2.3 定点乘法运算

提纲

2.3.1 原码乘法

- 一、人工算法与机器算法的同异性

- 设 x = 0.1101，y = 0.1011．下面让我们先用习惯方法求其乘积，其过程如下:

```
              0.  1  1  0  1   (x)
×             0.  1  0  1  1   (y)
-------------------------------
                  1  1  0  1
               1  1  0  1
            0  0  0  0
         1  1  0  1                  -
+-------------------------------
0.  1  0  0  0  1  1  1  1   (z)
```

2.3.1 原码乘法

- 一、人工算法与机器算法的同异性

- 上述的运算过程与十进制乘法相似：从乘数 y 的最低位开始,若这一位为“1”，则将被乘数 x 写下；若这一位为“0”，则写下全0。然后在对乘数 y 的次高位进行乘法运算，其规则同上，不过这一位乘数的权与最低位乘数的权不一样，因此被乘数 x 要左移一位。以此类推直到乘数个位乘完为止，最后将它们统统加起来便得到最后乘积 z 。同理，如果被乘数和乘数用定点整数表示，我们也会得到同样的结果。
- 数值部分的运算方法与普通的十进制乘法类似，不过对于用二进制表达式的数来说，其乘法规则更为简单一些。

2.3.1 原码乘法

- 一、人工算法与机器算法的同异性
- 在定点计算机中，两个原码表示的数相乘的运算规则是：乘积的符号位由两数的符号位按异或运算得到，而乘积的数值部分则是两个正数相乘之积。
- 设n位被乘数和乘数用定点小数表示

  被乘数 $[x]_{原} = x_f . x_{n-1} \cdots x_1 x_0$

  乘数 $[y]_{原} = y_f . y_{n-1} \cdots y_1 y_0$

- 则两数的乘积

  $[z]_{原} = (x_f \oplus y_f)\ (0.x_{n-1} \cdots x_1 x_0)(0.y_{n-1} \cdots y_1 y_0)$

  式中，x_f为被乘数符号，y_f为乘数符号。

2.3.1 原码乘法

- 人们习惯的算法对机器并不完全适用。原因之一，机器通常只有n位长，两个n位数相乘，乘积可能为2n位。原因之二，只有两个操作数相加的加法器难以胜任将n个位积一次相加起来的运算
- 早期计算机中为了简化硬件结构，采用串行的1位乘法方案，即多次执行“加法—移位”操作来实现。这种方法并不需要很多器件
- 然而串行方法毕竟太慢，自从大规模集成电路问世以来，出现了各种形式的流水式阵列乘法器，它们属于并行乘法器。

