

算术逻辑部件 设计

蒋小龙

2001.7.5

声 明

作此资料为本人个人行为，此资料版权为本人所有。

你可以任意使用，但你非经本人允许不得对此资料内容作任何修改。

你因使用此资料所带来任何收益，本人均不染指；因使用此资料所引起的任何不良后果，本人不承担任何形式的责任。

出版物引用，请注明！

蒋 小 龙

2001.7.5

目 录

声明	1
0、约定	3
1、一位加法器	4
2、4 位加法器	4
3、4 位超前进位链	5
4、16 位超前进位链	6
5、更多位超前进位链	8
6、算术运算设计	9
7、逻辑操作设计	10
8、标识位	10
9、设计示例 1 —— 16 位 7 功能算术逻辑部件	12
10、设计示例 2 —— 4 位 16 功能算术逻辑部件	15
后记	20
个人介绍	21

0、约定

运算数据:

- A: 操作数、位串;
- B: 操作数、位串;
- D: 结果数据、位串, 与操作数位宽同;
- Ci: 进位、一位, 以前操作所产生, 本次操作视情况考虑;
- Co: 进位、一位, 本次操作产生;
- C: 进位、位串, 由操作数生成的进位, 不会比操作数位宽小;
- G: 进位生成、位串, 与操作数位宽同, 具体意义见文中;
- P: 进位传输、位串, 与操作数位宽同, 具体意义见文中。

运算符:

- | | |
|-----------------------------------|-----------|
| +: 对其两边的数据作加法操作; | $A + B$; |
| -: 用左边的数据减去右边的数据作; | $A - B$ |
| -: 对跟在其后的数据作取补操作, 即用 0 减去跟在其后的数据; | $- B$ |
| &: 对其两边的数据按位作与操作; | $A \& B$ |
| #: 对其两边的数据按位作或操作; | $A \# B$ |
| @: 对其两边的数据按位作异或操作; | $A @ B$ |
| ~: 对跟在其后的数据作按位取反操作; | $\sim B$ |

对运算符的约定只在文中论述的时候有效。

设计示例用 VerilogHDL 语言实现, 所有的东西符合 VerilogHDL 语法。如有疑问, 请参阅 VerilogHDL 资料。

1、一位加法器

考虑一位全加器。如右图示真值表，D、Co 逻辑表达式为：

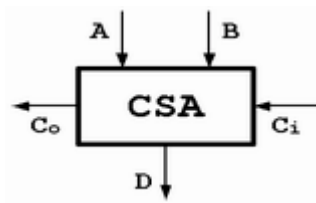
$$D = A \oplus B \oplus C_i \quad (1-1)$$

$$C_o = A \& B \# A \& C_i \# B \& C_i \quad (1-2)$$

$$= A \& B \# (A \# B) \& C_i \quad (1-3)$$

不难作出其逻辑图，此处省略。

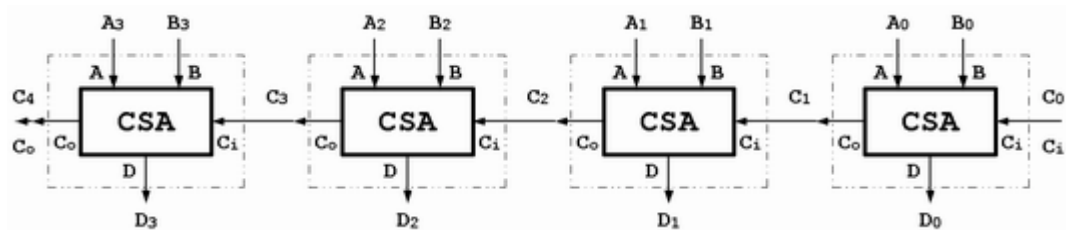
一位全加器又称“保留进位加法器”。因其简单，是研究其它高性能、高速加法器的基础。因其简单、快速，是构成其它高速处理部件的基本元件。其符号如右。



A	B	C _i	D	C _o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

由真值表可知，若将各位取反，并无影响。

2、4 位加法器



四位加法器可以采用四个一位全加器级连成串行进位加法器。如上图示：

对于这种方式，因高位运算必须等低位进位来到后才能进行。因此，它的延迟非常可观。高速运算肯定无法胜任。

对串行进位加法器研究可得：运算的延迟是由于进位的延迟。基于此，减小进位的延迟非常有效。

下面讨论的超前进位链能有效减少进位的延迟，它由进位门产生进位，各进位彼此独立，不依赖于进位传播。因此，它的延迟非常小、速度非常高。

既然进位已经解决，则加法器值

$$D = A \oplus B \oplus C \quad (2-1)$$

不存在问题。

由此可见，进位的解决是核心。

3、4 位超前进位链

研究式(1-3)

$$C_0 = A \& B \# (A \# B) \& C_i \quad (3-1)$$

可以看出：若 A、B 均为 1，则产生进位输出；若 A、B 存在 1，则进位输出依赖于低位进位 C_i 。换一种说法：若 A、B 均为 1，则产生进位；若 A、B 存在 1，则传输(低位)进位(C_i)。

A、B、 C_i 在本位运算，产生本位值 D，向高位进位 C_0 。更一般的：

$$D_n = A_n \oplus B_n \oplus C_n \quad (3-2)$$

$$C_{n+1} = A_n \& B_n \# (A_n \# B_n) \& C_n \quad (3-3)$$

令：

$$G_n = A_n \& B_n \quad (3-4)$$

$$P_n = A_n \# B_n \quad (3-5)$$

则式(3.-3)为：

$$C_{n+1} = G_n \# P_n \& C_n \quad (3-6)$$

这样，就引入了进位产生函数(G)、进位传输函数(P)。其意义为：若 G_n 为 1，必定产生进位；若 P_n 为 1，则向高位传输(低位)进位(C_n)，可认为低位进位越过本位直接向高位进位。此处是否说明仍存在进位传播，使进位延迟无法减小？

