数字集成电路

第十一讲 运算功能块

数字处理器结构

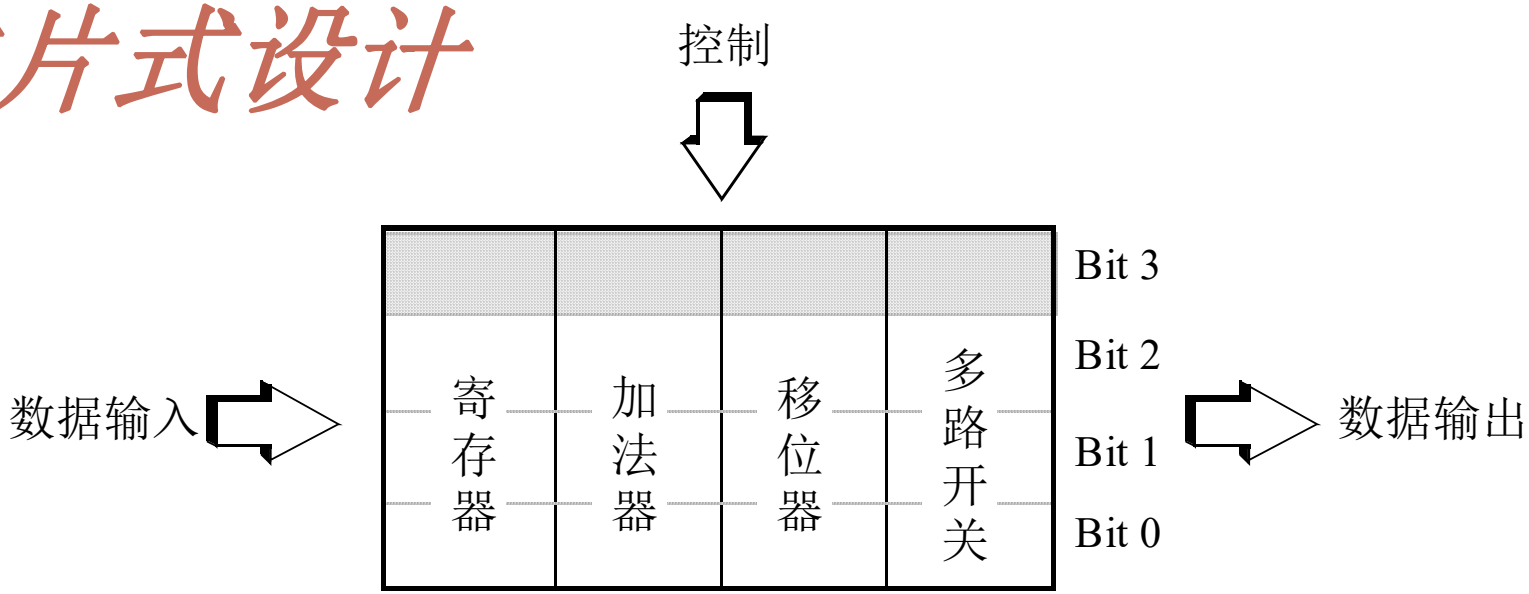
- 数据通路（核心）

  如算数运算器（加法、乘法、比较、移位）
- 存储器

  RAM，ROM，Buffers，Shift registers
- 控制器

  有限状态机、计数器
- 输入/输出模块

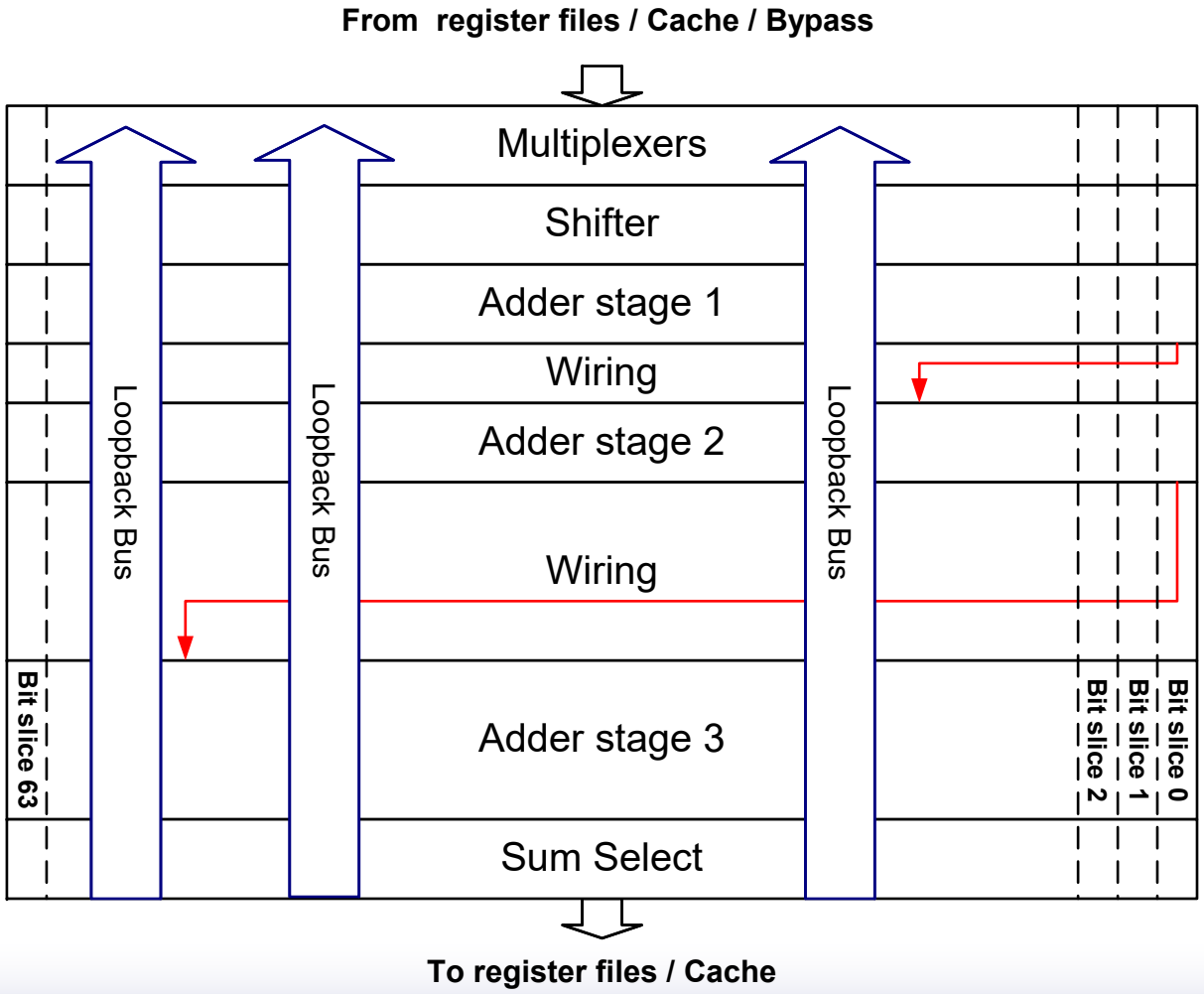
位片式设计



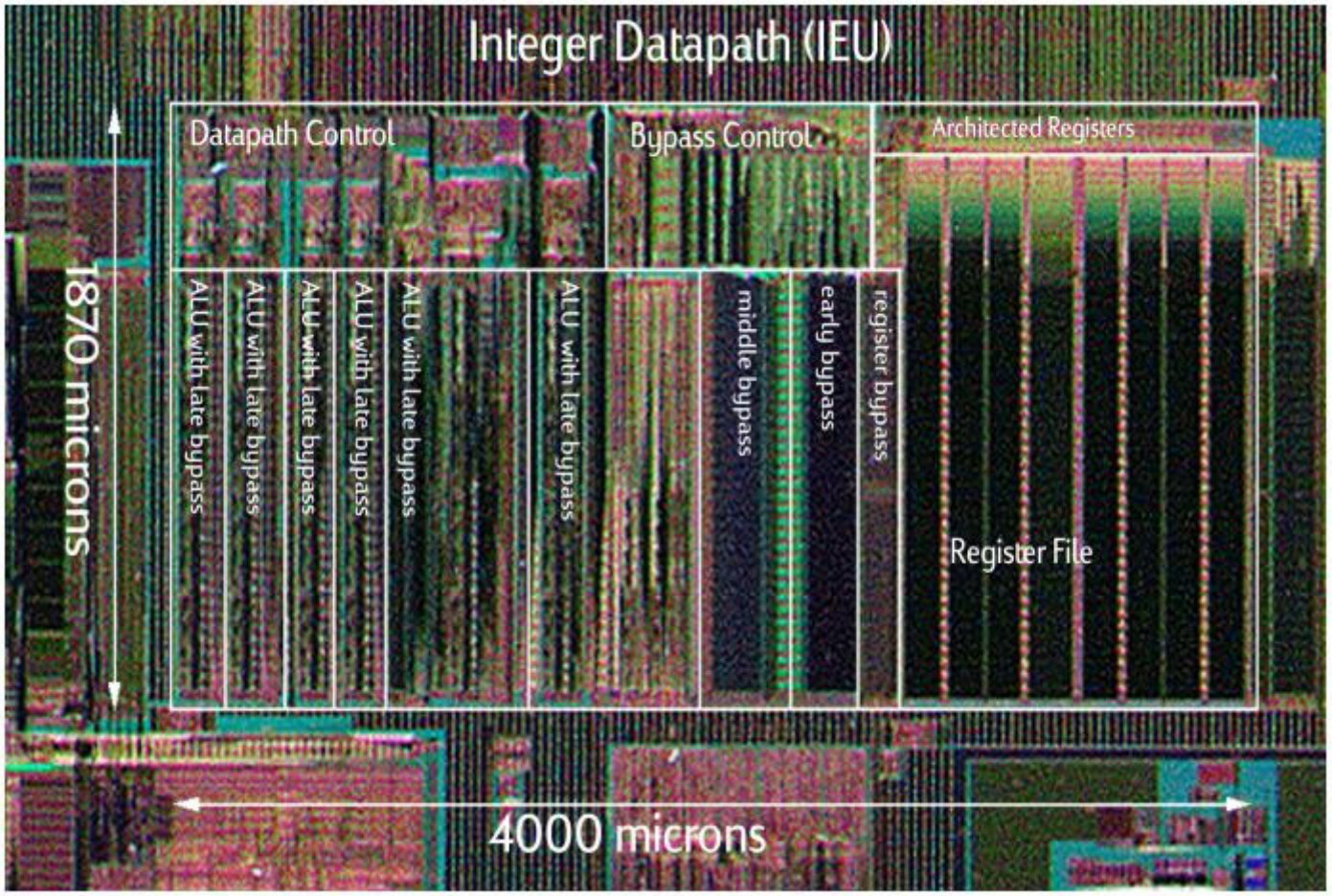
位片式数据通路结构

微处理器中的数据不是对单个位的数字信号进行操作，而是组织成以字（word）为基础的形式。一般来说，微处理器的数据通路是32位或64位宽。例如一个32位的处理器对32位宽的数据字进行操作数据通路含有32个位片，每一个对一位进行操作。所有位的位片或完全相同，或者类似于同样的结构。数据通路的设计者只需集中设计一个位片然后重复操作32次即可。

位片式数据通路

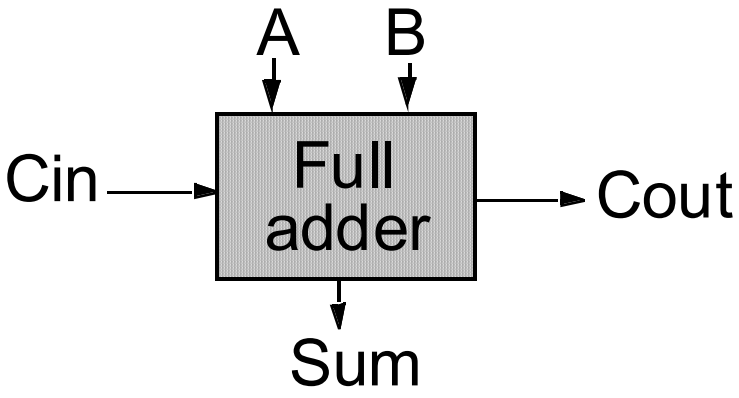

From register files / Cache / Bypass
Multiplexers
Shifter
Adder stage 1
Wiring
Adder stage 2
Wiring
Adder stage 3
Sum Select
Loopback Bus
Loopback Bus
Loopback Bus
Bit slice 63
Bit slice 2
Bit slice 1
Bit slice 0
To register files / Cache

Itanium Integer Datapath



加法器

二进制全加器



全加器真值表

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

$$C_o = AB + BC_i + AC_i$$

$$S = A \oplus B \oplus C_i$$

$$\bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i$$

用P, G, D表示 Sum 和Carry

定义三个中间信号G（进位产生，generate）、D（进位取消，delete）和P（进位传播，propagate）。G=1（D=1）时，将保证在C_o产生（取消）一个进位，而与C_i无关。而P=1将保证有一个进位输入传播至C_o。