2.3.2 不带符号的阵列乘法器

- 设有两个不带符号的二进制整数：

$$A = a_{m-1}\cdots a_1 a_0$$
$$B = b_{n-1}\cdots b_1 b_0$$

- 它们的数值分别为a和b,即

$$a = \sum_{i=0}^{m-1} a_i 2^i \qquad b = \sum_{j=0}^{n-1} b_j 2^j$$

- 在二进制乘法中，被乘数A与乘数B相乘，产生m + n位乘积P：

$$P = p_{m+n-1}\cdots p_1 p_0$$

- 乘积P的数值为

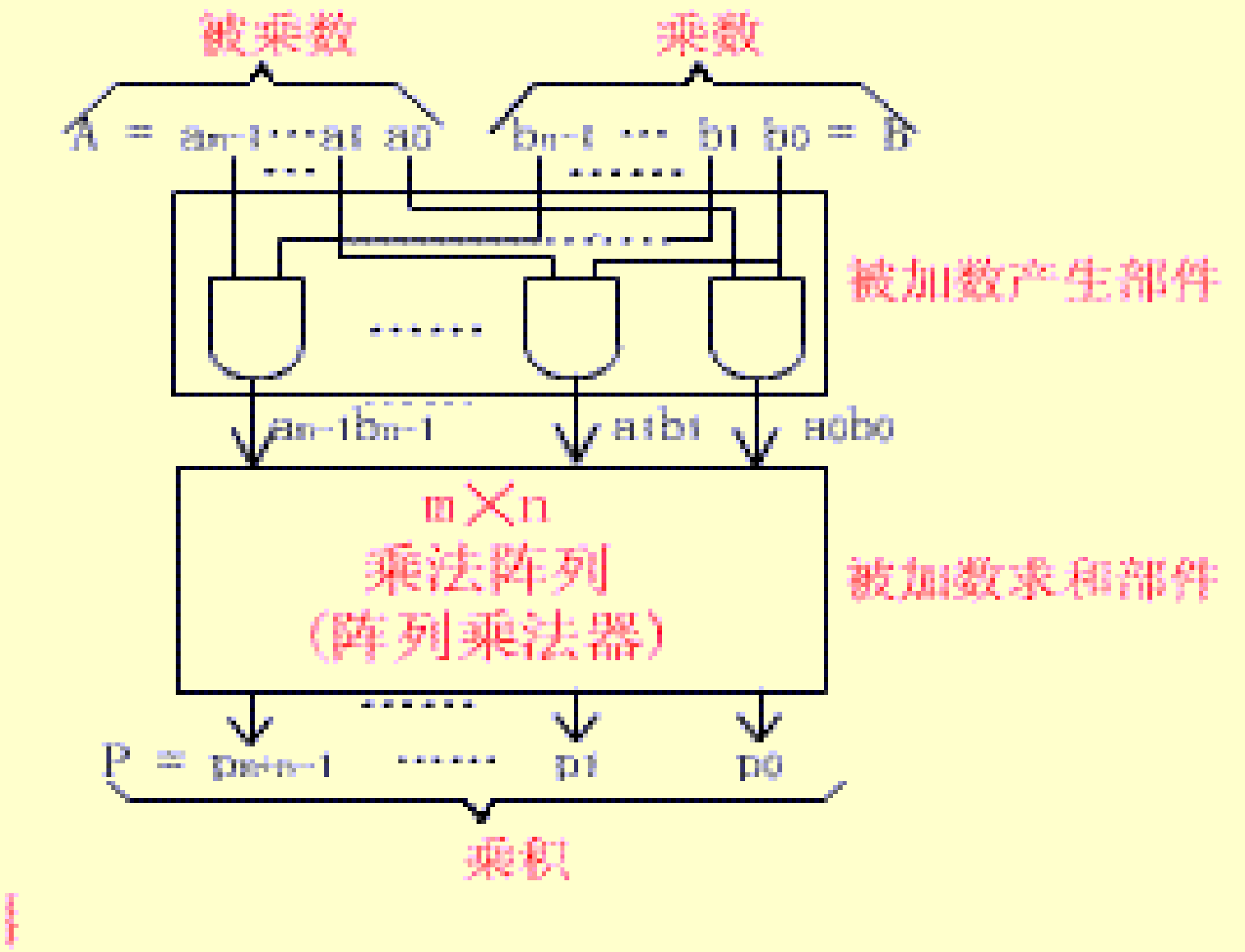
$$p = ab = \left(\sum_{i=0}^{m-1} a_i 2^i\right)\left(\sum_{j=0}^{n-1} b_j 2^j\right) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} (a_i b_j) 2^{i+j} = \sum_{k=0}^{m+n-1} p_k 2^k$$

2.3.2 不带符号的阵列乘法器

- 实现这个乘法过程所需要的操作和人们的习惯方法非常类似：

$$
\begin{array}{rcccccccc}
 & & & & a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 = A \\
 & & & & b_{n-1} & \cdots & b_1 & b_0 = B & \\
\hline
 & & & a_{m-1}b_0 & a_{m-2}b_0 & \cdots & a_1b_0 & a_0b_0 & \\
 & & a_{m-1}b_1 & a_{m-2}b_1 & \cdots & a_1b_1 & a_0b_1 & & \\
 & \iddots & \vdots & \iddots & & \iddots & & & \\
+) & a_{m-1}b_{n-1} & a_{m-2}b_{n-1} & \cdots & a_1b_{n-1} & a_0b_{n-1} & & & \\
\hline
p_{m+n-1} & p_{m+n-2} & p_{m+n-3} & \cdots & p_{n-1} & \cdots & p_1 & p_0 = P &
\end{array}
$$

