PPA: 从入门到放弃

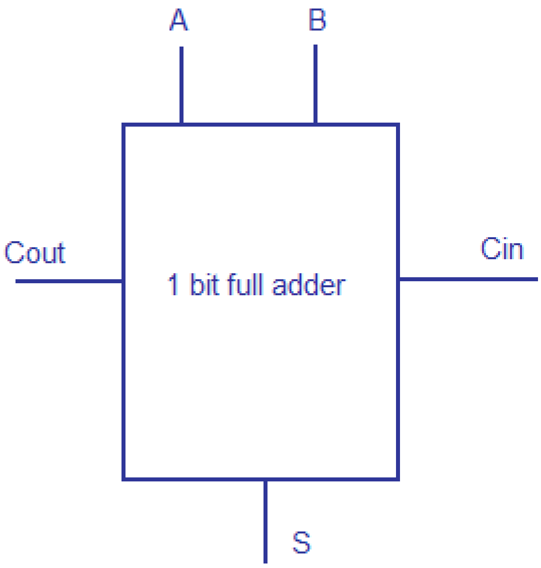
本次介绍电路设计中的PPA (Parallel Prefix Adder)，该技术可以高效求布尔状态下的2-输入加法，用于安全多方计算中算术分享对布尔分享的转化。接下来首先介绍Full Adder (FA) 和基于 FA 构造的 RCFA。进一步介绍PPA的构造。

0. 入门

0.1 Full Adder

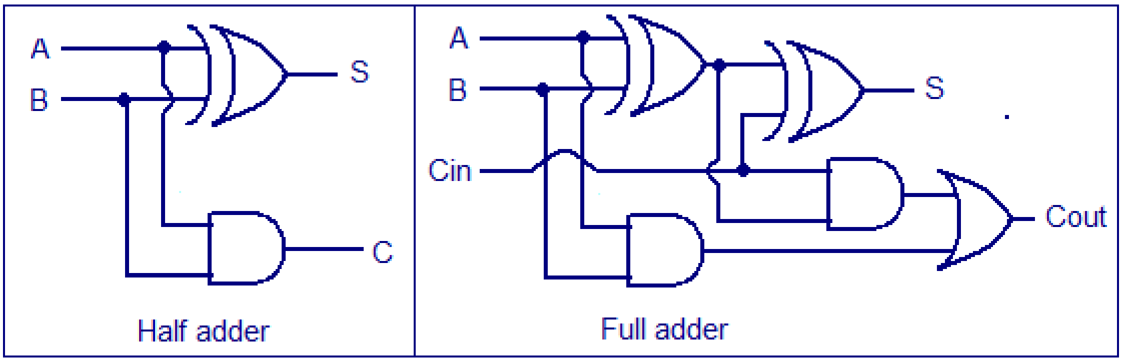
首先考虑对于1比特的输入加法：如图所示，FA 的输入有三个：A，B，和 C_{in}。其中，A 和 B 代表输入，C_{in} 表示低位的进位。输出中，$S = A \oplus B \oplus C_{in}$ 表示本位置的结果，$C_{out} = (A \cdot B) \vee ((A \oplus B) \cdot C_{in})$ 表示对高位的进位。其真值表和逻辑门表示如下。

Inputs			Outputs	
A	B	Cin	Cout	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



