306 - 加法器的优化——超前进位加法器（Carry-Lookahead Adder，CLA）

航航大魔王 关注
0.488 2017.07.27 19:26:05 字数 1,521 阅读 18,750

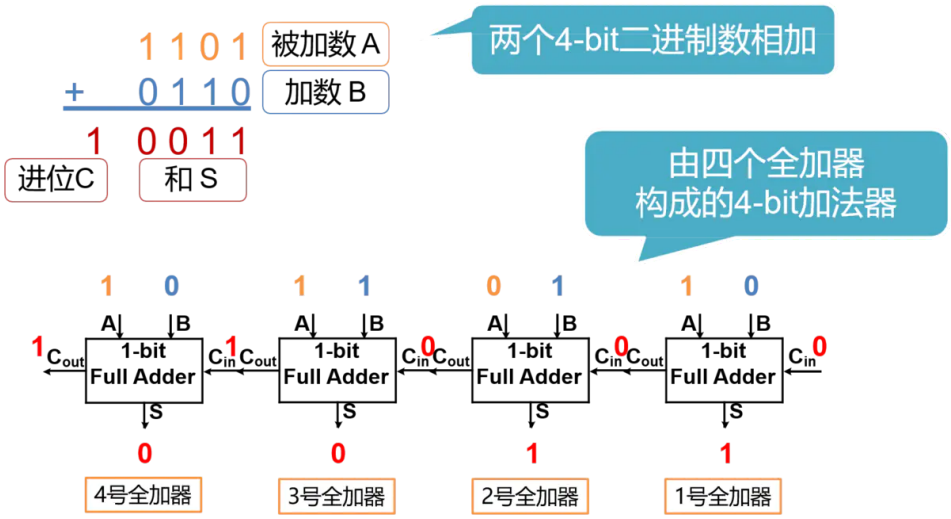
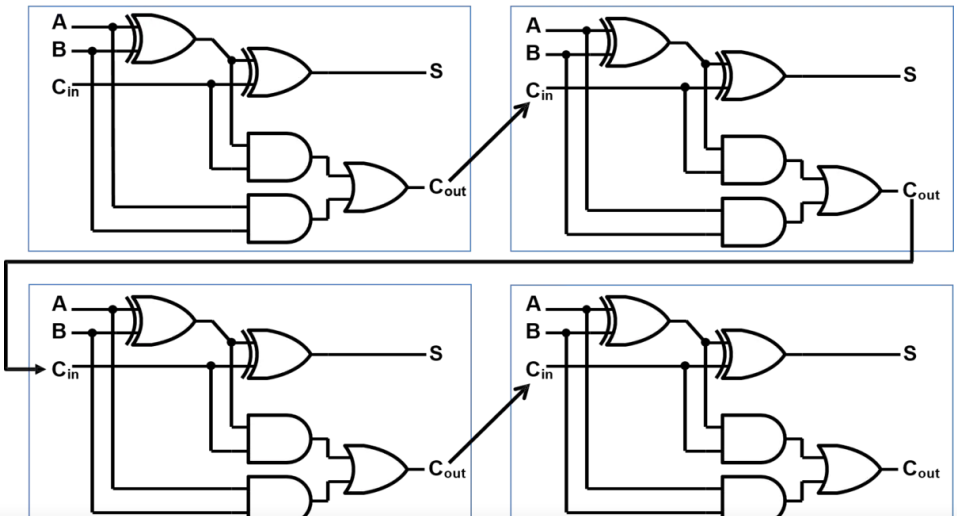


图1 - 4bit全加器原理

上一节我们学习了ALU的加法实现功能部件——全加器，进行两个4bit的二进制数相加，就要用到4个全加器（如图1所示）。那么在进行加法运算时，首先准备好的是1号全加器的3个input。而2、3、4号全加器的Cin全部来自前一个全加器的Cout，只有等到1号全加器运算完毕，2、3、4号全加器才能依次进行进位运算，最终得到结果。 这样进位输出，像波浪一样，依次从低位到高位传递， 最终产生结果的加法器，也因此得名为**行波进位加法器**（Ripple-Carry Adder，RCA）。