2.3.2 不带符号的阵列乘法器



2.3.2 不带符号的阵列乘法器

- 上述过程说明了在m位乘n位不带符号整数的阵列乘法中“加法—移位”操作的被加数矩阵。每一个部分乘积项(位积)a_ib_j叫做一个被加数。这 m×n 个被加数

$$\{ a_ib_j \mid 0 \le i \le m-1 \text{ 和 } 0 \le j \le n-1 \}$$

- 可以用m×n个“与”门并行地产生。由此说明设计高速并行乘法器的基本问题，就在于缩短被加数矩阵中每列所包含的1的加法时间。

- 下图是一个5×5（m×n)不带符号的阵列乘法器的逻辑电路图：

2.3.2 不带符号的阵列乘法器

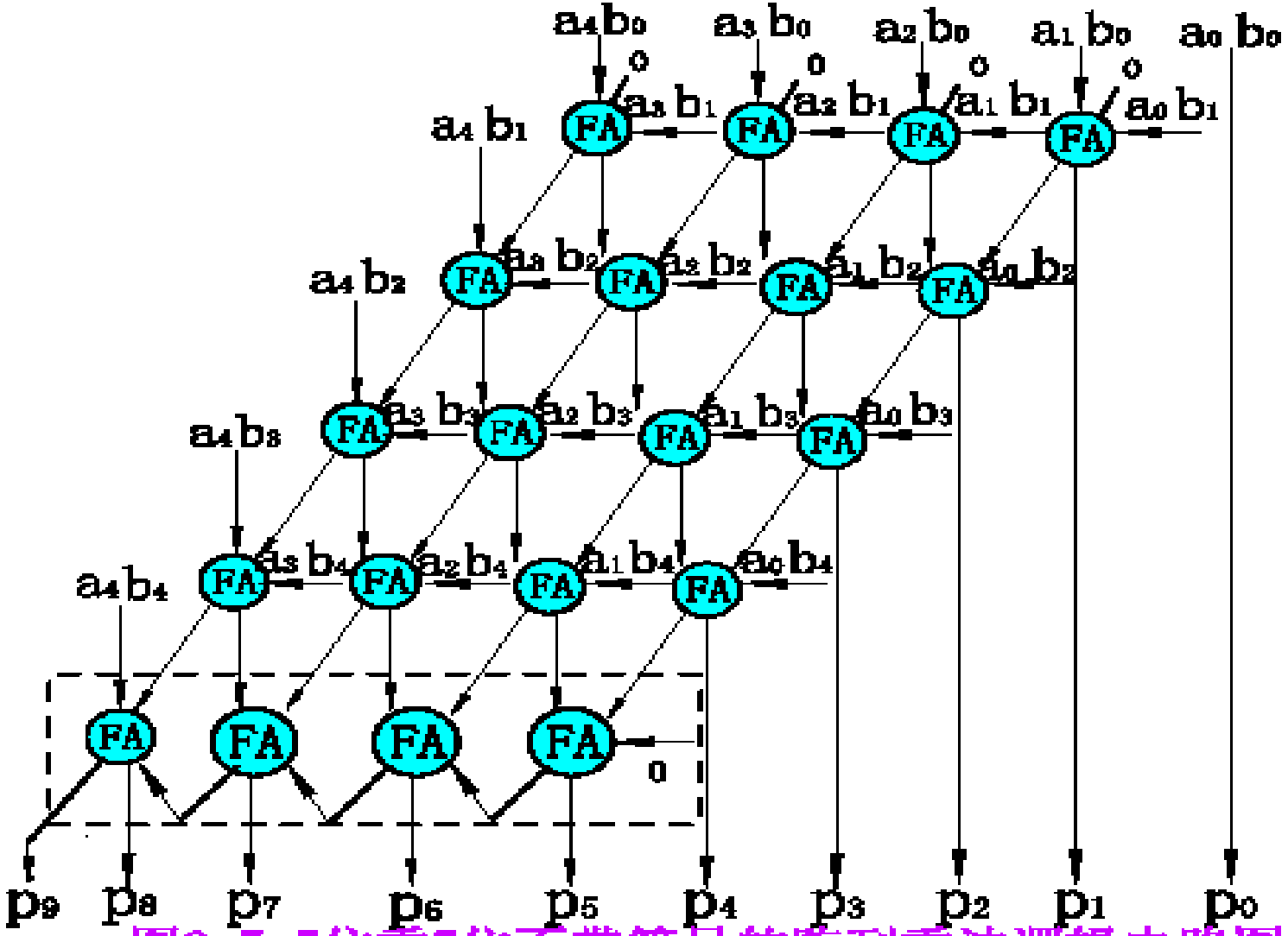


图2.5 5位乘5位不带符号的阵列乘法逻辑电路图

2.3.2 不带符号的阵列乘法器

- 这种乘法器要实现n位×n位时，需要$n(n-1)$个全加器和n^2个“与”门。该乘法器的总的乘法时间可以估算如下：令T_a为“与门”的传输延迟时间，T_f为全加器(FA)的进位传输延迟时间，假定用2级“与非”逻辑来实现FA的进位链功能，那么我们就有：

$$T_a = T,\ T_f = 2T$$

- 从上图可知，最坏情况下延迟途径，即是沿着矩阵p_4垂直线和最下面的一行。因而得n位×n位不带符号的阵列乘法器总的乘法时间为：

$$tm = T+(n-1)6T+(n-1)2T+3T=(8n-4)T$$

2.3.2 不带符号的阵列乘法器

- 从上图可知，最坏情况下延迟途径，即是沿着矩阵p_4垂直线和最下面的一行。因而得*n*位×*n*位不带符号的阵列乘法器总的乘法时间为：

$$tm = T+(n-1)6T+(n-1)2T+3T=(8n-4)T$$

- 斜线是进位链，用(n-1)2T+3T可以完成