1 bit full adder truth table & schematic

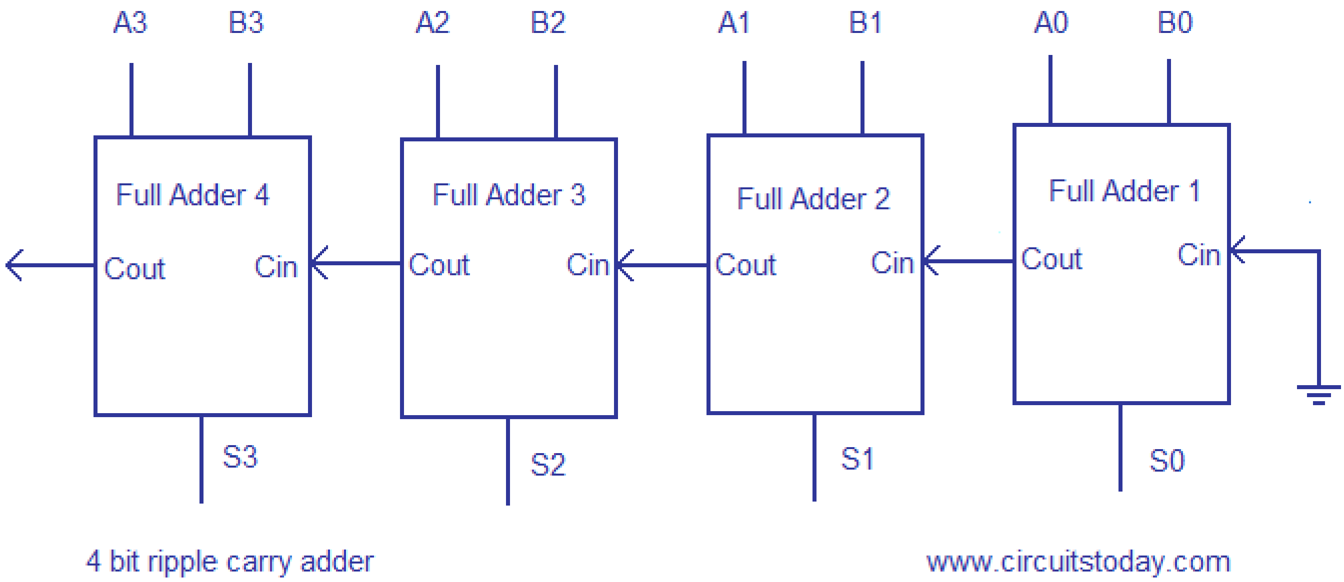
下图中，half adder表示对于两输入的加法电路求 S 和 C，而右图则表示FA具体利用AND、 XOR 和 OR 门构造方法：



从上图可以看出，half adder 需要1层AND门，而FA则包含了2层AND门（OR中包含AND）。

0.2 RCFA

基于FA，将多个FA串联则可以构造RCFA。如下图所示，RCFA可以在按比特操作，求 $A+B$。其中，最低位的 $C_{in}=0$。然而，直接利用 RCFA 在布尔电路下求两个 ℓ 比特的输入之和，需要的乘法为 $O(\ell)$。