RCA的优点是电路布局简单，设计方便， 我们只要设计好了全加器，连接起来就构成了多位的加法器。 但是缺点也很明显，也就是**高位的运算必须等待低位的运算完成**， 这样造成了整个加法器的延迟时间很长。那么，RCA的效率到底如何呢？让我们来算一算：

将4bit的RCA内部结构全部打开，就得到了如图2所示的4-bit RCA的门电路图。要对一个电路的性能进行分析，我们就要找出其中的最长路径。 也就是找出所有的从输入到输出的电路连接中，经过的门数最多的那一条，也称为关键路径（如图3所示）。



推荐阅读

LeetCode-python 1.两数之和
阅读 27

度量学习Loss篇
阅读 320

中央处理器（1）组成与功能
阅读 89

win10 预览窗口显示.py文件内容
阅读 29

剑指offer编程题—数组中只出现一次的树
阅读 28

航航大魔王

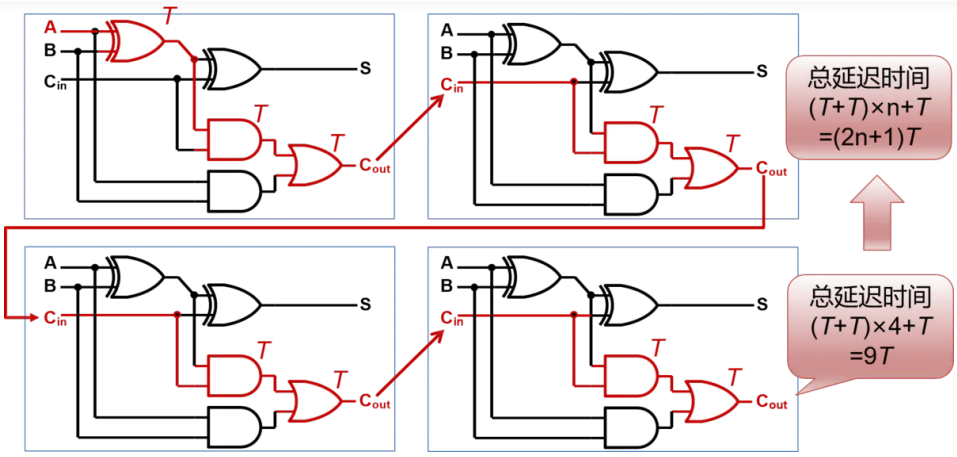


图3 - RCA的关键路径

我们来做一个简单的分析， 对于最低位的全加器，它在A、B和Cin都已经准备好。其实，输入信号进入到这块电路之后，在连接线上传递需要花时间。 称为**线延迟**，而经过这样的门，也需要花时间，称为**门延迟**。 在进行设计原理分析时，我们主要关注门延迟。

从第一个全加器的A-S这条通路来看，产生第一个S输出，需要通过两个门的延迟。 所以它显然不是最长的路径，当然，从A出发或着从B出发都是一样的， 所以对于第一个全加器，它的最长路径，是红色线标记的那条，后面的全加器关键路径同理可得。

那么，假设经过一个门电路的延迟时间为T，那么经过4个全加器所需要的总延迟时间就是：2T x 4 + T(第一个全加器产生3个T) = 9T。所以推出，经过n个全加器所产生的总延迟时间为2T x n + T = (2n+1)T。

对于一个32bit的RCA，有总延迟时间：(2n+1)T =(2×32+1)×T =65T，这是什么概念呢？举个例子，iPhone 5s的A7 SoC处理器采用28nm制造工艺，主频1.3GHz（0.66ns）。按照这个工艺水平，门延迟T设为0.02ns，那么32-bit RCA的延迟时间为1.3ns ，时钟频率为769MHz，远超A7处理器的主频延迟时间，更别说这个32bit的RCA只是一个加法运算器，更更别说，我们在计算过程中只考虑了门延迟，还有线延迟等各种延迟没有加入计算.....