以 4 位超前进位链为例：

$$C_0 = C_i$$

$$\begin{aligned} C_1 &= G_0 \# P_0 \& C_0 \\ &= G_0 \# P_0 \& C_i \end{aligned} \quad (3-7)$$

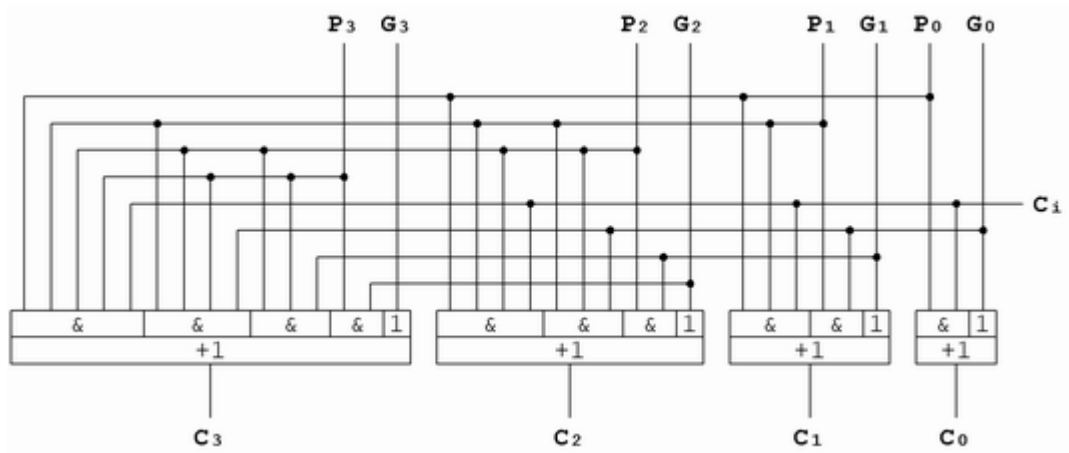
$$\begin{aligned} C_2 &= G_1 \# P_1 \& C_1 \\ &= G_1 \# P_1 \& G_0 \# P_0 \& C_i \end{aligned} \quad (3-8)$$

$$\begin{aligned} C_3 &= G_2 \# P_2 \& C_2 \\ &= G_2 \# P_2 \& G_1 \# P_1 \& G_0 \# P_0 \& C_i \end{aligned} \quad (3-9)$$

$$\begin{aligned} C_4 &= G_3 \# P_3 \& C_3 \\ &= G_3 \# P_3 \& G_2 \# P_2 \& G_1 \# P_1 \& G_0 \# P_0 \# \\ &\quad P_3 \& P_2 \& P_1 \& P_0 \& C_i \end{aligned} \quad (3-10)$$

$$C_0 = C_4$$

其逻辑图如下：



可见，将迭代关系去掉，则各位彼此独立，进位传播不复存在。因此，总的延迟是两级门的延迟。其高速也就自不待言。

对式(2-1)

$$D = A @ B @ C \quad (3-11)$$

$$= ((\sim A) \& B \# A \& (\sim B)) @ C$$

$$= ((\sim A) \& B \# A \& (\sim B) \# (\sim A) \& A \# (\sim B) \& B \# A \& (\sim A) \& B \& (\sim B)) @ C$$

$$= ((\sim A) \& (A \# B) \# (\sim B) \& (A \# B) \# A \& B \& (\sim A) \& (\sim B)) @ C$$

$$= (((\sim A) \# (\sim B)) \& (A \# B) \# A \& B \& (\sim (A \# B))) @ C$$

$$= ((\sim (A \& B)) \& (A \# B) \# A \& B \& (\sim (A \# B))) @ C$$

$$= (A \& B) @ (A \# B) @ C \quad (3-12)$$

$$= G @ P @ C \quad (3-13)$$

可见：

$$D = A @ B @ C = G @ P @ C = (A \& B) @ (A \# B) @ C \quad (3-14)$$

此特性对算术逻辑部件的设计非常有用！

4、16 位超前进位链

对 4 位超前进位加法器：二输入数据 A、B，进位输入 C_i ，生成数据 D，进位 C_o 。因此，可以用 4 个 4 位超前进位进位链单元串接形成 16 位进位链。这存在问题，就是单元内虽是超前进位，但单元间却是串行，存在进位传播问题，延迟不会短，速度也就高不了！

从 4 位超前进位链得到启示，能否在单元间实行超前进位？

考察第 4、8、12、16 位的进位，会得到：

$$\begin{aligned} C_4 &= G_3 \# P_3 \& C_3 \\ &= (G_3 \# P_3 \& G_2 \# P_3 \& P_2 \& G_1 \# P_3 \& P_2 \& P_1 \& G_0) \# \\ &\quad (P_3 \& P_2 \& P_1 \& P_0) \& C_i \end{aligned} \quad (4-1)$$

$$\begin{aligned} C_8 &= G_7 \# P_7 \& C_7 \\ &= (G_7 \# P_7 \& G_6 \# P_7 \& P_6 \& G_5 \# P_7 \& P_6 \& P_5 \& G_4) \# \\ &\quad (P_7 \& P_6 \& P_5 \& P_4) \& (G_3 \# P_3 \& G_2 \# P_3 \& P_2 \& G_1 \# P_3 \& P_2 \& P_1 \& G_0) \# \\ &\quad (P_7 \& P_6 \& P_5 \& P_4) \& (P_3 \& P_2 \& P_1 \& P_0) \& C_i \end{aligned} \quad (4-2)$$