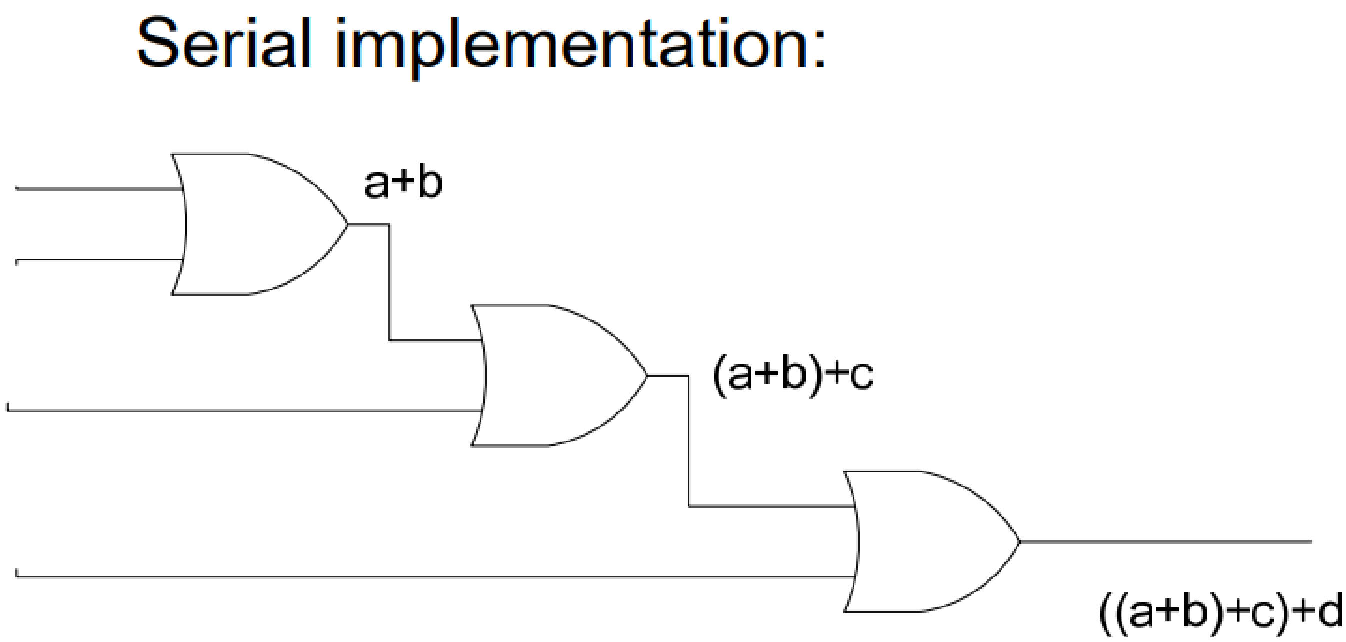


1. 放弃

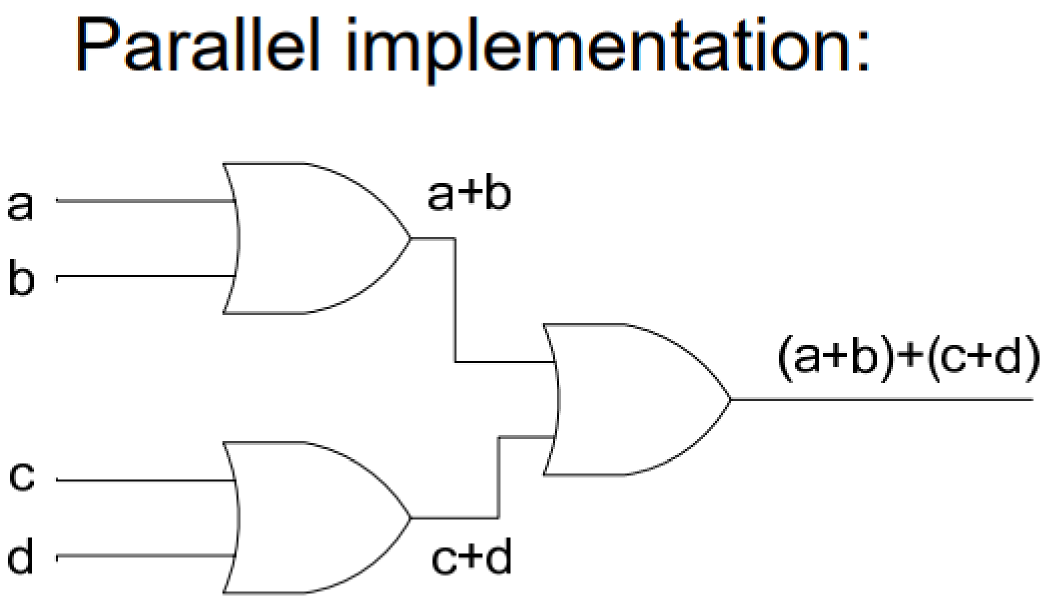
为了优化AND门的深度，我们接下来介绍 PPA。

1.1 Prefix Sum

在介绍PPA之前，首先思考计算 $a+b+c+d$ (此处 $+$ 表示 OR 门)。一种序列化的计算方法的计算电路如下：

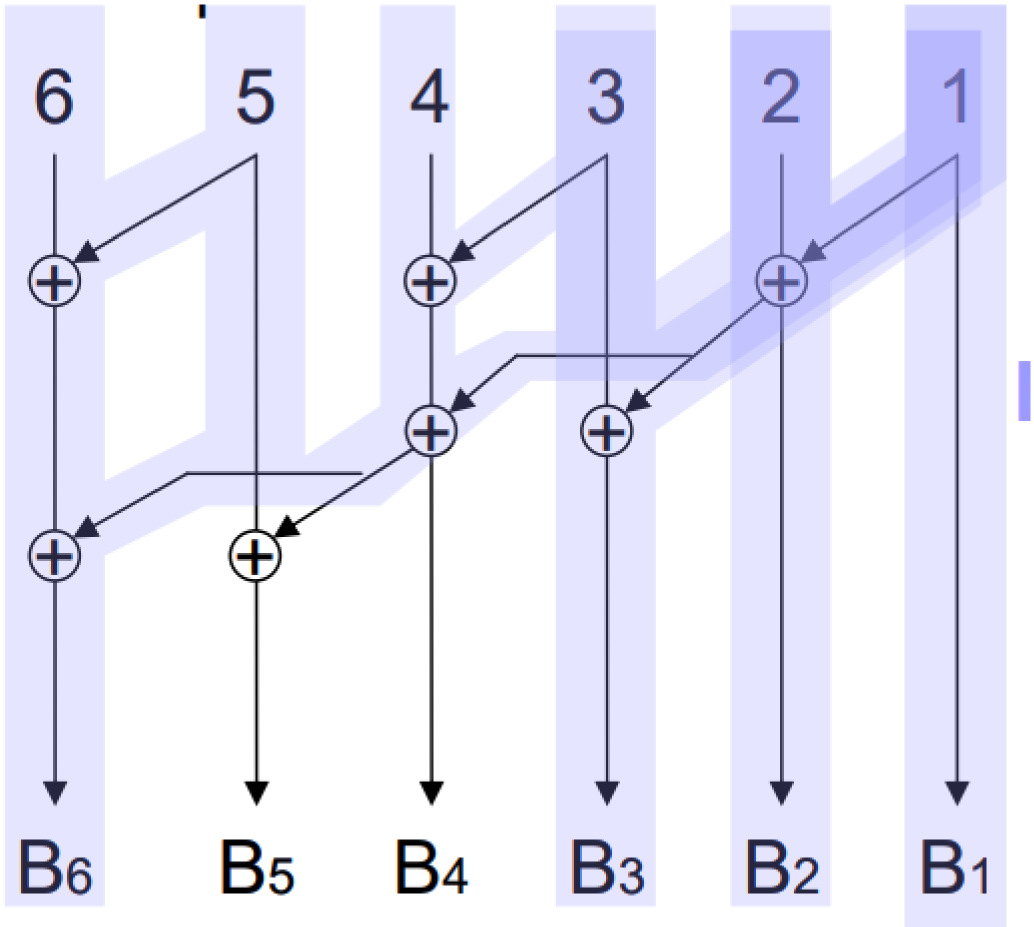


而又因为 $a+b+c+d=(a+b)+(c+d)$，可以有并行方案：



可以看到，并行的方案电路深度比序列化方法要浅，计算更加高效。

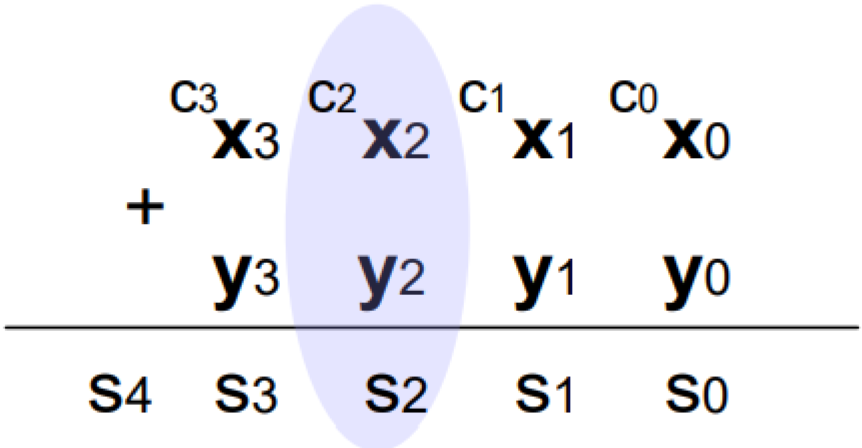
进一步，思考前缀和 (prefix sum)，即对于字符串 $A=6\ 5\ 4\ 3\ 2\ 1$，其前缀和为 $B=21\ 15\ 10\ 6\ 3\ 1$，而 B 也可以并行计算，逻辑示意图如下：