$$G=AB$$

$$D=\overline{A}\,\overline{B}$$

$$P=A \oplus B$$

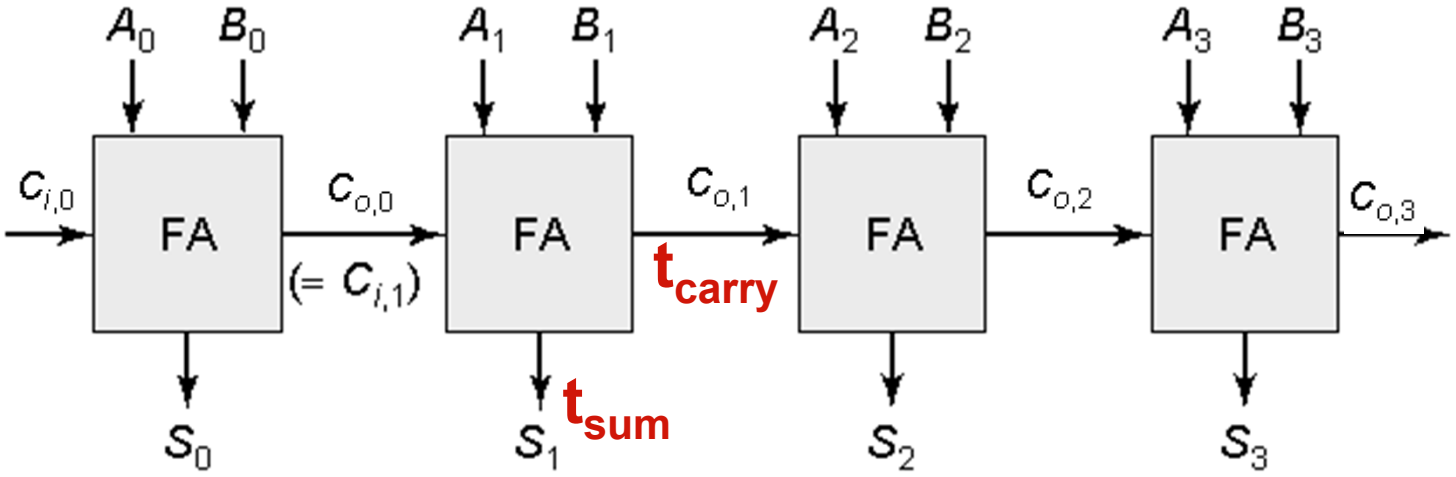
可以把S和C_0重新写成P和G的函数：

$$C_o=AB+BC_i+AC_i \qquad C_o(G,P)=G+PC_i$$

$$S=A \oplus B \oplus C_i \qquad S(G,P)=P \oplus C_i$$

同样，也可以推导出S(D, P)和C_o(D, P)的表达式。

行波进位加法器



关键路径（critical path）的传播延时定义为对所有可能的输入图形在最坏情形下的延时。在逐位进位加法器中，最坏情形的延时发生在当最低有效位(LSB)上产生的进位一直全程传播到最高有效位(MSB)时。延时正比于输入字的位数N并近似为：

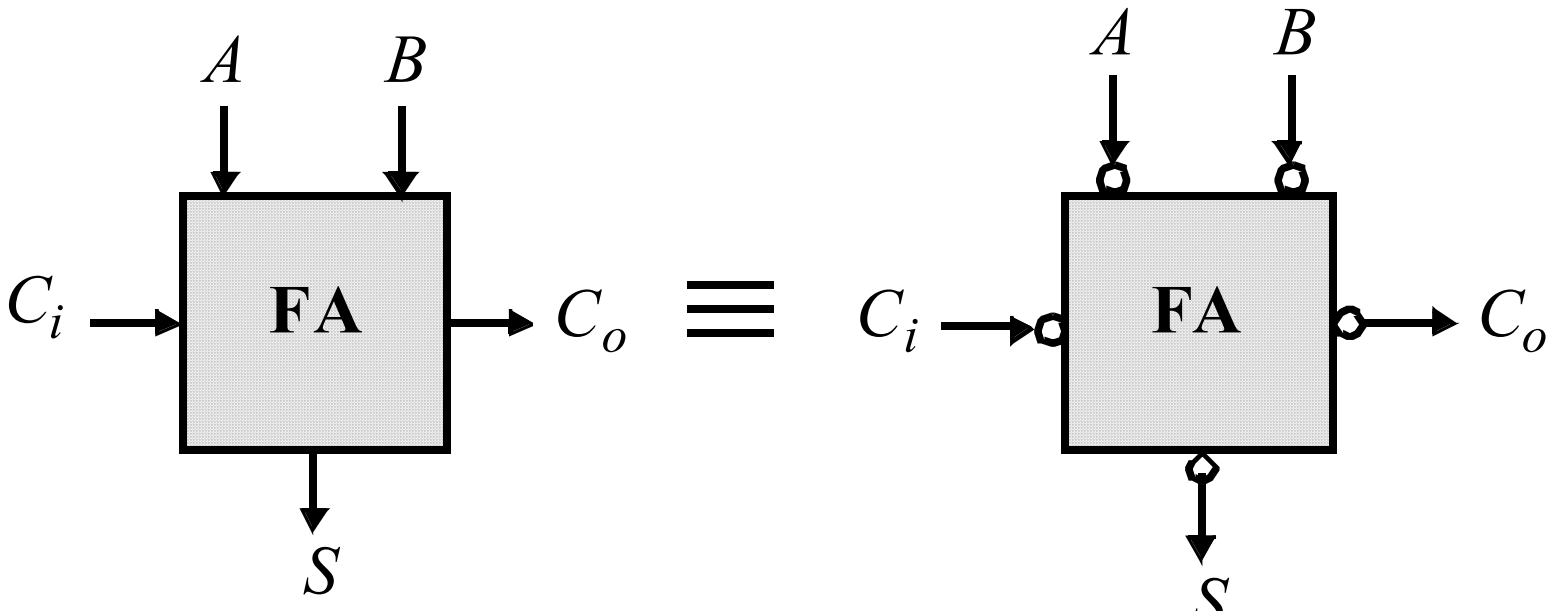
$$t_{adder} = (N\text{-}1)t_{carry} + t_{sum}$$

P413 例11.1

行波进位加法器延时的结论

- 传播延时与级数N成线性关系
  - 在设计当前和未来计算机所希望的宽数据通路（N＝32，64，128）加法器时这一点变得日益重要。
- 在设计一个快速行波进位加法器的全加器单元时，优化t_{carry}比优化t_{sum}重要的多，因为后者对加法器总延时值的影响较小。

加法器的反向特性



$$\bar{S}(A,B,C_i) = S(\bar{A},\bar{B},\bar{C_i})$$

$$\overline{C_o}(A,B,C_i) = C_o(\bar{A},\bar{B},\bar{C_i})$$

全加器电路

实现全加器的电路的一种方法是把逻辑方程直接转变为互补CMOS电路。

$C_0 = AB + BC_i + AC_i$

$S = A \oplus B \oplus C_i$

$\overline{A}\overline{B}C_i + \overline{A}B\overline{C}_i + A\overline{B}\overline{C}_i + ABC_i$

12 + 6 = 18 literals = 36 tr
3 input inverters = 6 tr
2 output inverters = 4 tr

total = 46 tr

进行某些逻辑变换可以帮助减少晶体管的数目。

$$C_o = AB + BC_i + AC_i$$

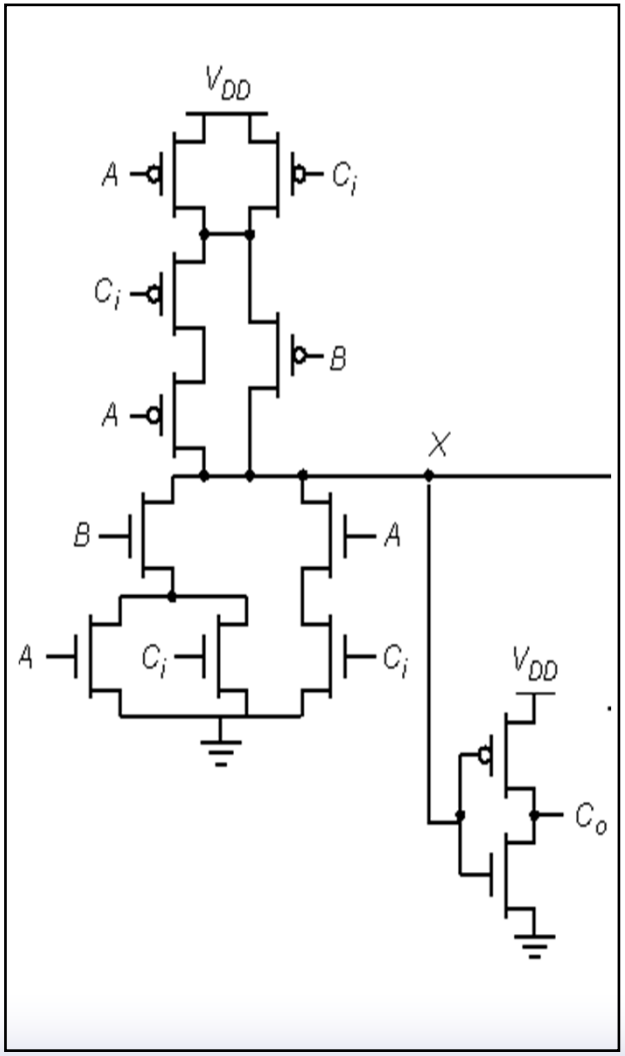
$$S = ABC_i + \overline{C}_o(A + B + C_i)$$