$$\begin{aligned} C_{12} &= G_{11} \# P_{11} \& C_{11} \\ &= (G_{11} \# P_{11} \& G_{10} \# P_{11} \& P_{10} \& G_9 \# P_{11} \& P_{10} \& P_9 \& G_8) \# \\ &\quad (P_{11} \& P_{10} \& P_9 \& P_8) \& (G_7 \# P_7 \& G_6 \# P_7 \& P_6 \& G_5 \# P_7 \& P_6 \& P_5 \& G_4) \# \\ &\quad (P_{11} \& P_{10} \& P_9 \& P_8) \& (P_7 \& P_6 \& P_5 \& P_4) \& \\ &\quad \quad (G_3 \# P_3 \& G_2 \# P_3 \& P_2 \& G_1 \# P_3 \& P_2 \& P_1 \& G_0) \# \\ &\quad (P_{11} \& P_{10} \& P_9 \& P_8) \& \\ &\quad \quad (P_7 \& P_6 \& P_5 \& P_4) \& (P_3 \& P_2 \& P_1 \& P_0) \& C_i \end{aligned} \quad (4-3)$$

$$\begin{aligned} C_{16} &= G_{15} \# P_{15} \& C_{15} \\ &= (G_{15} \# P_{15} \& G_{14} \# P_{15} \& P_{14} \& G_{13} \# P_{15} \& P_{14} \& P_{13} \& G_{12}) \# \\ &\quad (P_{15} \& P_{14} \& P_{13} \& P_{12}) \& \\ &\quad \quad (G_{11} \# P_{11} \& G_{10} \# P_{11} \& P_{10} \& G_9 \# P_{11} \& P_{10} \& P_9 \& G_8) \# \end{aligned}$$

$$\begin{aligned}
 & (P_{15} \& P_{14} \& P_{13} \& P_{12}) \& (P_{11} \& P_{10} \& P_9 \& P_8) \& \\
 & \quad (G_7 \# P_7 \& G_6 \# P_7 \& P_6 \& G_5 \# P_7 \& P_6 \& P_5 \& G_4) \# \\
 & (P_{15} \& P_{14} \& P_{13} \& P_{12}) \& (P_{11} \& P_{10} \& P_9 \& P_8) \& \\
 & \quad (P_7 \& P_6 \& P_5 \& P_4) \& (G_3 \# P_3 \& G_2 \# P_3 \& P_2 \& G_1 \# P_3 \& P_2 \& P_1 \& G_0) \# \\
 & (P_{15} \& P_{14} \& P_{13} \& P_{12}) \& (P_{11} \& P_{10} \& P_9 \& P_8) \& \\
 & \quad (P_7 \& P_6 \& P_5 \& P_4) \& (P_3 \& P_2 \& P_1 \& P_0) \& C_i
 \end{aligned} \tag{4-4}$$

令:

$$GX_4 = G_{15} \# P_{15} \& G_{14} \# P_{15} \& P_{14} \& G_{13} \# P_{15} \& P_{14} \& P_{13} \& G_{12} \tag{4-5}$$

$$GX_3 = G_{11} \# P_{11} \& G_{10} \# P_{11} \& P_{10} \& G_9 \# P_{11} \& P_{10} \& P_9 \& G_8 \tag{4-6}$$

$$GX_2 = G_7 \# P_7 \& G_6 \# P_7 \& P_6 \& G_5 \# P_7 \& P_6 \& P_5 \& G_4 \tag{4-7}$$

$$GX_1 = G_3 \# P_3 \& G_2 \# P_3 \& P_2 \& G_1 \# P_3 \& P_2 \& P_1 \& G_0 \tag{4-8}$$

$$PX_4 = P_{15} \& P_{14} \& P_{13} \& P_{12} \tag{4-9}$$

$$PX_3 = P_{11} \& P_{10} \& P_9 \& P_8 \tag{4-10}$$

$$PX_2 = P_7 \& P_6 \& P_5 \& P_4 \tag{4-11}$$

$$PX_1 = P_3 \& P_2 \& P_1 \& P_0 \tag{4-12}$$

可得:

$$C_4 = GX_0 \# PX_0 \& C_i \tag{4-13}$$

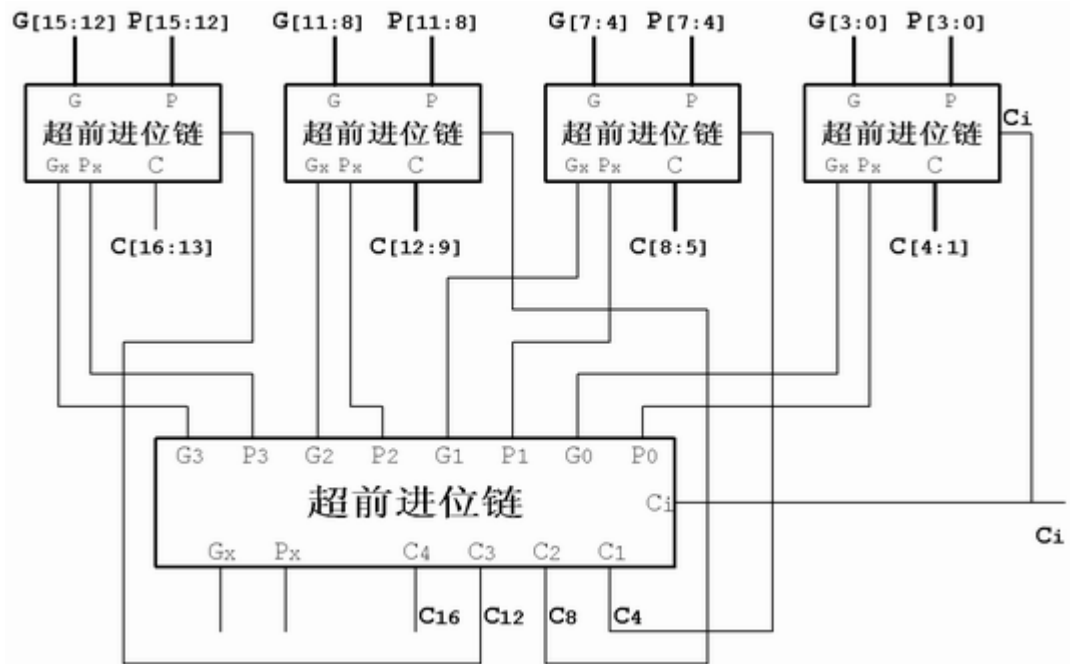
$$C_8 = GX_1 \# PX_1 \& GX_0 \# PX_1 \& PX_0 \& C_i \tag{4-14}$$