可以看到，在第二层可以并行计算三个 $\oplus$，而 B_2 的计算结果可以同时加入 B_3 和 B_4 的计算。其他位置类似。

1.2 Carry Look Ahead Adder (CLA)

深入分析二进制加法：



首先，在求 c_i 的时候，需要考虑 x_i，y_i 和 c_{i-1}。具体来说，真值表如下：

x_i	y_i	c_i
0	0	0
0	1	c_{i-1}

x_i	y_i	c_i
1	0	c_{i-1}
1	1	1

定义

$$p_i = x_i \oplus y_i,\ g_i = x_i \cdot y_i$$

有

$$c_i = g_i + p_i \cdot c_{i-1} = x_i \cdot y_i + (x_i \oplus y_i) \cdot c_{i-1},$$

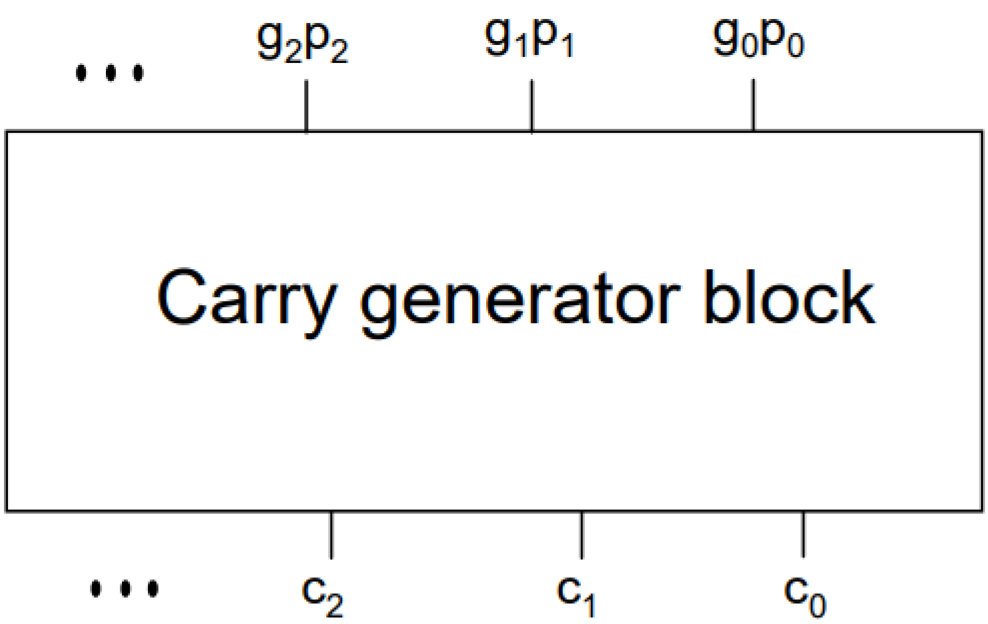
其中，p_i 被称为 \textit{propagate bit}，g_i 被称为 \textit{generate bit}。