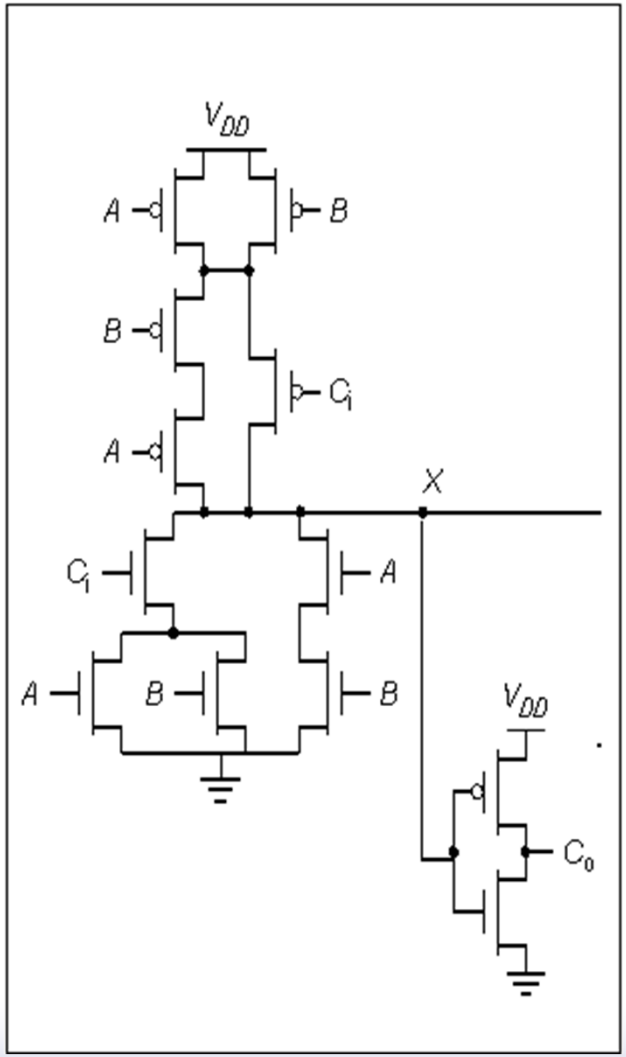
$$S = \overline{ABC_i + \overline{C}_o.(A+B+C_i)}$$

