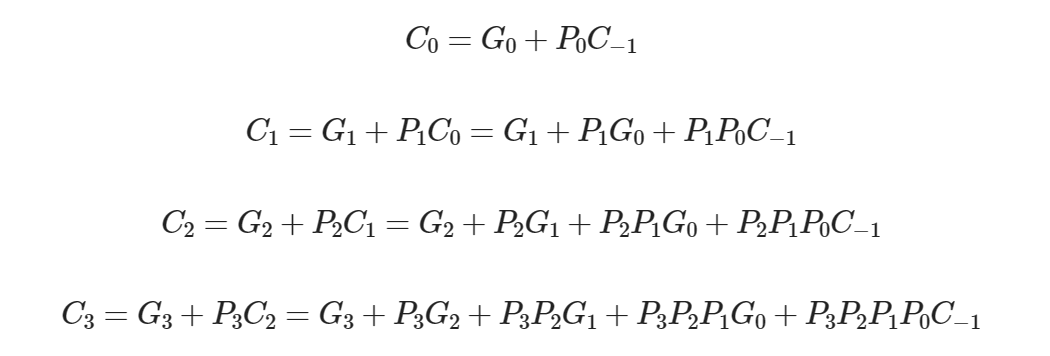
超前进位的公式的推导是通过形式化地表示出加法的每一位的计算结果，再展开进位得到的。下面我们以 4 位超前进位加法器作为例子说明原理及实现。具体来说，考虑计算 A+B 的结果，其中输入的第 i 位记作Ai,Bi，进位为 Ci，和为 Si，则有：



我们尝试做一代换，定义中间变量（注意，这里的中间变量都是可以直接由 A、B 计算出来的）Gi=Ai·Bi，Pi=Ai⊕Bi，则有



这一计算公式更加简洁，也更方便我们将每一位展开为以下形式：



上面的式子中，任意的Ci均只与 Gi,Pi,C−1 有关，而与低位的进位 Ci−1 无关，因此可以先并行计算与式，再对各部求或运算，从而加快了加法器的速度。因此，我们可以得到如下的 Verilog 代码：

module Adder\_LookAhead4 (

input [ 3 : 0] a, b,

input [ 0 : 0] ci,

output [ 3 : 0] s,

output [ 0 : 0] co

);

wire [3:0] C;

wire [3:0] G;

wire [3:0] P;

assign G = a & b;

assign P = a ^ b;

assign C[0] = G[0] | ( P[0] & ci );

assign C[1] = G[1] | ( P[1] & G[0] ) | ( P[1] & P[0] & ci );

assign C[2] = G[2] | ( P[2] & G[1] ) | ( P[2] & P[1] & G[0] ) | ( P[2] & P[1] & P[0] & ci );

assign C[3] = G[3] | ( P[3] & G[2] ) | ( P[3] & P[2] & G[1] ) | ( P[3] & P[2] & P[1] & G[0] ) | ( P [3] & P[2] & P[1] & P[0] & ci );

assign s[0] = P[0] ^ ci;

assign s[1] = P[1] ^ C[0];

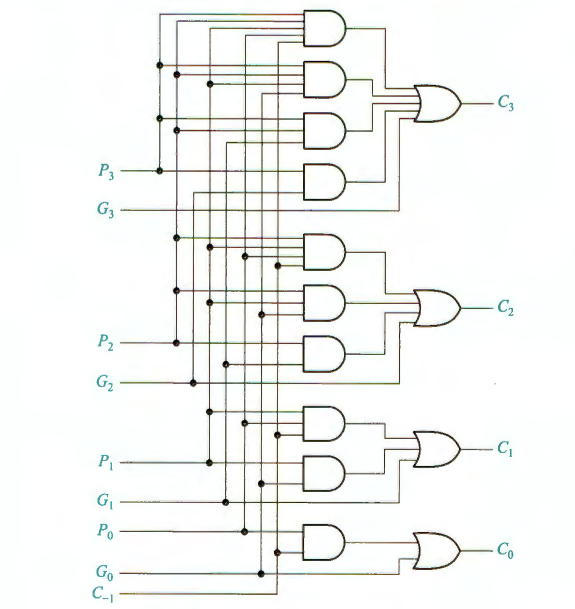
assign s[2] = P[2] ^ C[1];

assign s[3] = P[3] ^ C[2];

assign co = C[3];