$$C_{12} = GX_2 \# PX_2 \& GX_1 \# PX_2 \& PX_1 \& GX_0 \# PX_2 \& PX_1 \& PX_0 \& C_i \tag{4-15}$$

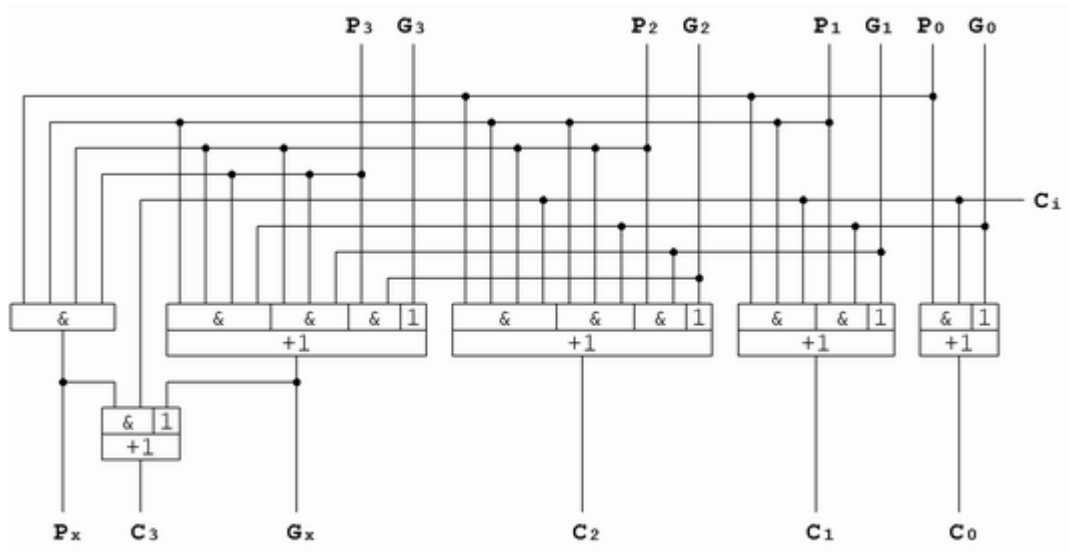
$$C_{16} = GX_3 \# PX_3 \& GX_2 \# PX_3 \& PX_2 \& GX_1 \# \\ PX_3 \& PX_2 \& PX_1 \& GX_0 \# PX_3 \& PX_2 \& PX_1 \& PX_0 \& C_i \tag{4-16}$$

比较式((4-13)~(4-16))与式((3-7)~(3-10)), 可看出: 单元间进位也可用超前进位链完成。

16 位超前进位链如下:



4 位超前进位链如下：



实际应用中，最高位进位一般不直接使用，对其要求可降低。本图中，此位需四级门延迟。

综合考察式(4-5)~(4-16)及上两幅图，可发现由 G、P 生成 GX、PX 有两级门延迟，由 GX、PX 生成单元间进位有两级门延迟，再需两级门延迟生成最终的进位。因此，总共需六级门延迟。

至此，问题已得到解决。

5、更多位超前进位链

对更多位超前进位链可参照上节，形成多级超前进位链树。

以 64 位超前进位链为例。

可用 4 位的超前进位链单元构成。第一级用 16 个单元，输出直接的进位；第二级用 4 个单元，用以形成第一级单元间进位；第三级用 1 个单元，为第二级提供单元间进位。这样，形成进位需 10 级门延迟：一、二级间需 2，二、三级间需 2，三、二级间需 2，二、一级间需 2，第一级输出需 2。