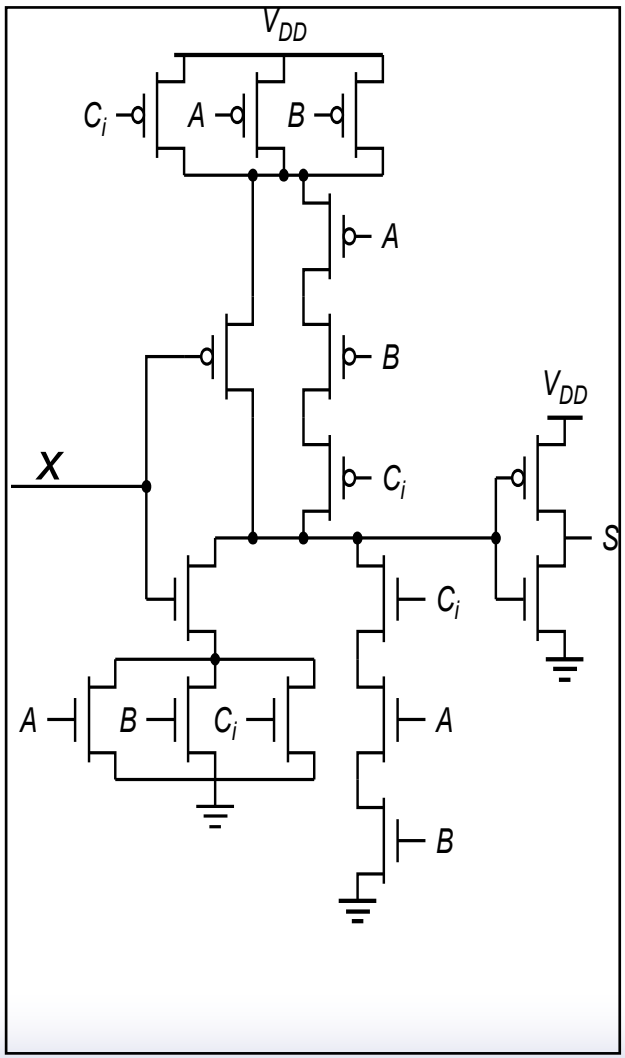
$$C_o = \overline{(A+C_i)B + AC_i}$$

$$S = \overline{ABC_i + \overline{C}_o.(A+B+C_i)}$$

$$C_o = \overline{(A+B)C_i + AB}$$







互补静态CMOS加法器

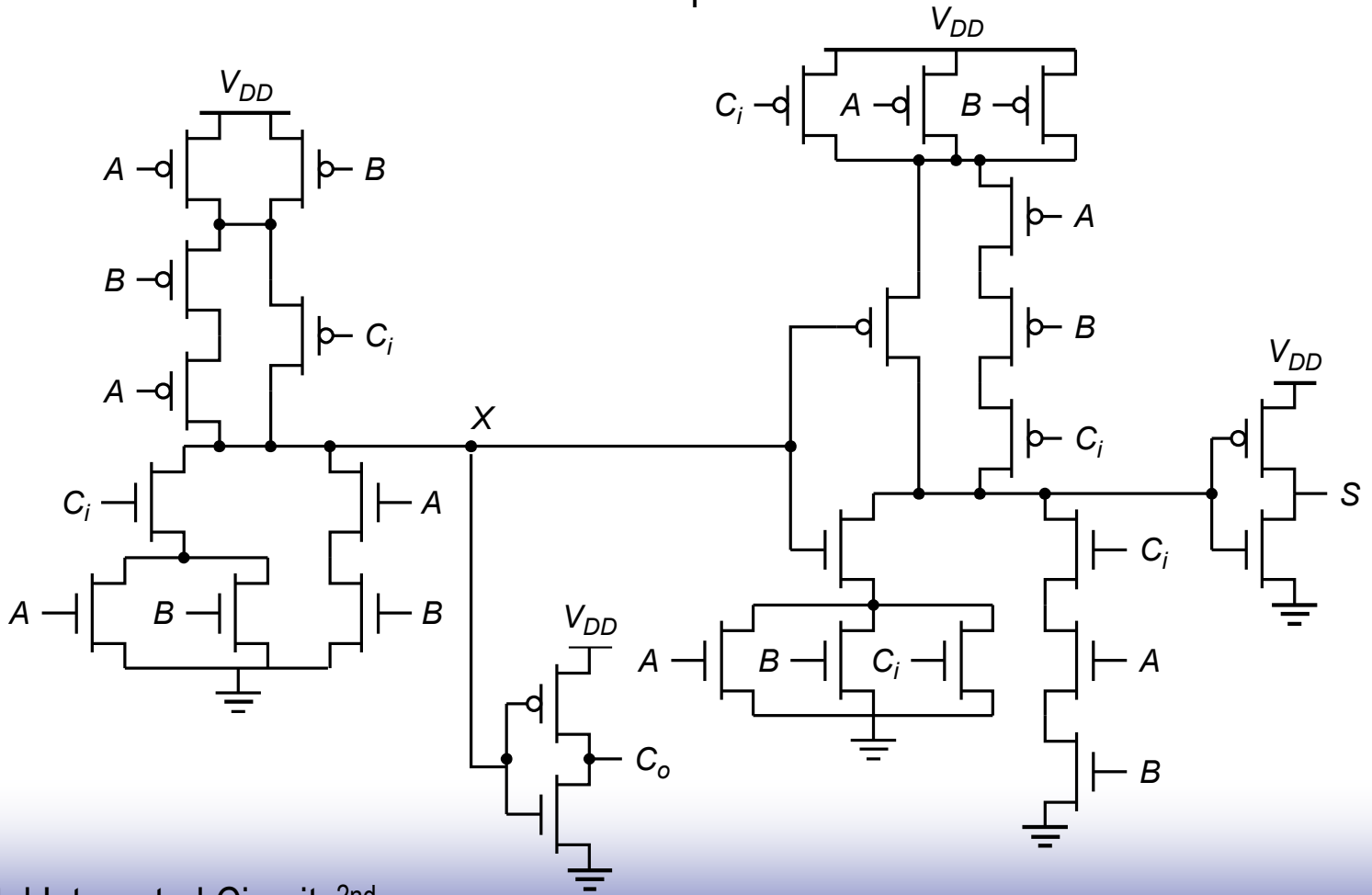
$$S = \overline{ABC + \overline{C_o}.(A+B+C)}$$

