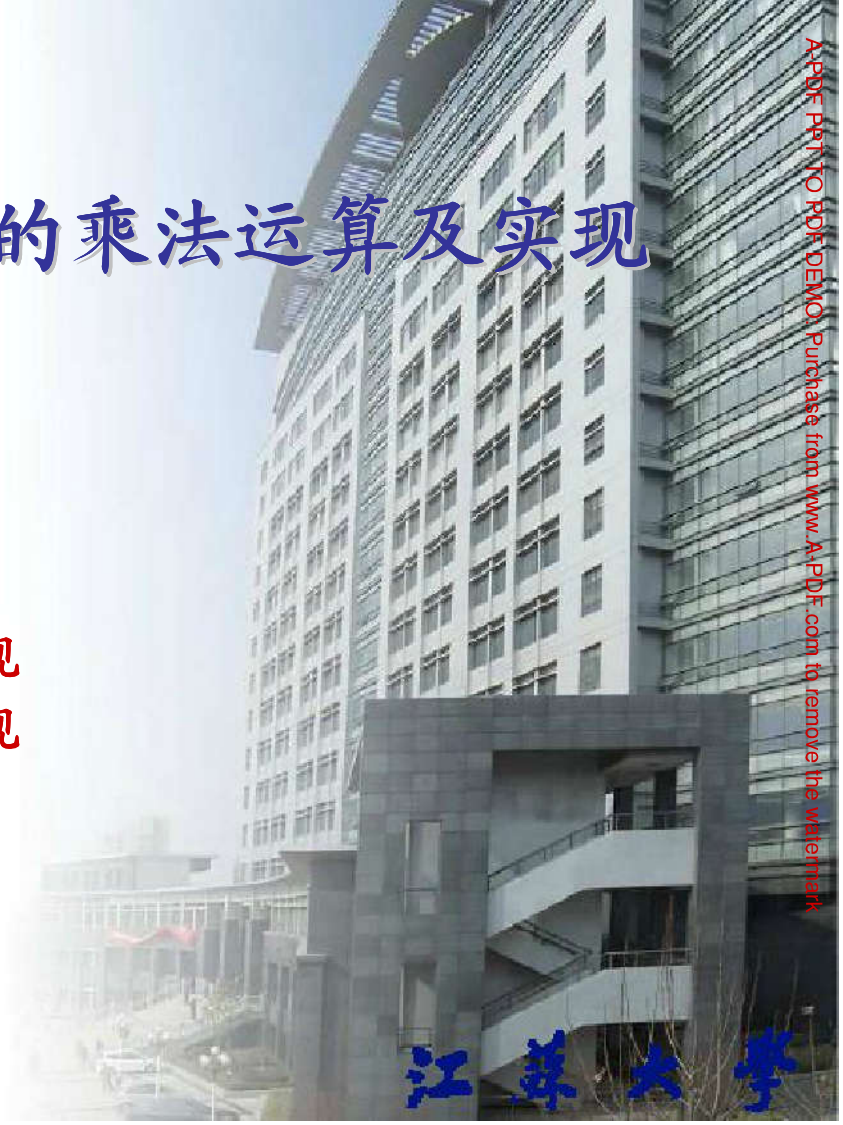


4.3 定点小数的乘法运算及实现

本节内容

- 原码乘法及实现
- 补码乘法及实现
- 阵列乘法器

❖在计算机中实现乘除法运算，根据对运算速度和价格的要求通常有三种方式：

1）若对运算速度要求不高可用软件实现；

2）在加减运算电路的基础上增加左、右移位及其他一些逻辑线路以实现乘除运算，并在机器中设有乘除法指令；

3）在要求快速乘除运算的机器中，设置专用的高速阵列乘除运算器，指令系统中有相应的乘除法指令

原码乘法及实现

❖原码乘法的运算规则：

1）积的符号位由两乘数的符号位异或得到

2）积的原码数值部分由两乘数原码数值部分相乘得到

3）将积的符号与积的数值拼接就得到积的原码

$$[X]_{原} \times [Y]_{原} = (Xs \oplus Ys)(|X| \times |Y|)$$

乘法分解为加法和移位操作

❖例：求$(1101)_2 \times (1011)_2$的手算式

1101 —— X

1011 —— $Y=y_3y_2y_1y_0$

1101 —— $X\cdot y_0\cdot 2^0$

1101 —— $X\cdot y_1\cdot 2^1$

0000 —— $X\cdot y_2\cdot 2^2$

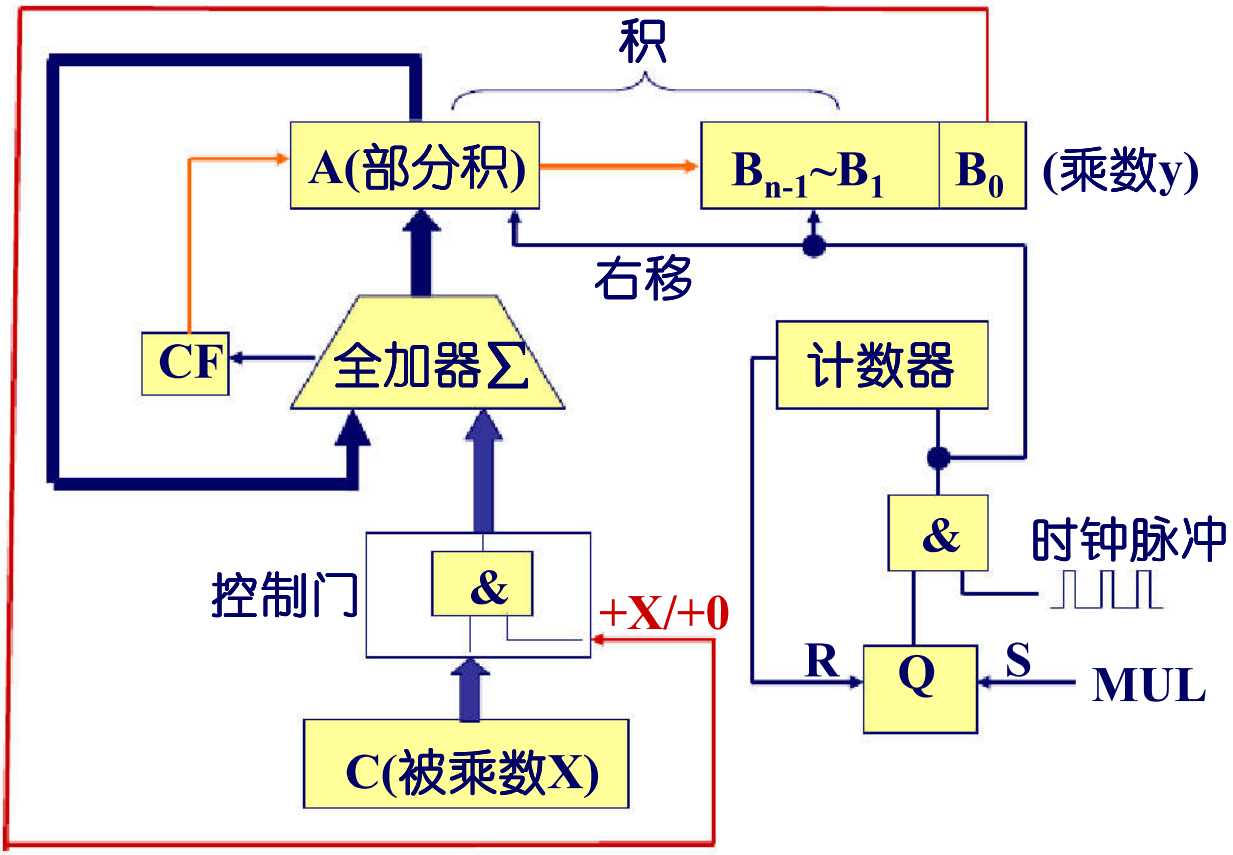