也可用 8 位超前进位链单元构成。第一级用 8 个单元，输出直接的进位；第二级用 1 个单元，为第一级提供单元间进位。这样，形成进位需 6 级门延迟。

实际上，每增加一级，就需要增加形成 GX、PX 的时间 2，再形成进位需时 2，总共增加 4。

这是否说明后一种方案就一定快！这不一定。需考虑电路中连线延迟、逻辑门的扇入扇出能力、逻辑门的延迟等多方面的因素。

再考虑 32 位、128 位超前进位链，应如何实现。

在实际应用中，拟制多种方案，考虑多方面因素，得到最优结果。

至此，作为高速加法器的核心——超前进位链——的讨论已完成。

6、算术运算设计

减法的数学表达式如下：

$$D = A - B - C_i \quad (6-1)$$

按代数运算法则以及补码的定义可得：

$$D = A - B - C_i \quad (6-2)$$

$$= A + (-B) - C_i \quad (6-3)$$

$$= A + (\sim B + 1) - C_i \quad (6-4)$$

$$= A + (\sim B) + 1 - C_i \quad (6-5)$$

$$= A + (\sim B) + (1 - C_i) \quad (6-6)$$

$$= A + (\sim B) + (\sim C_i) \quad (6-7)$$

由此得出的结论：

从被减数 A 减去减数 B 以及借位 C_i ，可以将 A 加上将 B 的按位取反的值以及将借位 C_i 取反的值！

加法的数学表达式：

$$D = A + B + C_i \quad (6-8)$$

比照式(6-7)、(6-8)，作出如下结论：

作加法时将 A、B、 C_i 相加，作减法时将 B、 C_i 取反与 A 相加！

引入一控制信号： as 。当其为 0 时作加法操作；当其为 1 时作减法操作。见下式：

$$D = A + (as ? B : (\sim B)) + (as @ C_i) \quad (6-9)$$

$$D = A + (asv @ B) + (as @ C_i) \quad (6-10)$$

在式(6-9)中，有一运算操作符： $? : ;$ ；其表达式为：

条件 ? 条件为真选项 : 条件为假选项

用之，因其为一位信号，不能与位串信号 B 直接操作。

在式(6-10)中，有一信号 asv ，它与信号 B 位宽相同，它由信号 as 填充。

超前进位链的输入值为 G、P。作加法运算时，G、P 为：

$$G = A \& B \quad (6-11)$$

$$P = A \# B \quad (6-12)$$

作减法运算时，G、P 为：

$$G = A \& (\sim B) \quad (6-13)$$

$$P = A \# (\sim B) \quad (6-14)$$

对于可控加减法运算

$$G = A \& (asv @ B) \quad (6-15)$$

$$P = A \# (asv @ B) \quad (6-16)$$

至此，算术运算已得到解决！

7、逻辑操作设计

逻辑操作是按位操作，不需要考虑位间进位。

逻辑操作也就：与、或、异或（同或）三种操作。

也很容易地想到：二操作数 A、B 直接进行逻辑操作得到结果 D。

还有一种方法：用 G、P 间接进行逻辑操作。

参照式(3-14)

$$D = A @ B @ C = G @ P @ C = (A \& B) @ (A \# B) @ C \quad (7-1)$$

逻辑操作表达式

$$D = A @ B = G @ P = (A \& B) @ (A \# B) \quad (7-2)$$

对于与操作：

$$D = A \& B = G \quad (7-3)$$

对于或操作：

$$D = A \# B = P \quad (7-4)$$

对于异或操作：

$$D = A @ B = G @ P \quad (7-5)$$

引入一二位控制信号：1c。当其为 00 时，作与操作；当其为 01 时，作或操作；当其为 1X 时，作异或操作。

$$G = (\sim (1c = 01)) ? (A \& B) : 0 \quad (7-6)$$

$$P = (\sim (1c = 00)) ? (A \# B) : 0 \quad (7-7)$$

$$D = G @ P \quad (7-8)$$

对这两种方法，我现在无从评测。我倾向于采用后一种。当然这样一来，多了一道环节，延迟不会没有。

8、标识位

任何一步算术逻辑操作都会产生各种各样的标识位。最主要的有：零标识、进位标识、符号标识、溢出标识。

若结果为零置零标识，或者说若待检测数据为零则置零标识。检测也不一定在此，可以在数据通路上检测。

符号标识的设置有两种：一是跟踪结果的符号位，即结果的最高位；二是使用扩展符号位，即：将二操作数作符号扩展一位，此时结果的最高位即用于设置符号标识。

在此讨论进位标识的设置，首先说明一下几个名词：进位、借位、进位标识、原进位。进位、借位是我们通常意义上的东西；进位标识跟踪进位、借位。原进位是指进位链上最高位。

加法操作时，可能会产生进位，因为结果可能会超出了表数范围，这在原进位上会表现出来。加法时，标识位的设置，只需要跟踪原进位即行。

我们知道，0 减去任何不为 0 的数，都会借位；0 减去 0，不会借位。遗憾的是，进位链上并不是如此表现。

0 减去一不为 0 的数 B，G 为 0。B 不为 0，则其位串中必定有 1，将 B 取反，此时位串中必定有 0，P 位串中也必定有 0。因此，原进位必定为 0。

0 减去 0，即为 0 加上 B 取反的值及 1，实际上，1 是作为进位引入的。此时 G 为 0。B 取反，此时位串全为 1，P 位串也全为 1，原进位跟踪最初输入的进位，其正为 1。因此，原进位必定为 1。

可以看到，原进位上的表现与实际的借位正好相反。

现在的讨论是：被减数为 0，非常特殊。

见下式：

$$D = A - B \quad (8-1)$$

$$= (0 + A) + (0 - B) \quad (8-2)$$

(0 + A) 原进位必会为 0。所以：

作减法时，进位标识的设置是将原进位取反。

溢出标识位的设置：作加法时，若二操作数符号相同，而结果符号相反，则置；作减法时，若减数、结果符号相同，而被减数符号相反，则置。

加法时

$$G = A \& B \quad (8-3)$$

$$P = A \# B \quad (8-4)$$

$$D = G @ P @ C \quad (8-5)$$

以 Amsb 表示数据 A 的最高位，以 Amsb+1 表示数据 A 的扩展最高位。

$$OV = Amsb \& Bmsb \& (\sim Dmsb) \# (\sim Amsb) \& (\sim Bmsb) \& Dmsb \quad (8-6)$$

$$= Gmsb \& (\sim (Gmsb @ Pmsb @ Cmsb)) \# (\sim Pmsb) \& (Gmsb @ Pmsb @ Cmsb) \quad (8-7)$$

$$= Gmsb \& (\sim Cmsb) \# (\sim Pmsb) \& Cmsb \quad (8-8)$$

参见式(3-6)

$$Cmsb+1 @ Cmsb = (Gmsb \# Pmsb \& Cmsb) @ Cmsb \quad (8-9)$$

$$= Gmsb \& (\sim Cmsb) \# (\sim Gmsb) \& (\sim Pmsb) \& Cmsb \quad (8-10)$$

$$= Gmsb \& (\sim Cmsb) \# (\sim ((Amsb \& Bmsb) \# (Amsb \# Bmsb))) \& Cmsb$$

$$= Gmsb \& (\sim Cmsb) \# (\sim (Amsb \# Bmsb)) \& Cmsb$$

$$= Gmsb \& (\sim Cmsb) \# (\sim Pmsb) \& Cmsb \quad (8-11)$$

可以看到：式(8-8)、(8-11)相等。可知：加法时：

$$OV = Cmsb+1 @ Cmsb \quad (8-12)$$

减法时

$$G = A \& (\sim B) \quad (8-13)$$

$$P = A \# (\sim B) \quad (8-14)$$

$$D = G @ P @ C \quad (8-15)$$

$$OV = Amsb \& (\sim Bmsb) \& (\sim Dmsb) \# (\sim Amsb) \& Bmsb \& Dmsb \quad (8-16)$$