而最终的计算结果，可以由 p_i 和 c_i 计算得到。

因此， CLA 的计算流程如下：

1. 对每一步，预计算 g_i 和 p_i；
2. 对每一步，计算 c_i；
3. 根据 p_i 和 c_i 计算和 s_i；

例如，对于下图三步计算：



有:

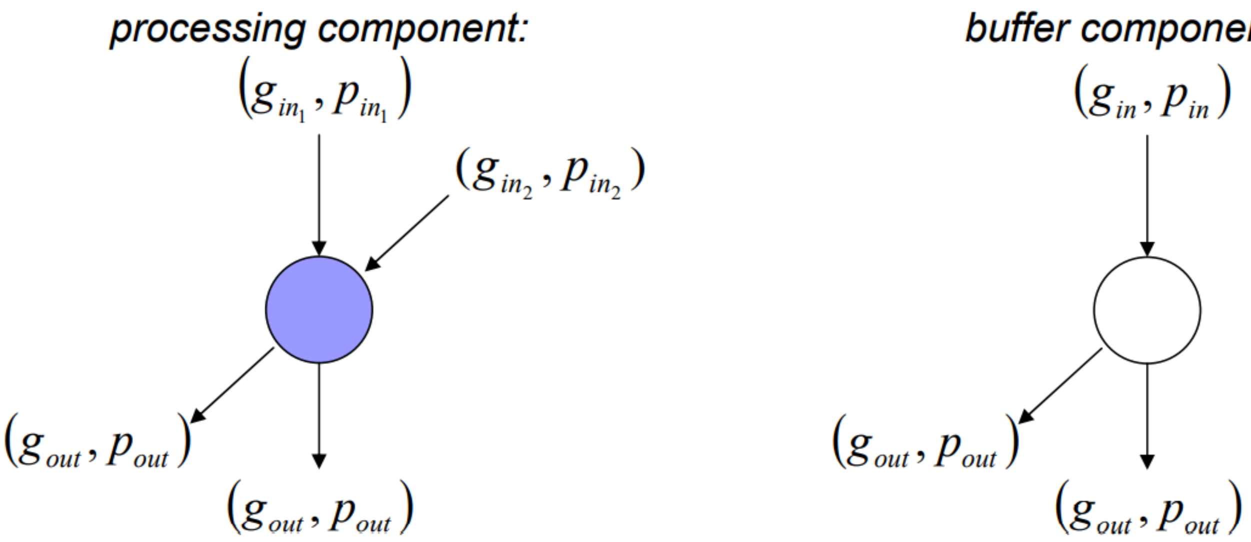
1. $c_0 = g_0$;
2. $c_1 = g_1 + p_1 \cdot g_0$;

3. $c_2 = g_2 + p_c \cdot c_1 = g_2 + p_2 \cdot (g_1 + p_1 \cdot g_0) = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0$；
4. ...
5. 最终 $s_i = p_i \oplus c_{i-1}$。