所以RCA的效率绝对是个问题。那么，有没有办法优化呢？RCA的主要问题是高位的运算必须等待低位的"进位输出信号"，那我们的优化思路就是'能否提前计算出"进位输出信号'？"

参见图2，我们可以用前一个全加器的参数来表示后面的进位输出（Cout），即：

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$
$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$

设：

- 生成（Generate）信号：$G_i = A_i \cdot B_i$
- 传播（Propagate）信号：$P_i = A_i + B_i$

则：$C_{i+1} = G_i + P_i \cdot C_i$

由此来表示4个全加器的进位输出为：

推荐阅读

LeetCode-python 1.两数之和
阅读 27

度量学习Loss篇
阅读 320

中央处理器（1）组成与功能
阅读 89

win10 预览窗口显示.py文件内容
阅读 29

剑指offer编程题—数组中只出现一次的树
阅读 28

航航大魔王

$C_2 = G_1 + P_1 \cdot C_1$

$= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$

$= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

$C_{i+1} = G_i + P_i \cdot C_i$

$C_3 = G_2 + P_2 \cdot C_2$

$= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0)$

$= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$

$C_4 = G_3 + P_3 \cdot C_3$

最终我们需要得到的是C4，经过换算，C4=G3+P3·G2+P3·P2·G1+P3·P2·P1·G0+P3·P2·P1·P0·C0，而这些参数，全部已知！并不需要前一个全加器运算输出，由此我们得到了提前计算进位输出的方法， 用这样的方法实现了加法器就被称为**超前进位加法器**（Carry-Lookahead Adder，CLA）。

根据上面的优化算法，我们重新绘制了CLA的布线方式（如图4）：

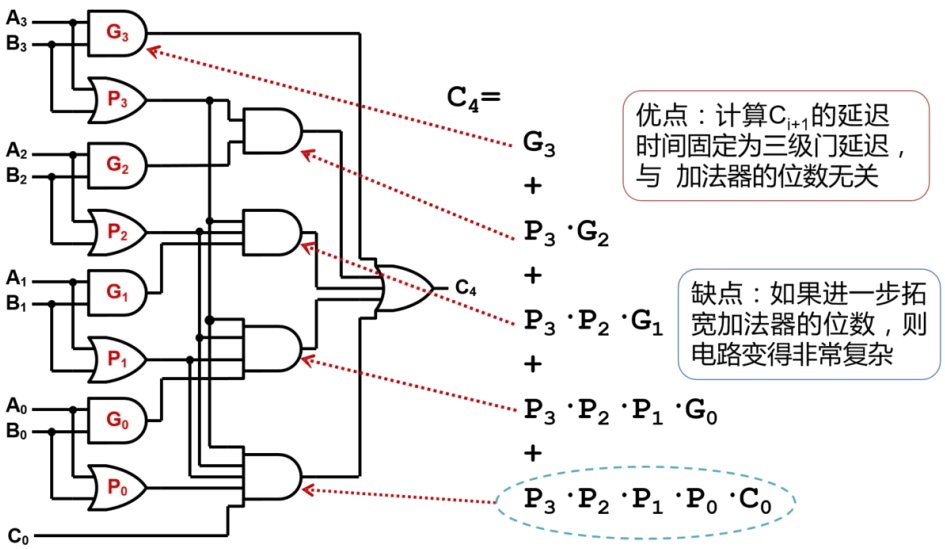


图4 - CLA原理图

图5

那么使用CLA来进行加法运算的效率如何呢？还是按照Apple A7处理器的工艺水平，单个CLA的延迟为0.08ns，4级CLA的延迟为0.26ns，时钟频率3.84GHz，都远远小于主频的延迟，完全符合标准。然而，由图可见，计算4bit的二进制数，就要平行排列4个全加器，那么要是计算8bit，