$$= Gmsb \& (\sim (Gmsb @ Pmsb @ Cmsb)) \# (\sim Pmsb) \& (Gmsb @ Pmsb @ Cmsb) \quad (8-17)$$

$$= Gmsb \& (\sim Cmsb) \# (\sim Pmsb) \& Cmsb \quad (8-18)$$

参见式(3-6)

$$C_{msb+1} @ C_{msb} = (G_{msb} \# P_{msb} \& C_{msb}) @ C_{msb} \quad (8-19)$$

$$\begin{aligned} &= G_{msb} \& (\sim C_{msb}) \# (\sim G_{msb}) \& (\sim P_{msb}) \& C_{msb} \\ &= G_{msb} \& (\sim C_{msb}) \# (\sim (G_{msb} \# P_{msb})) \& C_{msb} \end{aligned} \quad (8-20)$$

$$\begin{aligned} &= G_{msb} \& (\sim C_{msb}) \# (\sim ((A_{msb} \& (\sim B_{msb})) \# (A_{msb} \# (\sim B_{msb})))) \& C_{msb} \\ &= G_{msb} \& (\sim C_{msb}) \# (\sim (A_{msb} \# (\sim B_{msb}))) \& C_{msb} \\ &= G_{msb} \& (\sim C_{msb}) \# (\sim P_{msb}) \& C_{msb} \end{aligned} \quad (8-21)$$

可以看到：式(8-18)、(8-21)相等。可知：减法时：

$$OV = C_{msb+1} @ C_{msb} \quad (8-22)$$

所以：

$$OV = C_{msb+1} @ C_{msb} \quad (8-23)$$

至此，算术逻辑部件设计，理论上的东西已全部解决！

9、设计示例 1 —— 16 位 7 功能算术逻辑部件

在上面的讨论中，我们注意到一个热点：G、P。

其实，超前进位链是标准的，[算术逻辑部件设计最主要的工作是设计 G、P 函数！](#)

这里用两节篇幅示例算术逻辑部件的设计。

本节示例：设计一 16 位 7 功能算术逻辑部件。

输入：二数据：A、B，进位； C_i ；

输出：结果：D，标识位：进位 C_F 、零标识 Z_F 、扩展符号 S_F 、溢出 O_F 。

功能控制： F_c ：000：A + B；

001：A - B；

010：A + B + C_i ；

011：A - B - C_i ；

100：A & B；

101：A # B；

110：A @ B；

111：A @ B；

要求尽量高速。

尽量高速，用超前进位链；数据宽度为 16 位，用 4 位超前进位链搭建。

```

module ALU_Demo_1 (
    A, B, Ci, //
    D, //
    Fc, //
    cf, zf, sf, of//
);

input [15:0] A, B;
input      Ci;
output[15:0] D;
input [ 2:0] Fc;
output      cf, zf, sf, of;

reg  [15:0] G, P;
wire [16:0] C;

//  Fc      G      P      Cp      Descript
//  000      A & B  A | B  0      A + B
//  001      A & (~ B)  A | (~ B)  1      A - B
//  010      A & B  A | B  Ci     A + B + Ci
//  011      A & (~ B)  A | (~ B)  (~ Ci)  A - B - Ci
//  100      A & B  0      X      A & B
//  101      0      A | B  X      A | B
//  110      A & B  A | B  X      A ^ B
//  111      A & B  A | B  X      A ^ B

//----generat the G
always @(Fc or A or B)
    casex(Fc)
        3'b000:  G = A & B;
        3'b001:  G = A & (~ B);
        3'b010:  G = A & B;
        3'b011:  G = A & (~ B);
        3'b100:  G = A & B;
        3'b101:  G = {16{1'b0}};
        3'b110:  G = A & B;
        3'b111:  G = A & B;
        default: G = A & B;
    endcase

```

```

//---generat the P
always @(Fc or A or B)
  casex(Fc)
    3'b000: P = A | B;
    3'b001: P = A | (~ B);
    3'b010: P = A | B;
    3'b011: P = A | (~ B);
    3'b100: P = {16{1'b0}};
    3'b101: P = A | B;
    3'b110: P = A | B;
    3'b111: P = A | B;
    default: P = A | B;
  endcase

//---generat the C in leading carry link
wire Cp;
wire [ 3:0] G1,P1;
wire [ 4:1] Cx;
wire [16:1] Ct;

assign Cp = ((Ci & Fc[1]) ^ Fc[0]);

Carry_Leading_4 CL0 (G[ 3: 0],P[ 3: 0],Cp ,Ct[ 4: 1],G1[0],P1[0]);
Carry_Leading_4 CL1 (G[ 7: 4],P[ 7: 4],Cx[1],Ct[ 8: 5],G1[1],P1[1]);
Carry_Leading_4 CL2 (G[11: 8],P[11: 8],Cx[2],Ct[12: 9],G1[2],P1[2]);
Carry_Leading_4 CL3 (G[15:12],P[15:12],Cx[3],Ct[16:13],G1[3],P1[3]);

Carry_Leading_4 CLX (G1[ 3:0],P1[ 3:0],Cp,Cx[ 4:1],,);

//assign C = {Ct,Cp};
assign C = {Cx[4],Ct[15:13],Cx[3],Ct[11:9],Cx[2],Ct[7:5],Cx[1],Ct[3:1],Cp};

//---generate the result
assign D = G ^ P ^ ((~ Fc[2]) ? C[15:0] : {16{1'b0}});

assign cf = (C[16] ^ Fc[0]) & (~ Fc[2]);
assign zf = ! D;
assign sf = G[15] ^ P[15] ^ ((~ Fc[2]) & C[16]);
assign of = (C[15] ^ C[16]) & (~ Fc[2]);

endmodule