1.3 PPA

为了减少AND门的深度，PPA对CLA进行了进一步优化。不过PPA和CLA进行的计算流程大致一致，只是在计算 c_i 的时候进行了充分的并行优化。

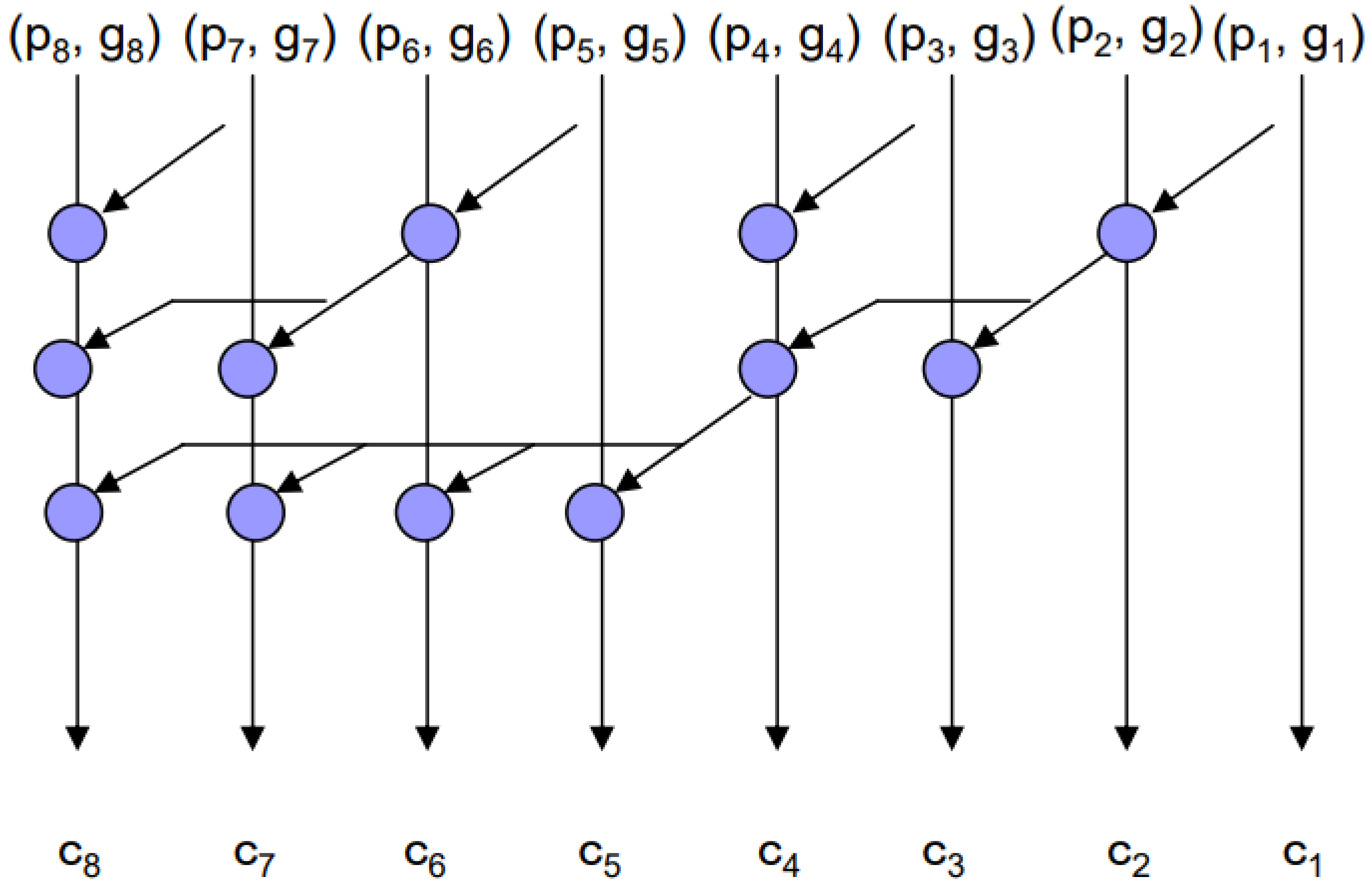
在PPA的设计中，主要有两种结构组件：processing component 和 buffer component，两种结构的构造和逻辑意义如下：