- 进位链完成之后，最后一行进行加法运算，共有n-1个斜线，每个斜线最后都需要进行加法运算，加法运算的输入有上面的和输出、斜线的进位、右侧的加法器的进位，所以需要6T，而不是3T，共需要(n-1)*6T

2.3.2 不带符号的阵列乘法器

- 例：已知两个不带符号的二进制整数A = 11011，B = 10101，求每一部分乘积项a_ib_j的值与$p_9p_8……p_0$的值。

```
                    1 1 0 1 1 =A(27₁₀)
×                   1 0 1 0 1 =B(21₁₀)
-------------------------------------
                    1 1 0 1 1
                  0 0 0 0 0
                1 1 0 1 1
              0 0 0 0 0
+           1 1 0 1 1
-------------------------------------
      1 0 0 0 1 1 0 1 1 1  = P
```

$$a_4b_0=1\ a_3b_0=1\ a_2b_0=0\ a_1b_0=1\ a_0b_0=1$$
$$a_4b_1=0\ a_3b_1=0\ a_2b_1=0\ a_1b_1=0\ a_0b_1=0$$
$$a_4b_2=1\ a_3b_2=1\ a_2b_2=0\ a_1b_2=1\ a_0b_2=0$$
$$a_4b_3=0\ a_3b_3=0\ a_2b_3=0\ a_1b_3=0\ a_0b_3=0$$
$$a_4b_4=1\ a_3b_4=1\ a_2b_4=0\ a_1b_4=1\ a_0b_4=1$$