$$C_o = \overline{(A+B)C + AB}$$

在进位产生电路中：

（1）C_i逻辑努力为2；

（2）C_i驱动的PMOS和NMOS靠近输出。

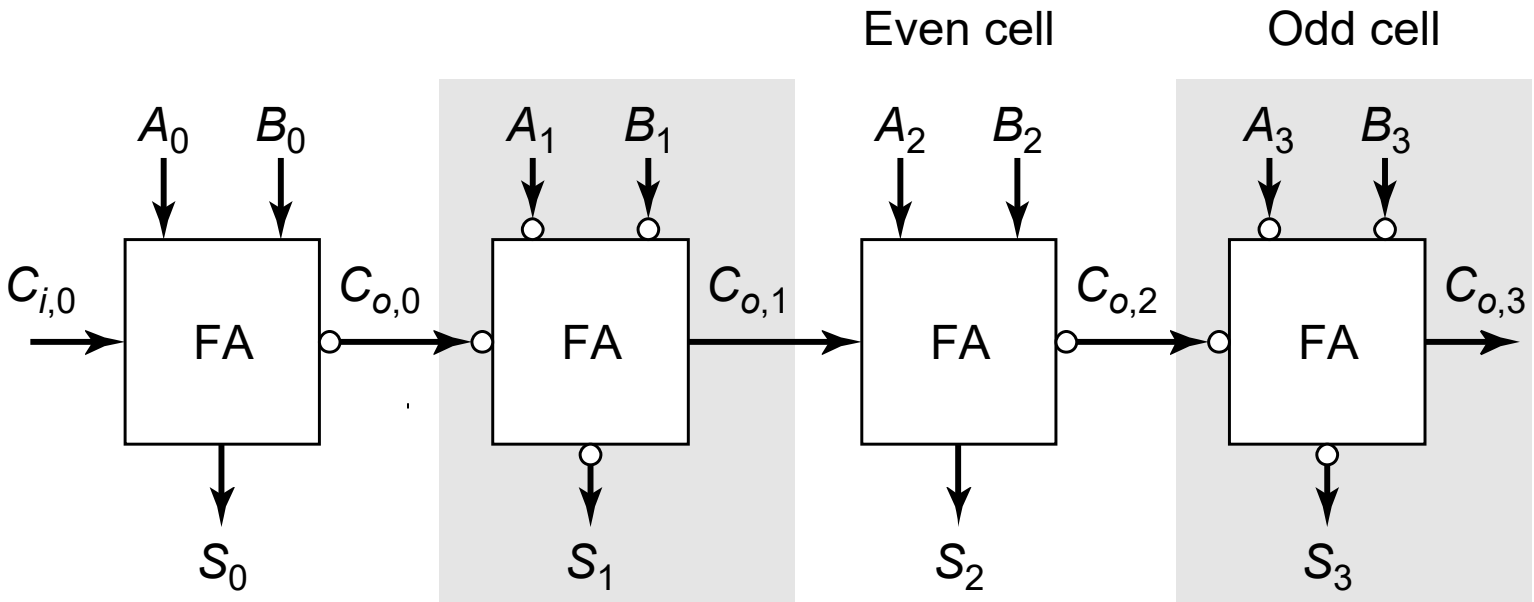


互补静态CMOS加法器

互补静态CMOS加法器速度慢：

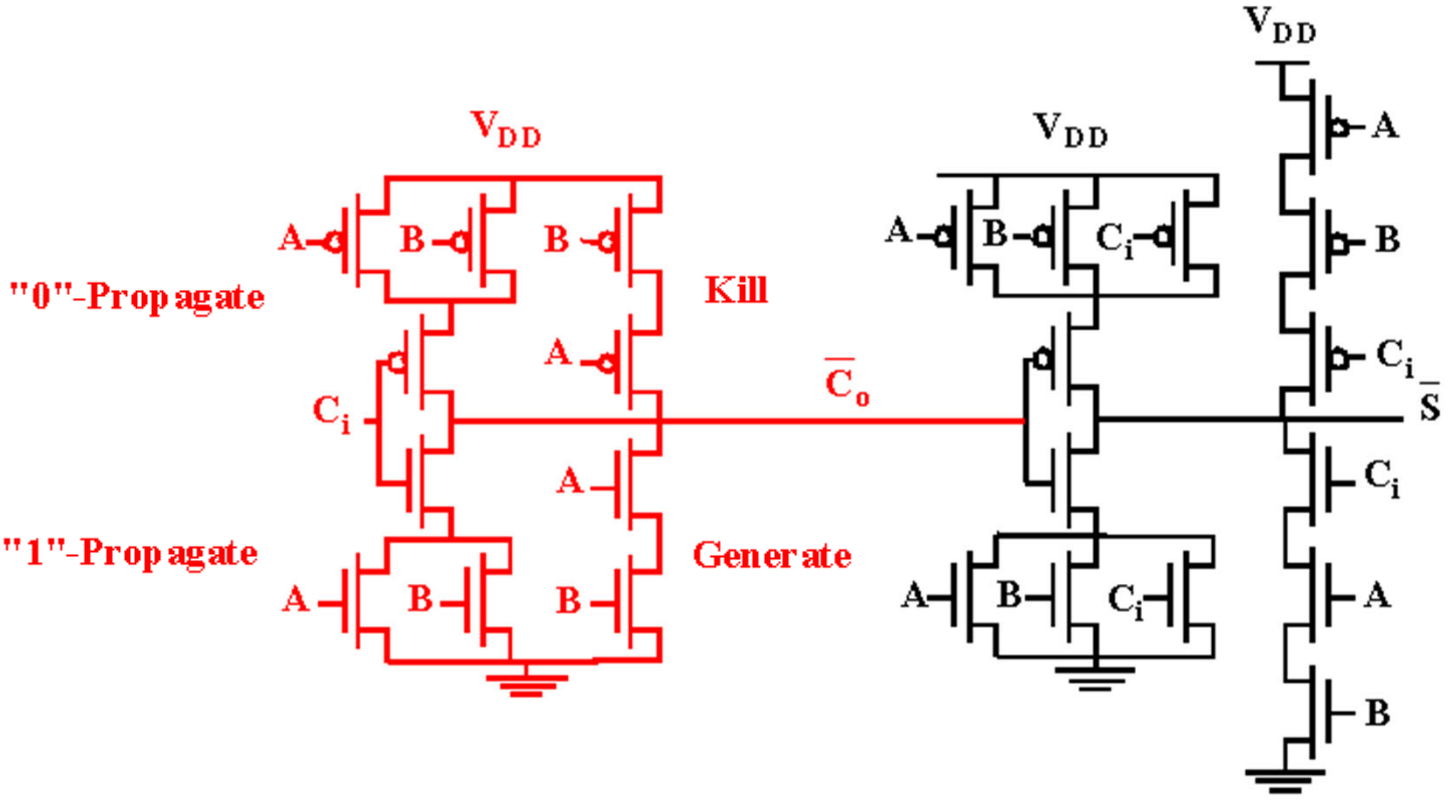
- 在进位（carry）产生与和（sum）产生电路中堆叠了许多PMOS管。
- C_o信号的本征负载电容很大，包括两个扩散电容、6个栅电容，再加上布线电容。
- 在进位产生电路中信号传播通过两个反相级。最小化进位路径的延时是高速加法器电路设计的首要目标。如果进位链输出上的负载很小，两个逻辑级就显得太多，引起额外的延时。
- 和的产生要求一个额外的逻辑级，这一项在逐位进位加法器的传播延时中只出现一次。

缩短关键路径长度



利用加法器的反向特性，把一个全加器单元的所有输入反向则它的所有输出也反相，可以减少进位路径中反相器的数目。

镜像加法器



如果$D=\overline{A}\,\overline{B}=1$，即A=B=0，$\overline{Co}=1$　　A=B=Ci=0,$\overline{S}=1$