```

```

module Carry_Leading_4 (G,P,Ci,C,G1,P1);

input  [ 3:0] G,P;
output [ 4:1] C;
input      Ci;
output     G1,P1;

wire  [ 3:0] G,P;
wire  [ 4:1] C;
wire      Ci;
wire      G1,P1;

assign C[1] = G[0] | P[0] & Ci;
assign C[2] = G[1] | P[1] & G[0] | P[1] & P[0] & Ci;
assign C[3] = G[2] | P[2] & G[1] | P[2] & P[1] & G[0] | P[2] & P[1] & P[0] & Ci;
assign C[4] = G1 | P1 & Ci;

assign G1 = G[3] | P[3] & G[2] | P[3] & P[2] & G[1] | P[3] & P[2] & P[1] & G[0];
assign P1 = P[3] & P[2] & P[1] & P[0];

endmodule

```

10、设计示例 2 —— 4 位 16 功能算术逻辑部件

本节示例：设计一 4 位 16 功能算术逻辑部件。

输入：二数据：A、B，进位：Ci；

输出：结果：D，标识位：进位CF、零标识ZF、扩展符号SF、溢出OF。

功能控制：Fc：0000：B；

0001：B + 1；

0010：A + B + Ci；

0011：A + B；

0100： \sim B；

0101： $-$ B；

0110：A - B + Ci - 1；

0111：A - B；

```

1000: B - 1;
1001: B - A;
1010: B - A + Ci - 1;
1011: ~ A;
1100: A & B;
1101: A # B;
1110: A @ B;
1111: abs A;

```

要求尽量高速。

出现在功能表中有一如下操作:

$$\begin{aligned}
 D &= A - B + C_i - 1 \\
 &= A + (-B) + C_i - 1 \\
 &= A + (\sim B) + 1 + C_i - 1 \\
 &= A + (\sim B) + C_i
 \end{aligned}$$

```

module ALU_Demo_2 (
    A, B, Ci, //
    D, //
    Fc, //
    cf, zf, sf, of //
);

input [ 3:0] A, B;
input      Ci;
output [ 3:0] D;
input [ 3:0] Fc;
output      cf, zf, sf, of;

reg [ 3:0] G, P;
reg [ 4:0] C;
wire      ol; // logic operate if it is '1'
wire      so; // subtract operate when it is '1'

// D = A - B + Ci - 1
//   = A + (- B) + Ci - 1
//   = A + (~ B) + 1 + Ci - 1
//   = A + (~ B) + Ci

```

```

//Function table & it descript in other word
//   Fc      Descript      ow      T
//   0000    B              0 + B + 0    0 + B + 0
//   0001    B + 1          0 + B + 1    0 + B + 1
//   0010    A + B + Ci     A + B + Ci   A + B + Ci
//   0011    A + B          A + B        A + B
//   0100    ~ B            0 - B - 1    0 + (~ B)
//   0101    - B            0 - B        0 + (~ B) + 1
//   0110    A - B + Ci - 1  A - B + Ci - 1  A + (~ B) + Ci
//   0111    A - B          A - B        A + (~ B) + 1
//   1000    B - 1          B - 0 - 1    (~ 0) + B
//   1001    B - A          B - A        (~ A) + B + 1
//   1010    B - A + Ci - 1  B - A + Ci - 1  (~ A) + B + Ci
//   1011    ~ A            0 - A - 1    (~ A) + 0
//   1100    A & B          A & B        A & B
//   1101    A | B          A | B        A | B
//   1110    A ^ B          A ^ B        A ^ B
//   1111    abs A          (0 - A) / (0 + A)  (A[3] ? (~ A) : A) + 0 + (A[3])

//-----generat the G
// G = oa & ob
always @(Fc or A or B)
  casex(Fc)
    4'b0000: G = 4'b0000; //oa = 0; ob = B; cp = 0;
    4'b0001: G = 4'b0000; //oa = 0; ob = B; cp = 1;
    4'b0010: G = A & B; //oa = A; ob = B; cp = Ci;
    4'b0011: G = A & B; //oa = A; ob = B; cp = 0;
    4'b0100: G = 4'b0000; //oa = 0; ob = (~ B); cp = 0;
    4'b0101: G = 4'b0000; //oa = 0; ob = (~ B); cp = 1;
    4'b0110: G = A & (~ B); //oa = A; ob = (~ B); cp = Ci
    4'b0111: G = A & (~ B); //oa = A; ob = (~ B); cp = 1;
    4'b1000: G = B; //oa = (~ 0); ob = B; cp = 0;
    4'b1001: G = (~ A) & B; //oa = (~ A); ob = B; cp = 1;
    4'b1010: G = (~ A) & B; //oa = (~ A); ob = B; cp = Ci;
    4'b1011: G = 0; //oa = (~ A); ob = 0; cp = 0;
    4'b1100: G = A & B; //oa = A; ob = B; cp = x;
    4'b1101: G = 4'b0000; //oa = A; ob = B; cp = x;
    4'b1110: G = A & B; //oa = A; ob = B; cp = x;
    4'b1111: G = 4'b0000; //oa = (A[3] ? (~ A) : A); ob = 0; cp = A[3];
    default: G = 4'bxxxx; //oa = X; ob = X; cp = x;
  endcase