$$P=p_9p_8p_7p_6p_5p_4p_3p_2p_1p_0=1000110111\ (567_{10})$$

2.3.3 带符号的阵列乘法器

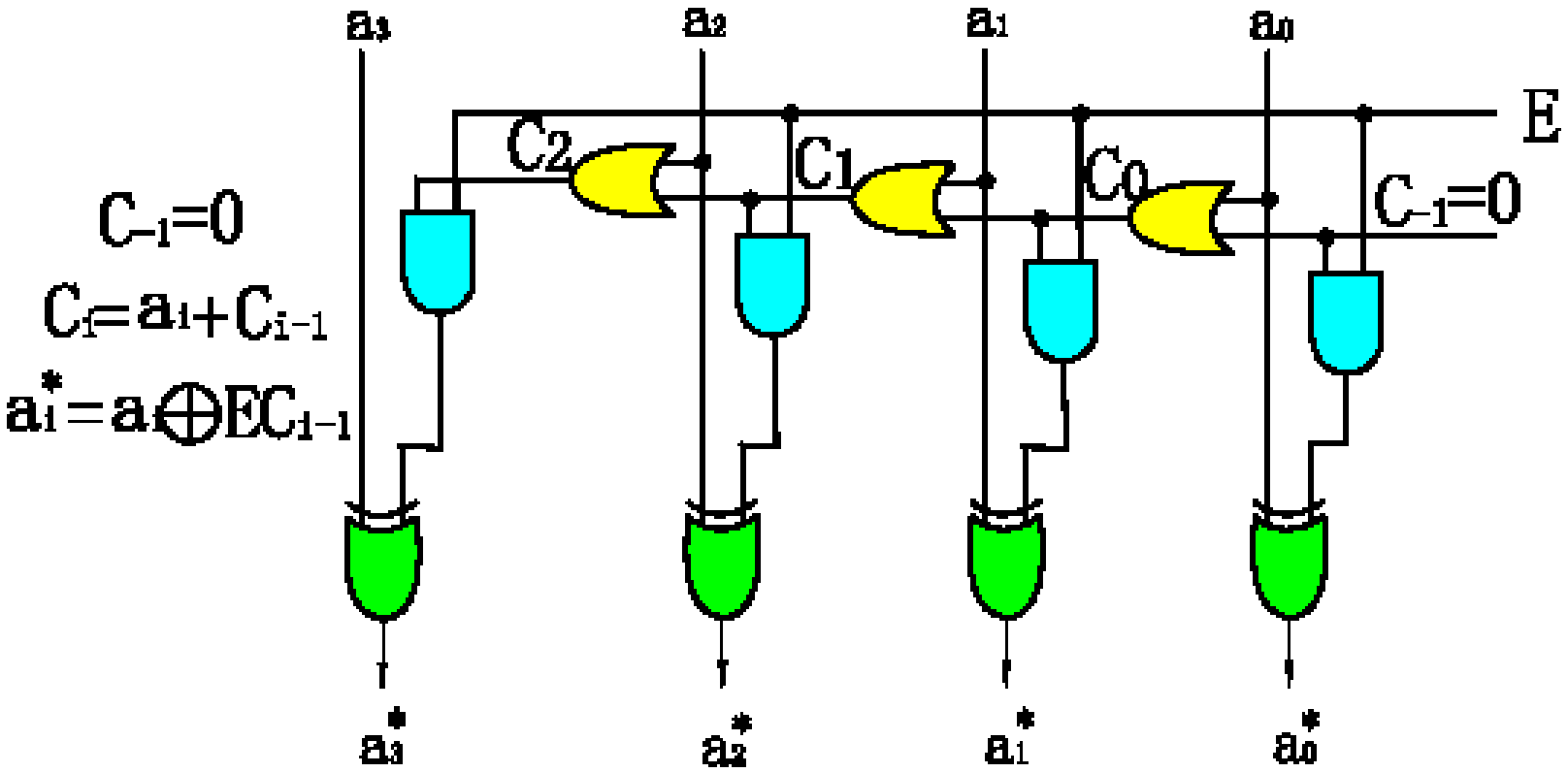
- 一、对2求补器电路

- 我们先来看看算术运算部件设计中经常用到的求补电路。下图示出一个具有使能控制的二进制对2求补器电路，其逻辑表达式如下：

$$C_{-1} = 0, \quad C_i = a_i + C_{i-1}$$
$$a_i^* = a_i \oplus \mathrm{E}\, C_{i-1}, \quad 0 \le i \le n$$