1101 —— $X\cdot y_3\cdot 2^3$

10001111 —— $0+X\cdot y_0\cdot 2^0+X\cdot y_1\cdot 2^1+X\cdot y_2\cdot 2^2+X\cdot y_3\cdot 2^3$

机算与手算的区别：

- 多项相加改为两项逐次累加，只需要普通加法器；
- 左移改为右移，只需要n位加法器；
- 乘数Y与积的低n位共用一个寄存器，右移后只需判断y_0决定加X或加0

硬件实现原码乘法



例：若$[X]_{原}$=0,1101，$[Y]_{原}$=1,1011，求$[Z]_{原}$=?

解：符号：$Z_s = X_s \oplus Y_s = 0 \oplus 1 = 1$

CF	部分积	乘数 Y_0	操作说明	计数器
0	0 0 0 0	1 0 1 1	Y_0=1,+|X|	0
	+ 1 1 0 1			
0	1 1 0 1			
0	0 1 1 0	1 1 0 1	→1	1
	+ 1 1 0 1		Y_0=1,+|X|	
1	0 0 1 1			
1	1 0 0 1	1 1 1 0	→1	2
	+ 0 0 0 0		Y_0=0,+0	
0	1 0 0 1			
0	0 1 0 0	1 1 1 1	→1	3
	+ 1 1 0 1		Y_0=1,+|X|	
1	0 0 0 1			
0	1 0 0 0	1 1 1 1	→1	4