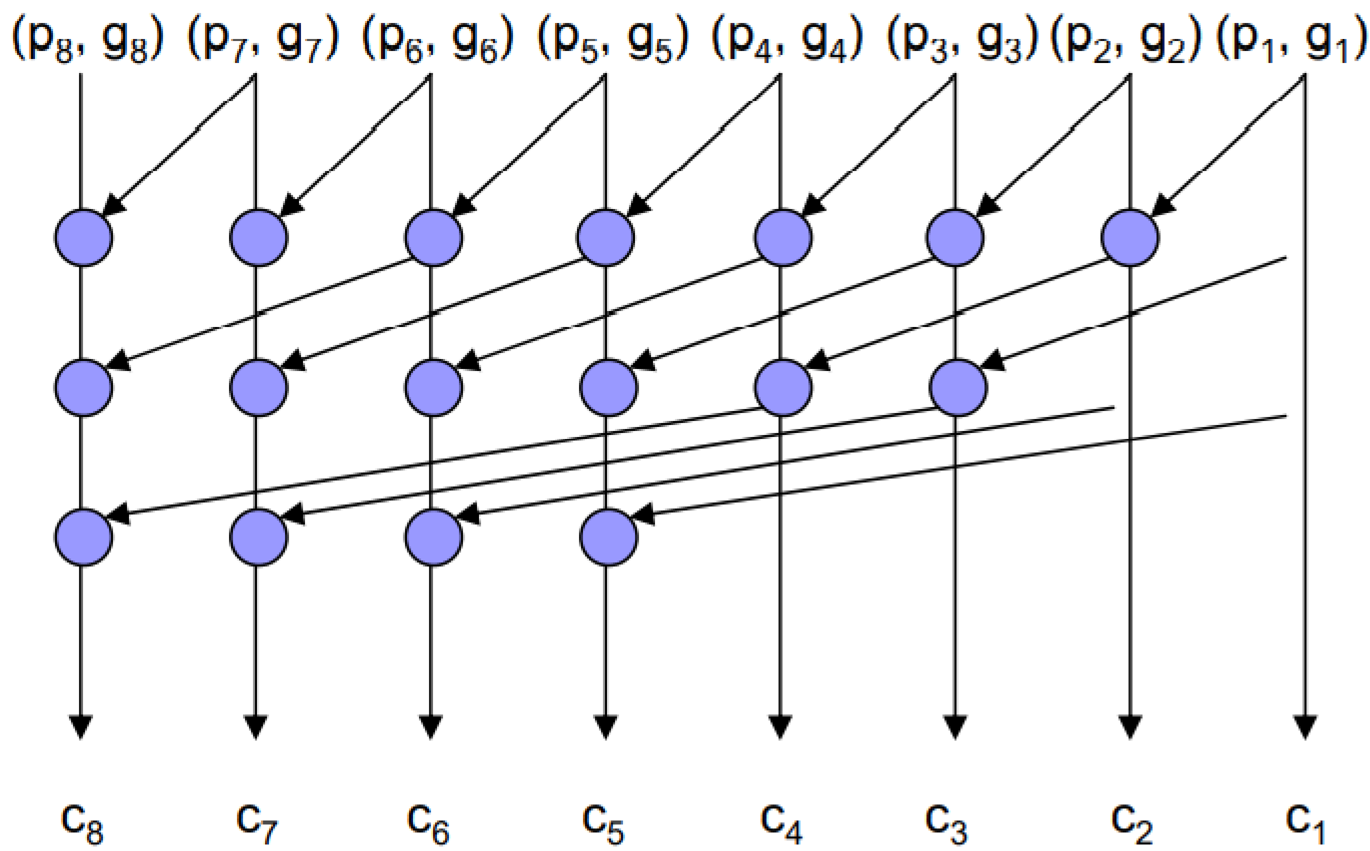


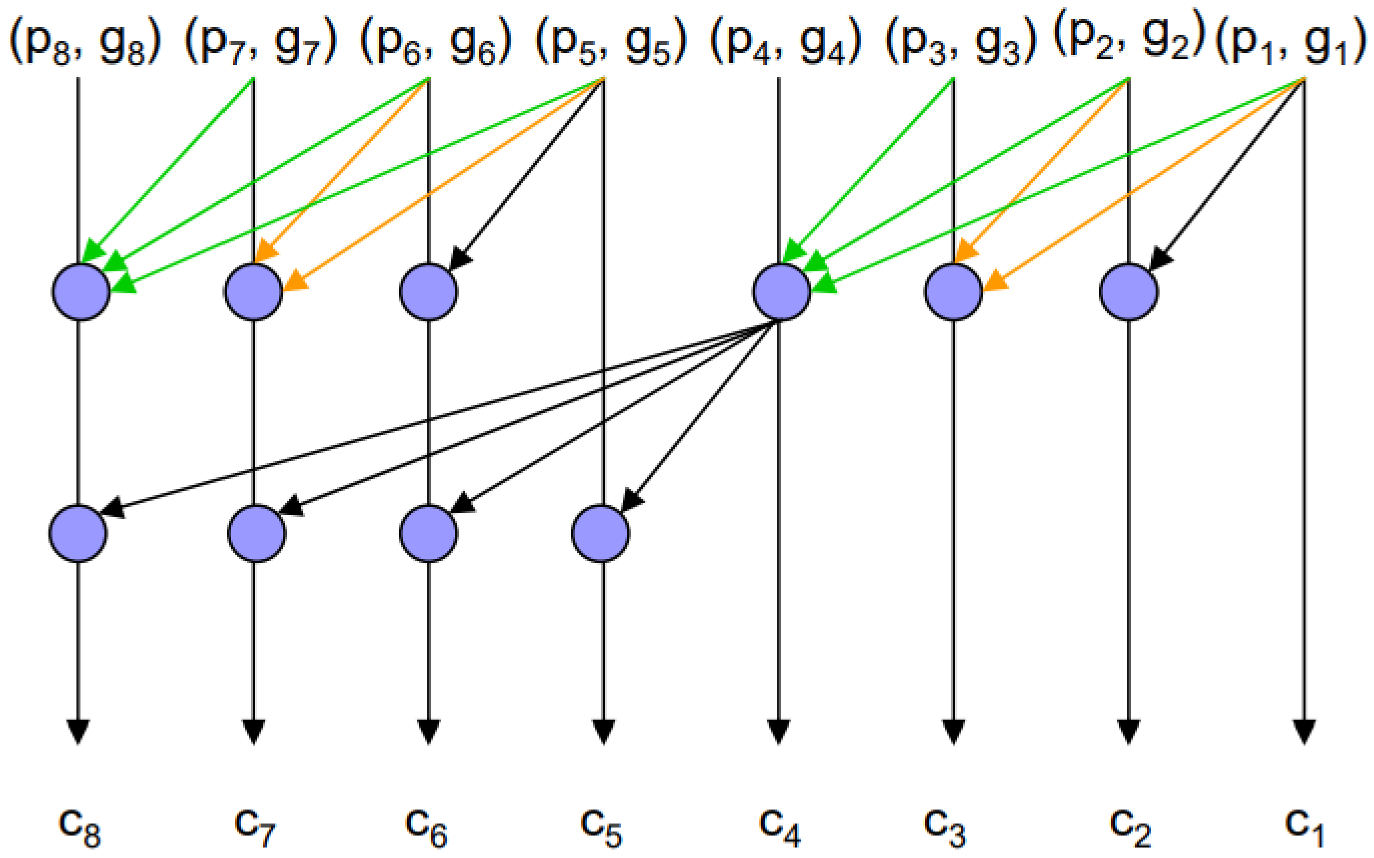
$$(g_{out}, p_{out}) = (g_{in_1} + p_{in_1} \cdot g_{in_2}, p_{in_1} \cdot p_{in_2})$$

$$(g_{out}, p_{out}) = (g_{in}, p_{in})$$

目前的多种PPA变体，其主要设计思路是在加法器范围、电路深度、节点输出数量和整体布线四点上做出均衡。常见的变体有：


(p_8, g_8) (p_7, g_7) (p_6, g_6) (p_5, g_5) (p_4, g_4) (p_3, g_3) (p_2, g_2) (p_1, g_1)
c_8 c_7 c_6 c_5 c_4 c_3 c_2 c_1


(p_8, g_8) (p_7, g_7) (p_6, g_6) (p_5, g_5) (p_4, g_4) (p_3, g_3) (p_2, g_2) (p_1, g_1)
c_8 c_7 c_6 c_5 c_4 c_3 c_2 c_1



1.4 Application

介绍了这么多关于PPA的知识，PPA在安全多方计算中主要用在A2B过程中，减少AND门深度，从而减少通信轮数。以ABY3中的A2B过程为例简单介绍一下 PPA 的应用：