推荐阅读

LeetCode-python 1.两数之和
阅读 27

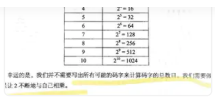
度量学习Loss篇
阅读 320

中央处理器（1）组成与功能
阅读 89

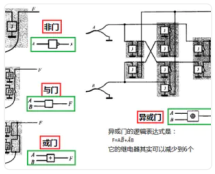
win10 预览窗口显示.py文件内容
阅读 29

剑指offer编程题—数组中只出现一次的树
阅读 28

fatfatEddy 阅读 780 评论 0 赞 2



从计算机原理的角度展望人工智能的前景



单从技术史的角度看计算机的发展史，似乎那么多形成条件是偶合的，不免惊叹于它的奇妙，但如果从产品的角度梳理它的原理又...

行不易一 阅读 929 评论 8 赞 37

生活不拘泥于格式。

周日上午十一点，我在上班。你，在做什么？ 时间过了很久，你没有回答。 作罢。 二零一五年已然进入了尾声，年末之际，...

哎呦喂_梦丢了 阅读 111 评论 0 赞 0

推荐阅读

LeetCode-python 1.两数之和
阅读 27

度量学习Loss篇
阅读 320

中央处理器（1）组成与功能
阅读 89

win10 预览窗口显示.py文件内容
阅读 29

剑指offer编程题—数组中只出现一次的树
阅读 28

-	RCA	CLA
结构特点	低位全加器的Cout连接到高一位全加器Cin	每个全加器的进位输入并不来自于前一级的全加器，而是来自超前进位的逻辑
优点	电路布局简单，设计方便	计算Ci+1的延迟时间固定为三级门延迟，与加法器的位数无关
缺点	高位的运算必须等待低位的运算完成，延迟时间长	如果进一步拓宽加法器的位数，则电路变得非常复杂

32位的加法器如果采用行波进位的方式，我们已经分析过需要65级的门延迟， 那如果采用超前进位的方式，理想情况下也只需要四级的门延迟，但可惜的是， 这也只是一个理想。因为要实现32位的完全的超前进位，电路就会变得非常的复杂。 因此通常的实现方法， 是采用多个小规模的超前进位加法器拼接而成一个较大的加法器，例如，用4个8-bit的超前进位加法器连接成32-bit加法器。

航航大魔王 万般皆下品，唯有读书高。
总资产2 (约0.24元) 共写了2.8W字 获得81个赞 共72个粉丝
关注

被以下专题收入，发现更多相似内容

计算机组成原理

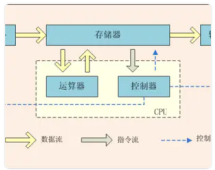
推荐阅读

更多精彩内容 >

造个计算机--1、设计运算器

话接上文： 造个计算机 废话少说，先聊聊我们要造的这个计算机的硬件结构，本项目（整体计划的实现，对于个人来说算个小...

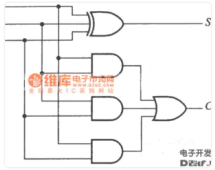
老鱼 阅读 4,518 评论 11 赞 61



半加器、全加器及其应用

半加器、全加器是组合电路中的基本元器件，也是CPU中处理加法运算的核心，理解、掌握并熟练应用是硬件课程的最基本要求...

CodingTech 阅读 5,585 评论 0 赞 3



推荐阅读

LeetCode-python 1.两数之和
阅读 27
度量学习Loss篇
阅读 320
中央处理器（1）组成与功能
阅读 89
win10 预览窗口显示.py文件内容
阅读 29
剑指offer编程题—数组中只出现一次的树
阅读 28