endmodule

对应的电路图如下：

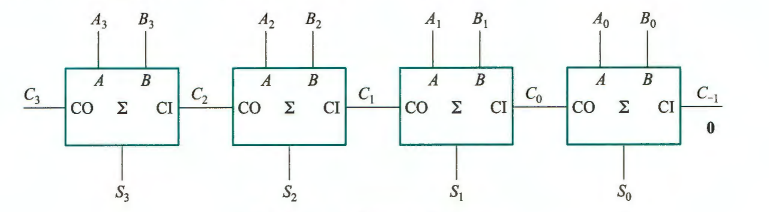
[](https://soc.ustc.edu.cn/Digital/lab6/figs/adder/lookahead.png)

### ****2.2 延时分析****

为什么超前进位加法器的延迟会比普通的加法器低呢？我们这里给出一个简单的理论解释。

如果将每个逻辑门的延迟量化为 t（这里的逻辑门认为是可以接受任意多个输入的），那么对于一个一位全加器，由公式 Si=Ai⊕Bi⊕Ci−1，Ci=AiBi+(Ai⊕Bi)Ci−1 可知，进位信号 Ci 需要经过异或门、与门、或门各一个，总共会产生 3t 的延迟。

那么，对于一个 4 位的串行进位加法器，由于四个全加器前后相接，W每一个的计算都依赖于前一个的进位，所以共需要 12t 的延迟。

[](https://soc.ustc.edu.cn/Digital/lab6/figs/adder/ripper.png)

然而，对于超前进位加法器，可以注意到，计算出 P、G 经过了一级门电路，利用 P、G 和 C−1 计算出 C 的过程中，计算与或式共经过了与、或两级门电路，最后利用 C 与 P 计算出 S 的过程中，又经过了一级异或门电路。因此，总共的延迟为 4t。

超前进位加法器的 4t 的延迟，相比于串行进位的 12t 的延迟，有着显著的降低。此外，可以注意到，超前进位加法器的延迟与位数几乎无关，当位数更多时，其优势更加明显。

然而，以上的理论分析是建立在理想的逻辑门延迟上的，即任意数量输入的逻辑门都只经过 t 的延迟。而实际上，逻辑门的延迟是与其输入的数量有关的，因此，超前进位加法器的延迟并不是完全与位数无关的，但其延迟的增长速度要比串行进位加法器慢得多。

此外，超前进位加法器也并不是位数越大效果越好的。实际上，超前进位加法器在获取高性能的同时并非完全没有代价，它占用的资源更多，需要的面积也更大。这正是硬件设计中常见的时间与空间的权衡问题，这也引出了层次扩展技术的产生。

### ****2.3 层次扩展：从 4 位到 32 位****

正如前面所述，超前进位加法器并非毫无代价的，它是在用空间换取时间上的优势。当我们需要计算的位数足够大，比如 32 位或 64 位时，如果完全使用超前进位加法器的方式来设计，且不谈代码的编写复杂性，其资源使用量将会是串行进位加法器的几十倍甚至上百倍，这显然是不可接受的。

那么，有没有什么方法，能在保证计算延迟降低的同时，又不至于占用过多的资源呢？

答案是肯定的。我们可以模仿串行进位加法器的思路，将多个超前进位加法器首尾相接串联起来。如果说每个四位超前进位加法器的延迟和资源使用量分别为 T 和 S，那么这种设计就可以以 4S 的资源，换取 4T 的延迟——这将显著优于 32 位的串行加法器。

一个 32bits 位宽的层次扩展加法器的电路示意图如下：

[](https://soc.ustc.edu.cn/Digital/lab6/figs/adder/adder.png)

在硬件设计中，我们常常要面临时间与空间的权衡，许多设计在获得较好的时间性能时，都要付出较大的空间牺牲。但是，我们往往可以尝试将高速但耗费资源的设计与节省资源但低速的设计以某种方式结合起来，从而得到一个在时间和空间上都不错的设计方案。本次实验的练习正是对这一设计理念的体现。

# 思考与练习

1. 请参照实验文档中对 4bits 超前进位加法器和层次扩展的描述，设计并编写一个 32bits 加法器。要求分别基于 4bits 超前进位加法器与 8bits 超前进位加法器实现。本题需要实现：

* 4 位超前进位加法器 + 层次扩展成 32 位；
* 8 位超前进位加法器；
* 8 位超前进位加法器 + 层次扩展成 32 位；