1. 首先 P_0, P_1 和 P_2 分别拥有 (x_0, x_1)， (x_1, x_2)，和 (x_2, x_0)；
2. 首先并行调用 ℓ 次独立的 $FA(x_0[i], x_1[i], x_2[i]) \to (c[i], s[i])$，将 $x_2[i]$ 理解为FA 的进位；
3. 进一步，利用PPA计算 $2[c]^B + [s]^B$， $2[c]^B$ 是因为 $c[i]$ 需要进位到 $s[i+1]$。
   需要 $O(\log_2 \ell)$ 次通信。

上述方案可以应用于半诚实和而已模型。如果只考虑半诚实模型，可以让 P_0 本地计算并进行布尔分享得到 $[x_0 + x_1]^B$，并进一步利用PPA计算 $[x_0 + x_1]^B + [x_2]^B$。在实现中，TF-encrypted 中 使用了 Sklansky结构和 Kogge-Stone结构。

而ABY2.0中则使用了Beaumont-Smith结构，并利用其提出的多输入乘法门进行优化。

参考资料

1. https://en.wikipedia.org/wiki/Kogge–Stone_adder
2. https://www.circuitstoday.com/ripple-carry-adder
3. https://users.encs.concordia.ca/~asim/COEN_6501/Lecture_Notes/Parallel prefix adders presentation.pdf