积高部　　积低部

$\therefore [Z]_{原}$=1,10001111

原码两位乘

❖原码一位乘依据Y_i决定操作，原码两位乘每次根据连续两位乘数$Y_{i+1}Y_i$ $(=2Y_{i+1}+Y_i)$决定本次操作

❖n位数的乘法一般只需n/2次操作

Yi+1 Yi	$2Y_{i+1}+Y_i$	操作
0 0	0	+ 0，右移2位
0 1	1	+ |X|，右移2位
1 0	$\frac{1}{2}\left(\frac{1}{2}P_i+X\right)=\frac{1}{4}\left(P_i+2X\right)$	
1 1	$\frac{1}{2}\left(\frac{1}{2}\left(P_i+X\right)+X\right)=\frac{1}{4}\left(P_i+3X\right)$	-|X|，右移2位，下一步多加|X|

原码两位乘运算规则

+3|X| = -|X| + 4|X|，即：减|X|, 右移2位, 下一步再加|X|

- 欠账标志：Cj

$Y_{i+1}\ Y_i$	C_j	$Y_{i+1}\ Y_i$ +) C_j	操作
00	0	00	+0　2→　0→Cj
00	1	01	+|X|　2→　0→Cj
01	0	01	+|X|　2→　0→Cj
01	1	10	+2|X|　2→　0→Cj
10	0	10	+2|X|　2→　0→Cj
10	1	11	-|X|　2→　1→Cj
11	0	11	-|X|　2→　1→Cj
11	1	00	+0　2→　1→Cj

原码两位乘法 例

例: 若$[X]_{原}$=0,0111111, $[Y]_{原}$=1,0111001, 求$[Z]_{原}$=?

解: 符号: Zs = Xs ⊕ Ys = 0 ⊕ 1 = 1

|X| = 000,0111111

2|X| = 000,1111110

$[-|X|]_{补}$= 111,1000001

- –|X|转换为 +$[-|X|]_{补}$，所以采用补码加法
- 因为2X可能溢出，再+|X|又可能再次溢出，故采用3位符号位

|Y| = 00111001

- 数值位增加一位

❖注意：运算数是X、Y的绝对值。

❖结果 = 1,00111000000111

例: $[X]_{原}$=0,0111111, $[Y]_{原}$=1,0111001, 求$[Z]_{原}$

符号	部分积	乘数Y	Cj	操作
0 0 0	0 0 0 0 0 0 0	0 0 1 1 1 0 0 1	0	+|X|
+0 0 0	0 1 1 1 1 1 1			
0 0 0	0 1 1 1 1 1 1			0→Cj, 2→
0 0 0	0 0 0 1 1 1 1	1 1 0 0 1 1 1 0	0	+2|X|
+0 0 0	1 1 1 1 1 1 0			
0 0 1	0 0 0 1 1 0 1			0→Cj, 2→
0 0 0	0 1 0 0 0 1 1	0 1 1 1 0 0 1 1	0	-|X|
+1 1 1	1 0 0 0 0 0 1			
1 1 1	1 1 0 0 1 0 0			1→Cj, 2→
1 1 1	1 1 1 1 0 0 1	0 0 0 1 1 1 0 0	1	+|X|
+0 0 0	0 1 1 1 1 1 1			
0 0 0	0 1 1 1 0 0 0			0→Cj, 1→
0 0 0	0 0 1 1 1 0 0	0 0 0 0 1 1 1 0	0	

积高位　　积低位　　最后一步只移一位

$[Z]_{原}$ = 1,0011100000111

4.3.2 补码乘法运算

Booth法。目标：求$[X \cdot Y]_{补}$

设被乘数X、乘数Y均为n+1位的定点小数，

$X_{补} = x_0.x_1x_2\cdots x_n$, $Y_{补} = y_0.y_1y_2\cdots y_n$。

则$[Y]_{补}=y_0\cdot 2^0+y_1\cdot 2^{-1}+\cdots\cdots+y_n\cdot 2^{-n}$

- $y_0 = 0$（Y>0）时，

  $Y = Y_{补} = y_0\cdot 2^0+y_1\cdot 2^{-1}+\cdots\cdots+y_n\cdot 2^{-n}$

- 当$y_0 = 1$（Y<0）时，$Y_{补} = 2 + Y$可得

  $Y = Y_{补} - 2 = y_0+y_1\cdot 2^{-1}+\cdots\cdots+y_n\cdot 2^{-n} - 2y_0$

  $= -y_0\cdot 2^0+y_1\cdot 2^{-1}+\cdots\cdots+y_n\cdot 2^{-n}$

两式可合写为：

$Y= -y_0\cdot 2^0+y_1\cdot 2^{-1}+\cdots\cdots+y_n\cdot 2^{-n}$

$$Y = -y_0 \cdot 2^0 + y_1 \cdot 2^{-1} + \cdots + y_{n-1} \cdot 2^{-(n-1)} + y_n \cdot 2^{-n}$$