2.3.3 带符号的阵列乘法器

- 一、对2求补器电路



对2求补时，采用按位扫描技术，进行求补的方法就是从数的最右端a_0开始，由右向左，直到找出第一个“1”，例如，$a_i=1$，$0<i<n$，这样，a_i以右的每一个输入位，包括a_i自己，都保持不变，而a_i以左的每一个输入位都求反，即1变0，0变1。

2.3.3 带符号的阵列乘法器

- 一、对2求补器电路

- 在对2求补时，要采用按位扫描技术来执行所需要的求补操作。令$A = a_n...a_1a_0$是给定的$(n+1)$位带符号的数，要求确定它的补码形式。进行求补的方法就是从数的最右端a_0开始，由右向左，直到找出第一个“1”，例如 $a_i=1$，$0 \le i \le n$。这样，a_i以左的每一个输入位都求反，即1变0，0变1。最右端的起始链式输入C_{-1}必须永远置成“0”。当控制信号线E为“1”时，启动对2求补的操作。当控制信号线E为“0”时，输出将和输入相等
- 显然，我们可以利用符号位来作为控制信号。

2.3.3 带符号的阵列乘法器

- 一、对2求补器电路

- 例：在一个4位的对2求补器中，如果输入数为1010，那么输出数应是0110，其中从右算起的第2位，就是所遇到的第一个“1”的位置。
- 用这种对2求补器来转换一个(n + 1)位带符号的数，所需的总时间延迟为