```

```

//---generat the P
// P = oa | ob
always @(Fc or A or B)
  casex(Fc)
    4'b0000: P = B; //oa = 0; ob = B; cp = 0;
    4'b0001: P = B; //oa = 0; ob = B; cp = 1;
    4'b0010: P = A | B; //oa = A; ob = B; cp = Ci;
    4'b0011: P = A | B; //oa = A; ob = B; cp = 0;
    4'b0100: P = (~ B); //oa = 0; ob = (~ B); cp = 0;
    4'b0101: P = (~ B); //oa = 0; ob = (~ B); cp = 1;
    4'b0110: P = A | (~ B); //oa = A; ob = (~ B); cp = Ci;
    4'b0111: P = A | (~ B); //oa = A; ob = (~ B); cp = 1;
    4'b1000: P = 4'b1111; //oa = (~ 0); ob = B; cp = 0;
    4'b1001: P = (~ A) | B; //oa = (~ A); ob = B; cp = 1;
    4'b1010: P = (~ A) | B; //oa = (~ A); ob = B; cp = Ci;
    4'b1011: P = (~ A); //oa = (~ A); ob = 0; cp = 0;
    4'b1100: P = 4'b0000; //oa = A; ob = B; cp = x;
    4'b1101: P = A | B; //oa = A; ob = B; cp = x;
    4'b1110: P = A | B; //oa = A; ob = B; cp = x;
    4'b1111: P = (A[3] ? (~ A) : A); //oa = (A[3] ? (~ A) : A); ob = 0; cp = A[3];
  default: P = 4'bxxxx; //oa = X; ob = X; cp = x;
endcase

```

```

//---generat the Ci
// P = oa | ob
always @(Fc or A or B)
  casex(Fc)
    4'b0000: C[0] = 1'b0; //oa = 0; ob = B; cp = 0;
    4'b0001: C[0] = 1'b1; //oa = 0; ob = B; cp = 1;
    4'b0010: C[0] = Ci; //oa = A; ob = B; cp = Ci;
    4'b0011: C[0] = 1'b0; //oa = A; ob = B; cp = 0;
    4'b0100: C[0] = 1'b0; //oa = 0; ob = (~ B); cp = 0;
    4'b0101: C[0] = 1'b1; //oa = 0; ob = (~ B); cp = 1;
    4'b0110: C[0] = Ci; //oa = A; ob = (~ B); cp = Ci;
    4'b0111: C[0] = 1'b1; //oa = A; ob = (~ B); cp = 1;
    4'b1000: C[0] = 1'b0; //oa = (~ 0); ob = B; cp = 0;
    4'b1001: C[0] = 1'b1; //oa = (~ A); ob = B; cp = 1;
    4'b1010: C[0] = Ci; //oa = (~ A); ob = B; cp = Ci;
    4'b1011: C[0] = 1'b0; //oa = (~ A); ob = 0; cp = 0;
    4'b1100: C[0] = 1'bx; //oa = A; ob = B; cp = x;
    4'b1101: C[0] = 1'bx; //oa = A; ob = B; cp = x;
    4'b1110: C[0] = 1'bx; //oa = A; ob = B; cp = x;
    4'b1111: C[0] = A[3]; //oa = (A[3] ? (~ A) : A); ob = 0; cp = A[3];
  default: C[0] = 1'bx; //oa = X; ob = X; cp = x;
endcase

```

```

assign ol = (Fc == 4'b1100) | (Fc == 4'b1101) | (Fc == 4'b1110);

assign so = (Fc == 4'b0101) | (Fc == 4'b0110) | (Fc == 4'b0111) | (Fc == 4'b1000) |
           (Fc == 4'b1001) | (Fc == 4'b1010) | ((Fc == 4'b1111) & A[3]);

//---generat the C in leading carry link
Carry_Leading_4 CL4 (G,P,C[0],C[ 4:1],,);

//---generate the result
assign D = G ^ P ^ ((~ ol) ? C[ 3:0] : 4'b0000);

assign cf = (C[4] ^ so) & (~ ol);
assign zf = ! D;
assign sf = G[3] ^ P[3] ^ C[4];
assign of = (C[4] ^ C[3]) & (~ ol);

endmodule

module Carry_Leading_4 (G,P,Ci,C,Gl,P1);

input  [ 3:0] G,P;
output [ 4:1] C;
input          Ci;
output        Gl,P1;

wire  [ 3:0] G,P;
wire  [ 4:1] C;
wire          Ci;
wire          Gl,P1;

assign C[1] = G[0] | P[0] & Ci;
assign C[2] = G[1] | P[1] & G[0] | P[1] & P[0] & Ci;
assign C[3] = G[2] | P[2] & G[1] | P[2] & P[1] & G[0] | P[2] & P[1] & P[0] & Ci;
assign C[4] = Gl | P1 & Ci;

assign Gl = G[3] | P[3] & G[2] | P[3] & P[2] & G[1] | P[3] & P[2] & P[1] & G[0];
assign P1 = P[3] & P[2] & P[1] & P[0];

endmodule

```

后 记

本人从事集成电路设计以来，所见过有关算术逻辑部件的设计，无一例外属于系统级描述。在和有关从业人员的交谈中，甚至于超前进位不知道的也大有人在。

我于早期作过《超前进位链》一文，希望（目睹者）能于算术逻辑部件设计有所帮助。但后来发现：效果为零。一气之下，作此资料。

应该说，以寄存器传输级描述算术逻辑部件设计应是此行业从业人员基本能力要求。

如果将设计能力分为三个层次：艺术级、匠级、学徒级。系统级描述只需学徒级水平就行。

在微处理器设计中，在走到 FPGA 功能验证这一步时，算术逻辑部件设计需要匠级水平，译码控制需艺术级水平——我所见到的算术逻辑部件设计是系统级描述。

作此文，希望能于各位工作带来帮助。

此资料“超前进位链”部分直接取材我以前（为公司）所作《超前进位链》（只有极微改动），因此涉及版权问题。但超前进位链技术不涉及版权问题，各位不用担心，本资料所提供技术可放心使用、流转。

本人学历不高、学问不深、实践不足、更兼一气之作，难免谬误，望各位不吝指正。

蒋 小 龙

2001.7.5

个 人 介 绍

蒋小龙乃身份证名，亦用名：蒋维隆

集成电路逻辑设计。设计过 16 位、32 位微处理器，现考虑超标量微处理器实现

联系：jiangweilong@cmmail.com

jiangweilong@china.com