$$= -y_0 \cdot 2^0 + y_1 \cdot 2^0 - y_1 \cdot 2^{-1} - y_{n-1} \cdot 2^{-(n-1)} + y_n \cdot 2^{-(n-1)} - y_n \cdot 2^{-n} + 0 \cdot 2^{-n}$$

$$= (y_1 - y_0) \cdot 2^0 + (y_2 - y_1) \cdot 2^{-1} + \cdots\cdots + (y_n - y_{n-1}) \cdot 2^{-(n-1)} + (0 - y_n) \cdot 2^{-n}$$

$$[X \cdot Y]_{补} = [(y_1 - y_0) \cdot 2^0 \cdot X]_{补} + [(y_2 - y_1) \cdot 2^{-1} \cdot X]_{补} + \cdots\cdots + [(y_n - y_{n-1}) \cdot 2^{-(n-1)} \cdot X]_{补} + [(0 - y_n) \cdot 2^{-n} \cdot X]_{补}$$

上式表明：求$[XY]_{补}$也可以转化为加法和移位

❖$[(y_{i+1} - y_i)\cdot 2^{-i}\cdot X]_{补} = [(y_{i+1} - y_i)\cdot X]_{补}\cdot 2^{-i}$

$y_i\ y_{i+1}$	$y_{i+1}-y_i$	操　作
0　0	0	部分积+0，1
0　1	1	部分积+$[X]_{补}$，1
1　0	−1	部分积+$[-X]_{补}$，1
1　1	0	部分积+0，1

Booth算法

❖ Booth算法描述如下：

1）参加乘法运算的两乘数用补码表示，乘积也是补码

2）符号位参与运算，无须单独处理

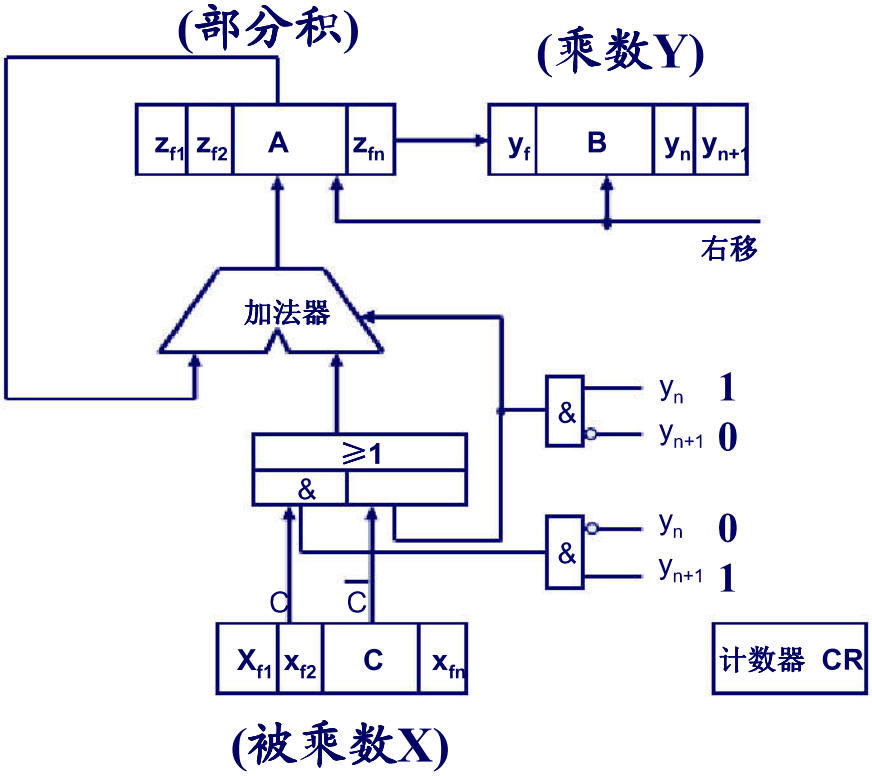
3）乘数最低位后附加一位y_{n+1}，初值为0

4）判断$y_n y_{n+1}$，决定$+X_{补}$或$+[-X]_{补}$或+0

5）部分积算术右移一位

6）重复4、5步骤，共需做n+1次累加，n次移位，第n+1次不移位。

Booth乘法原理框图



例：已知$[X]_{补}$=0.1010，$[Y]_{补}$=1.0011，用Booth法求$[X\cdot Y]_{补}$

解：$[-X]_{补}$=1.0110

是否需要双符号位？

	部分积	计数器
	0 0 0 0 0	0
+	1 0 1 1 0	
	1 0 1 1 0	
	1 1 0 1 1	1
	1 1 1 0 1	2
+	0 1 0 1 0	
	0 0 1 1 1	
	0 0 0 1 1	3
	0 0 0 0 1	4
+	1 0 1 1 0	
	1 0 1 1 1	5