如果$G=AB=1$，即$A=B=1$,$\overline{Co}=0$　　A=B=Ci=1,$\overline{S}=0$

如果$P=A \text{ xor } B=1$，$\overline{Co}=\overline{Ci}$

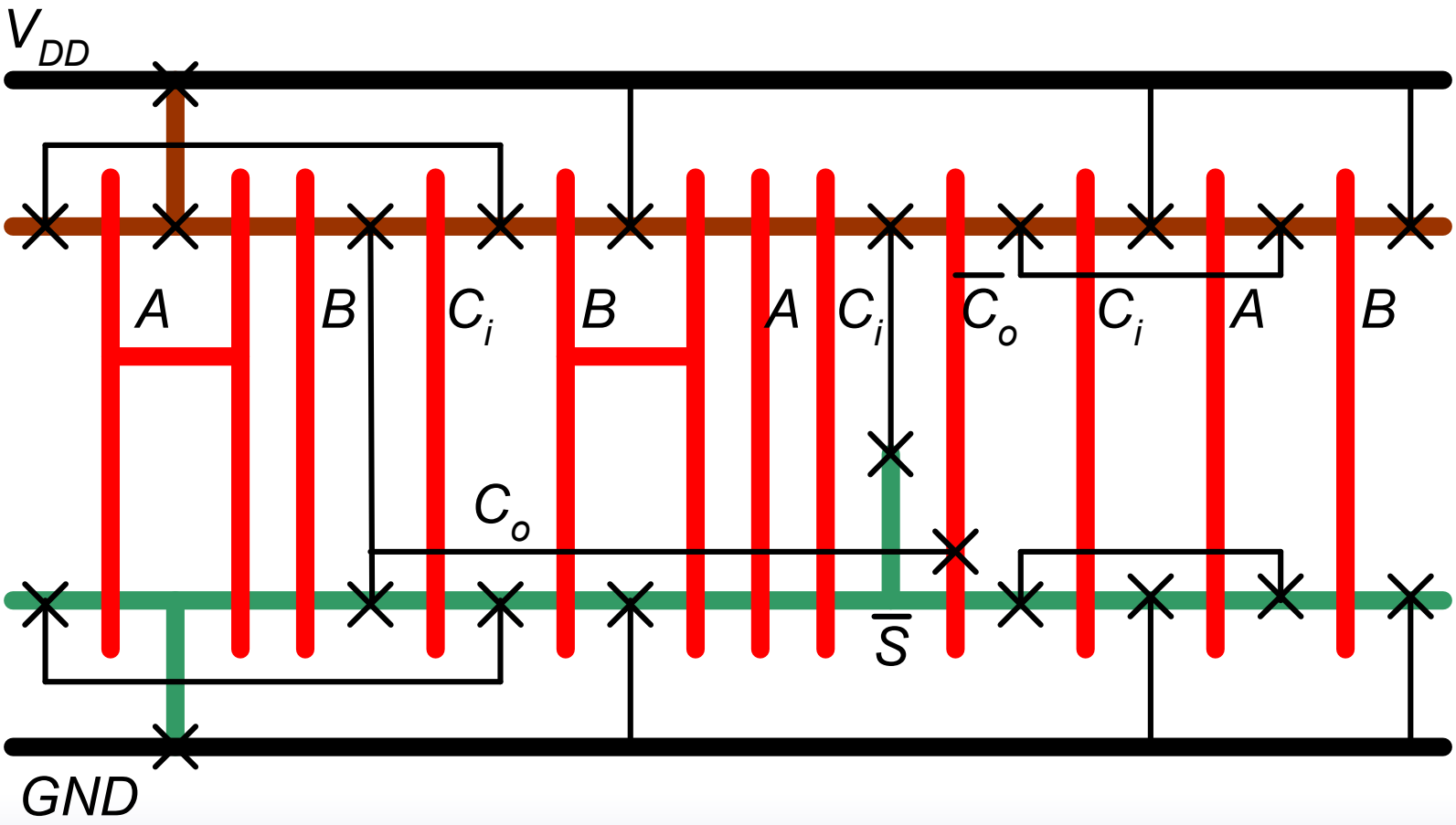
其它情况下，根据真值表可得，$S=\overline{Co}$

镜像加法器

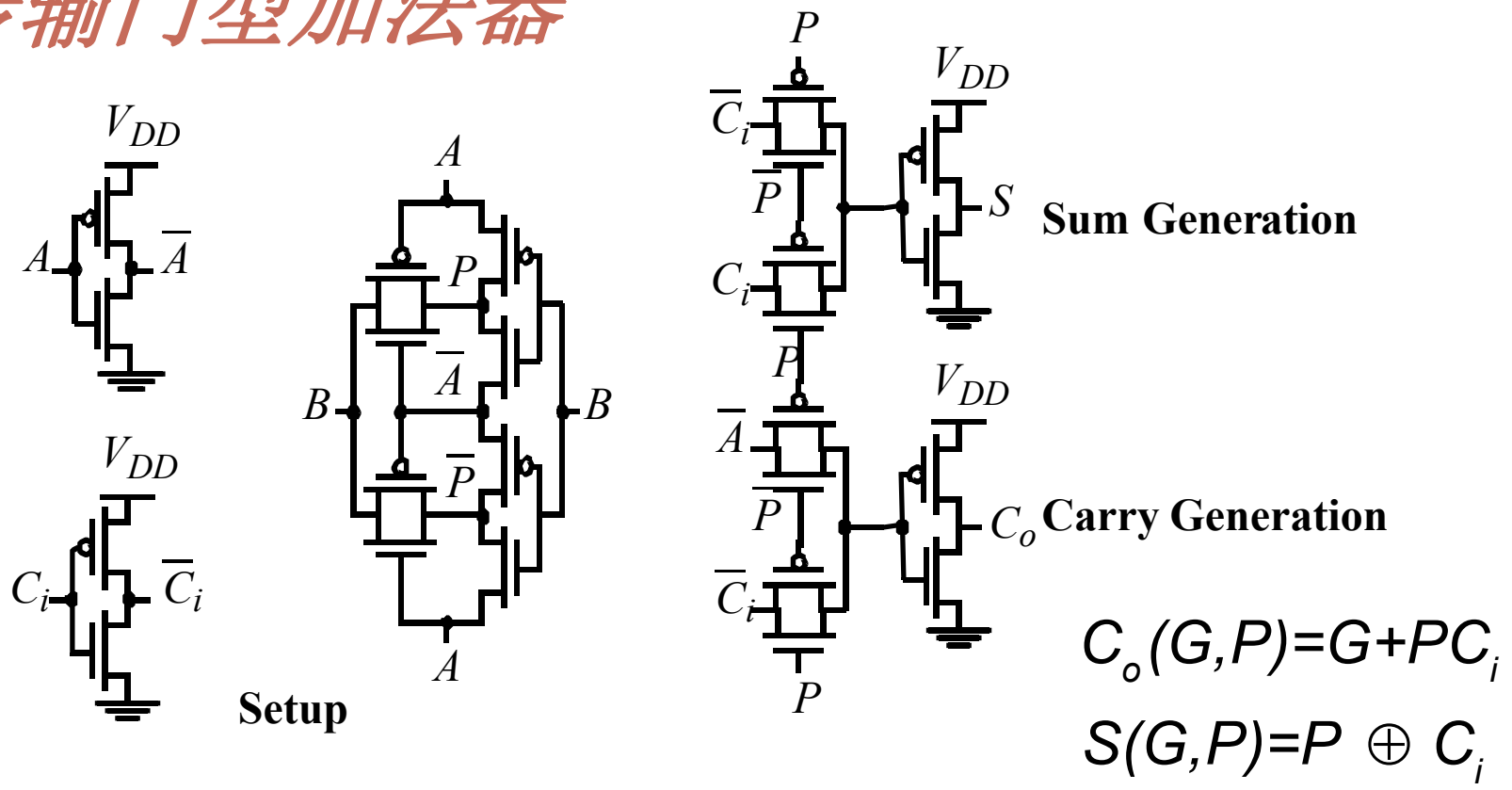
- 镜像全加器单元仅需要24个晶体管。
- NMOS和PMOS链完全对称，由于求和与进位功能的自对偶性，它同样能产生正确的功能。但这一结果使在进位产生电路中最多只有两个晶体管串联。
- 连接Ci的晶体管放在最接近门的输出端处。
- 只有在进位电路中的晶体管才需要优化尺寸以改善速度。求和电路中晶体管都可以采用最小尺寸。
- 当设计该单元版图时，最关键的问题是使节点处$\overline{C}_o$的电容最小，共享扩散区可以减少堆叠（晶体管）的节点电容。
- 确定反相器的尺寸来驱动下一级加法器的Ci输入。