$$t_{TC} = n \cdot T + 4T = (n + 4)T$$

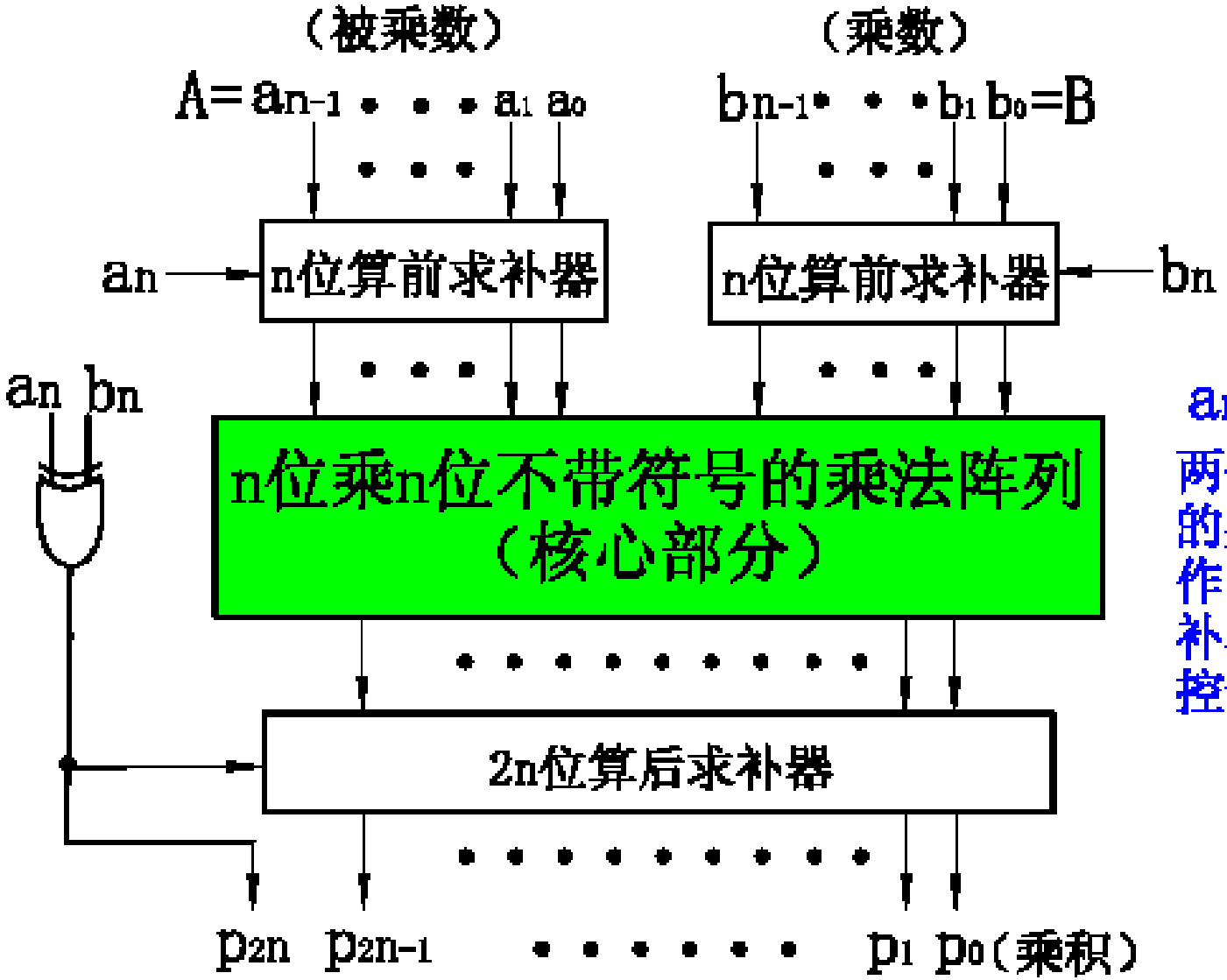
- 其中每个扫描级需T延迟，而4T则是由于“与”门和“异或”门引起的。

2.3.3 带符号的阵列乘法器

- 二、符号求补的阵列乘法器

- 下图是(n + 1)×(n + 1)位带求补器的阵列乘法器逻辑方框图。
- 通常，把包括这些求补级的乘法器又称为符号求补的阵列乘法器。在这种逻辑结构中，共使用三个求补器。其中两个算前求补器的作用是：将两个操作数A和B在被不带符号的乘法阵列(核心部件)相乘以前，先变成正整数。而算后求补器的作用则是：当两个输入操作数的符号不一致时，把运算结果变成带符号的数。

2.3.3 带符号的阵列乘法器



an和bn两个符号位的异或结果作为算后求补器的使能控制信号。

图2.7 （n+1）位乘（n+1）位带补级的阵列乘法器

2.3.3 带符号的阵列乘法器

- 二、符号求补的阵列乘法器

- 上面所示的带求补级的阵列乘法器既适用于原码乘法，也适用于间接的补码乘法。不过在原码乘法中，算前求补和算后求补都不需要，因为输入数据都是立即可用的。而间接的补码阵列乘法却需要增加三个硬件求补器。
- 实际上我们可以看到带符号的阵列乘法器其内部仍是一个基本的原码阵列乘法器，只是对输入的补码数据在乘前进行了值还原，同时在乘后再次将乘积数转换为补码形式输出。

2.3.3 带符号的阵列乘法器

- 例：设 x = +15，y = -13，用带求补器的原码阵列乘法器求出乘积 x·y = ?
- 设最高位为符号位，则输入数据为

$[x]_{补} = 01111$　　$[y]_{补} = 10011$

符号位单独考虑，经过算前求补级后

$|x| = 1111,$　　$|y| = 1101$

```
              1 1 1 1
 ×            1 1 0 1
-------------------------
              1 1 1 1
            0 0 0 0
          1 1 1 1
 +    1 1 1 1
-------------------------
    1 1 0 0 0 0 1 1
```

符号位运算：0⊕1=1

2.3.3 带符号的阵列乘法器

- 算后经求补级输出并加上乘积符号位1,
  - 乘积的补码值为　100111101。
  - 则原码乘积值为　111000011。
- 换算成二进制数真值是

$$x \cdot y = (-11000011)_2 = (-195)_{10}$$

- 十进制数验证:
- $x \times y = 15 \times (-13) = -195$相等。