$[X\cdot Y]_{补}$=1.01111110

补码两位乘

❖将比较$Y_{i-1}Y_i$的状态与比较Y_iY_{i+1}的状态所执行的运算合成一步

对照：补码一位乘法

❖例：已知$[X]_{补}$=0,0110011，$[Y]_{补}$=1,1001110，用补码两位乘求$[X\cdot Y]_{补}$

❖解：

$[-X]_{补}$ = 11,1001101

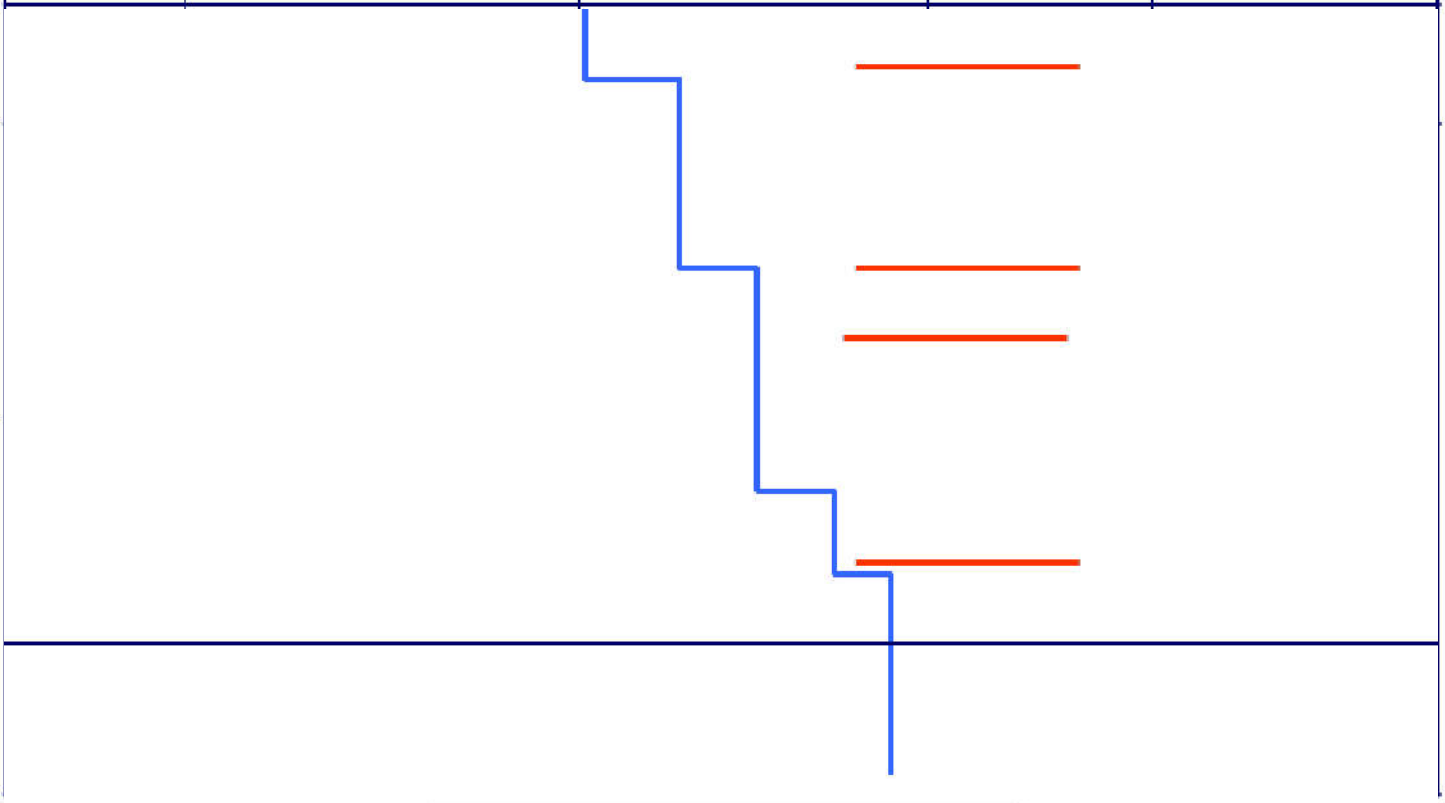
$2[-X]_{补}$= 11,0011010

$2[X]_{补}$ = 00,0110011

- 因为2X可能溢出，故采用双符号位

例:$[X]_{补}=0,0110011$, $[Y]_{补}=1,1001110$, 求$[X\cdot Y]_{补}$

	部分积	乘数	附加位	操作



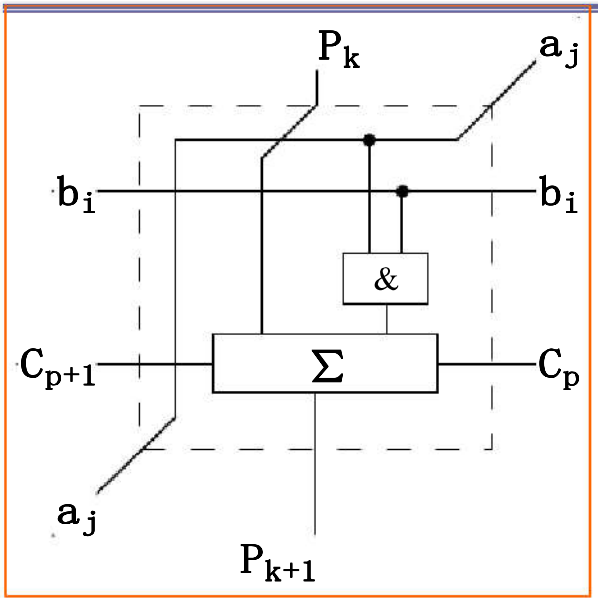
$[Z]_{补}=1,11011000001010$

4.3.3 阵列乘法器

随着大规模集成电路的迅速发展以及硬件价格的降低，出现了多种阵列乘法组件，目的就是利用硬件的叠加方法或流水处理的方法来提高乘法运算速度。

设A=a3a2a1a0，B = b3b2b1b0，A × B手算式:

阵列乘法器



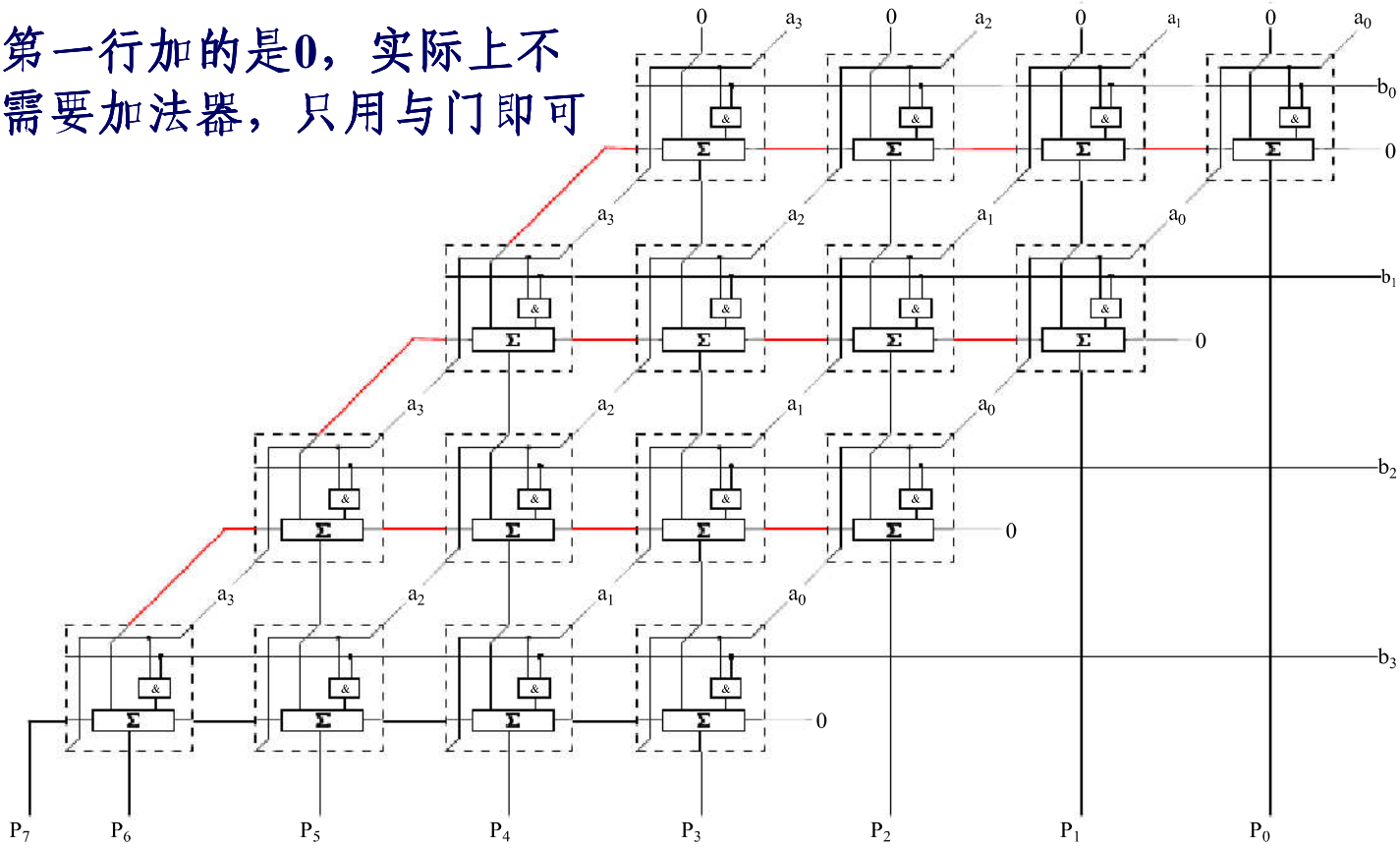
基本思想：

- 位积a_jb_i可用与门产生
- 设计加法网络，将多项位积一次求和

			a3	a2	a1	a0	
×)			b3	b2	b1	b0	
			a3b0	a2b0	a1b0	a0b0	
		a3b1	a2b1	a1b1	a0b1		
	a3b2	a2b2	a1b2	a0b2			
+)	a3b3	a2b3	a1b3	a0b3			
P6	P5	P4	P3	P2	P1	P0	

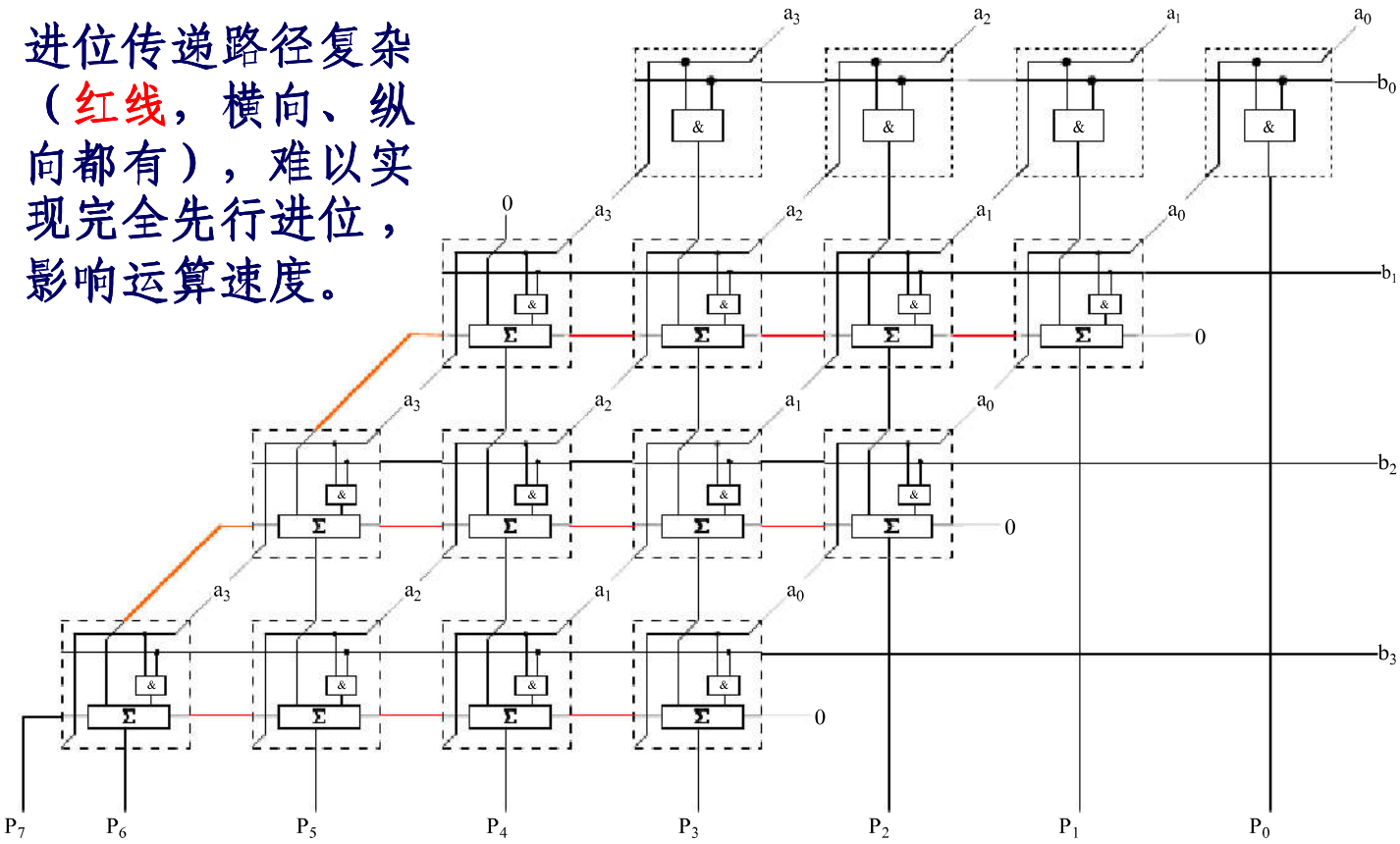
进位传递CPA (Carry Propagate Adder)的加法网络

第一行加的是0，实际上不需要加法器，只用与门即可



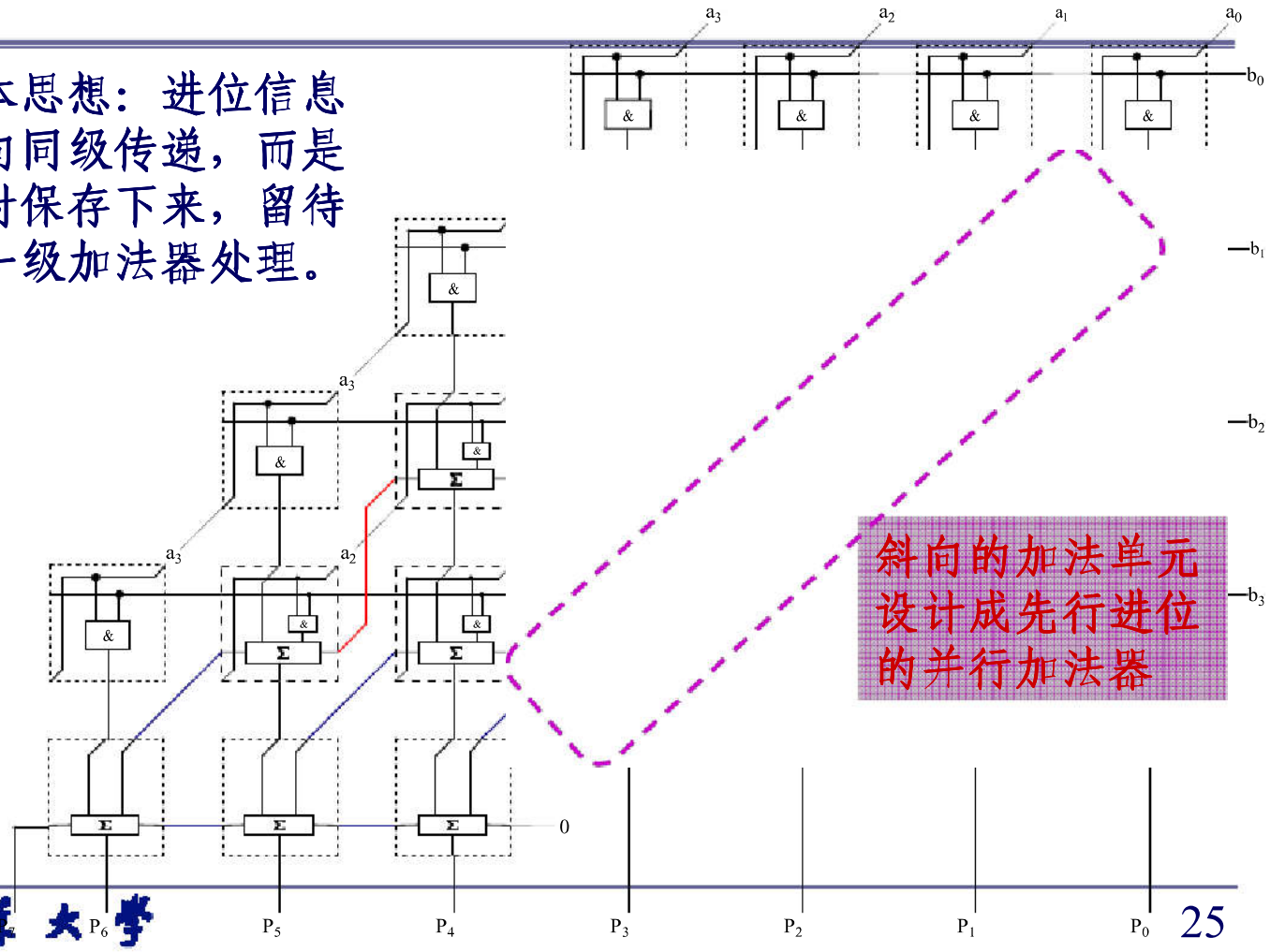
n × n位的乘法，需要n^2个与门，n-1个n位的并行加法器

进位传递路径复杂（红线，横向、纵向都有），难以实现完全先行进位，影响运算速度。



采用CSA (Carry Save Adder) 进位存储的加法网络

基本思想：进位信息不向同级传递，而是暂时保存下来，留待下一级加法器处理。



斜向的加法单元设计成先行进位的并行加法器

作业

- 4.6 (2)
- 4.9