镜像加法器

Stick Diagram



传输门型加法器



$C_o(G,P)=G+PC_i$

$S(G,P)=P \oplus C_i$

根据真值表，P＝1时，Co＝Ci；P＝0时，Co＝A（或B）

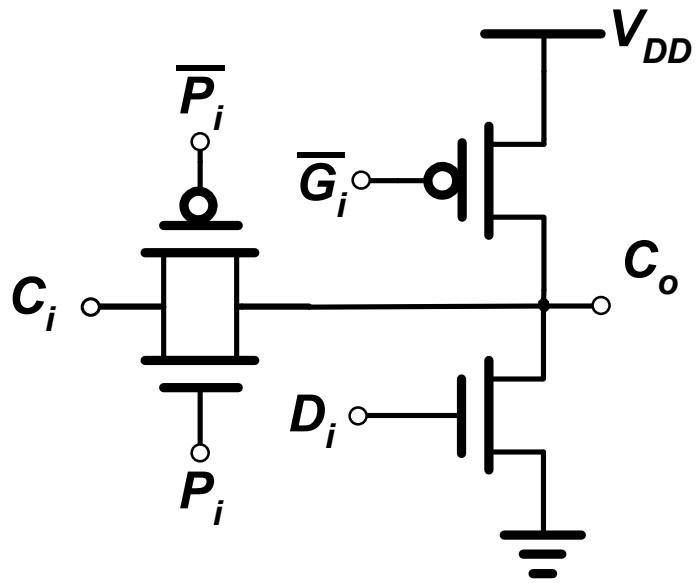
P＝1时，S=Ci；P=0时，S＝$\overline{Ci}$

特点：和输出与进位输出具有近似的延时。

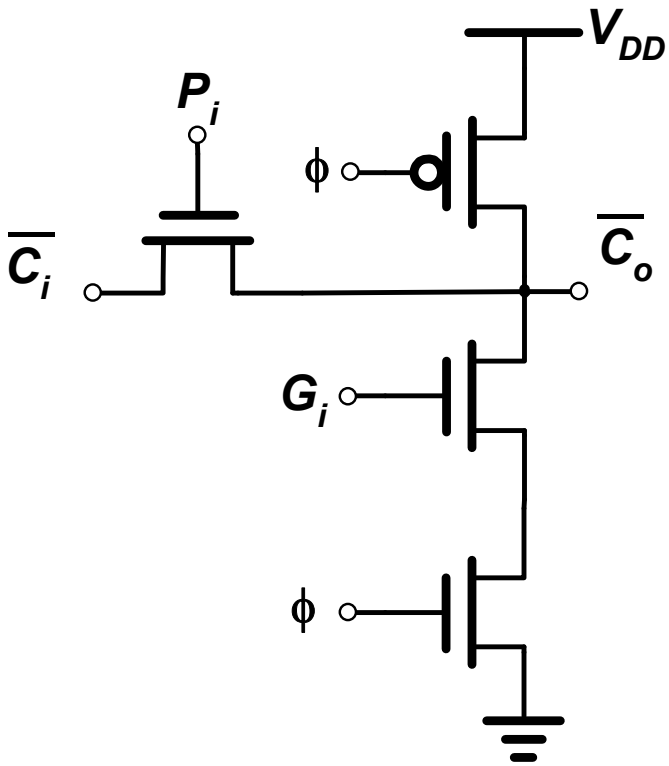
曼彻斯特进位加法器

静态实现，采用进位传播、进位产生和进位消除

根据真值表

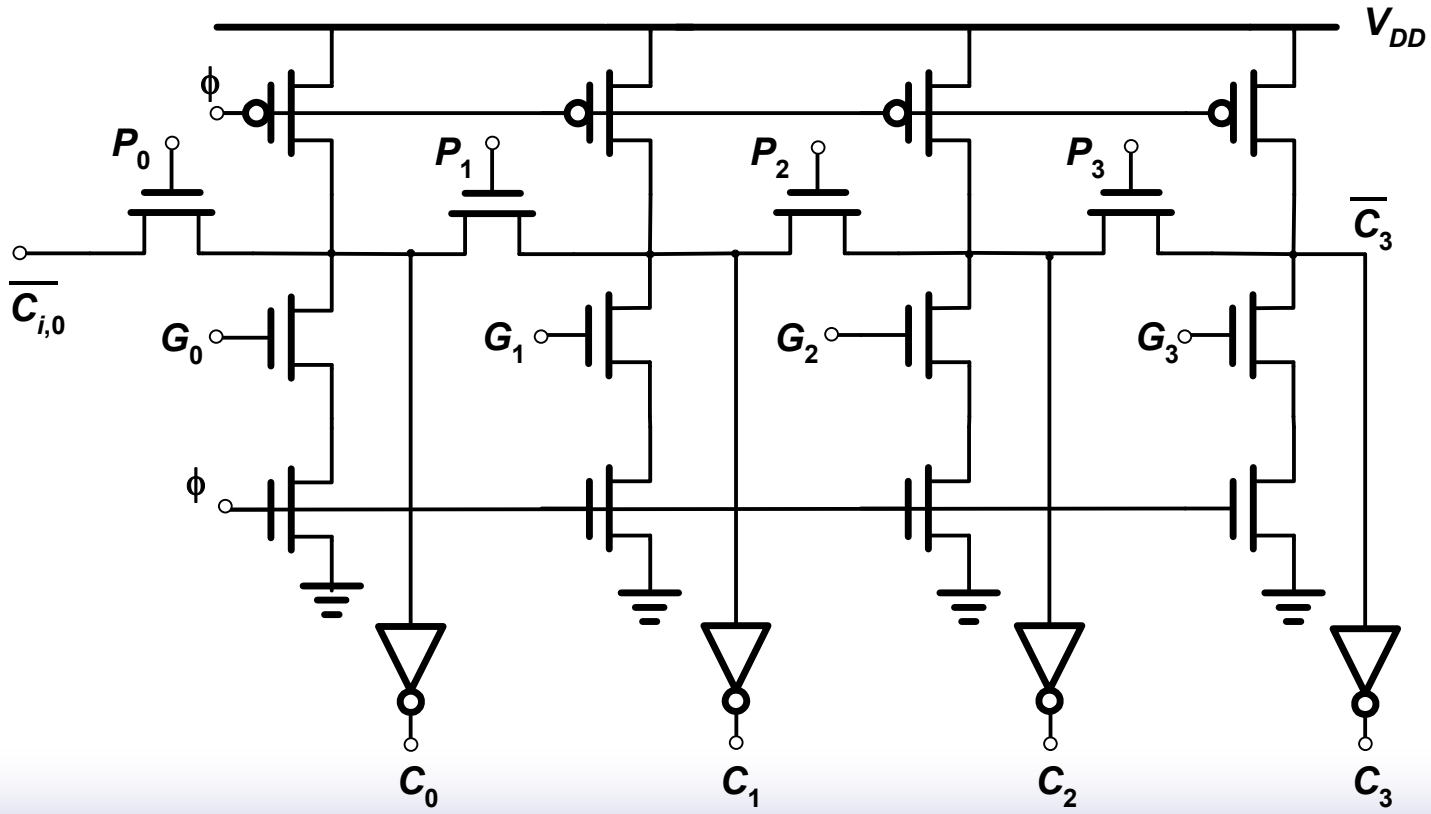


动态实现，只用进位传播和进位产生信号

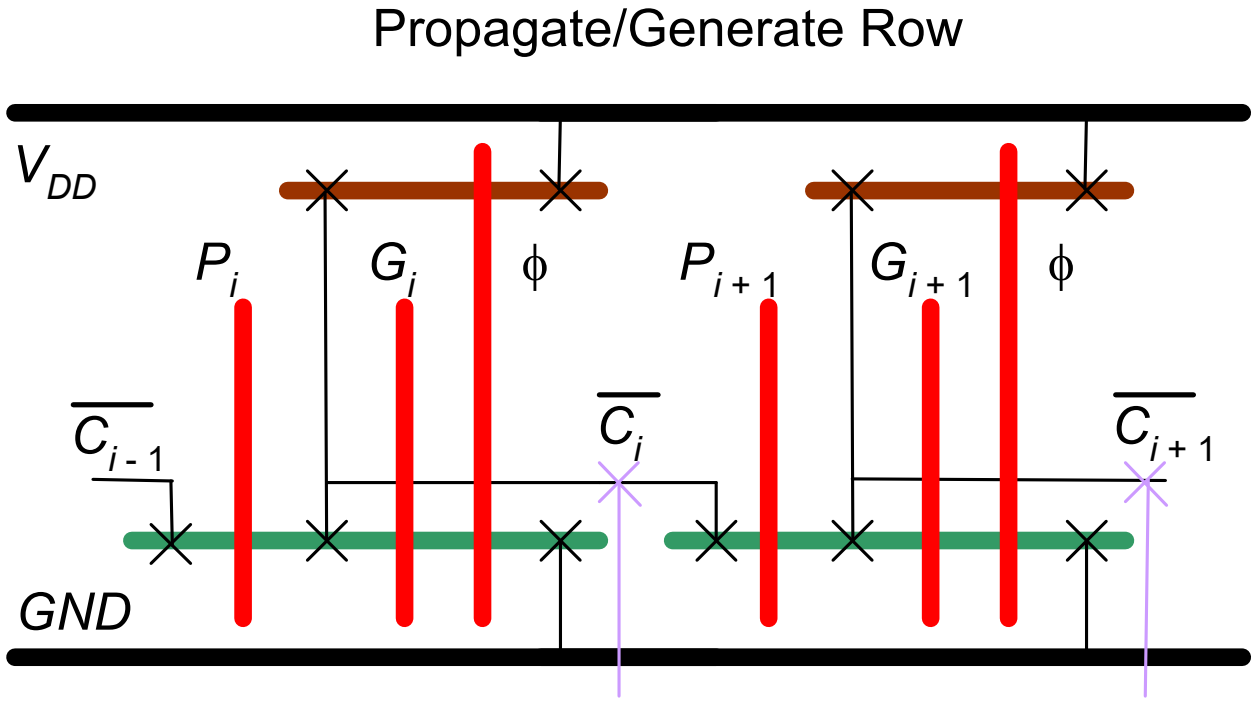


曼彻斯特进位加法器

在预充电阶段（ϕ=0），传输管进位链中的所有中间节点都被预充电到V_{DD}。在求值阶段，当有输入进位且传播信号Pk为高电平时，或者当第k级的进位产生信号为高电平时，节点Ck放电。



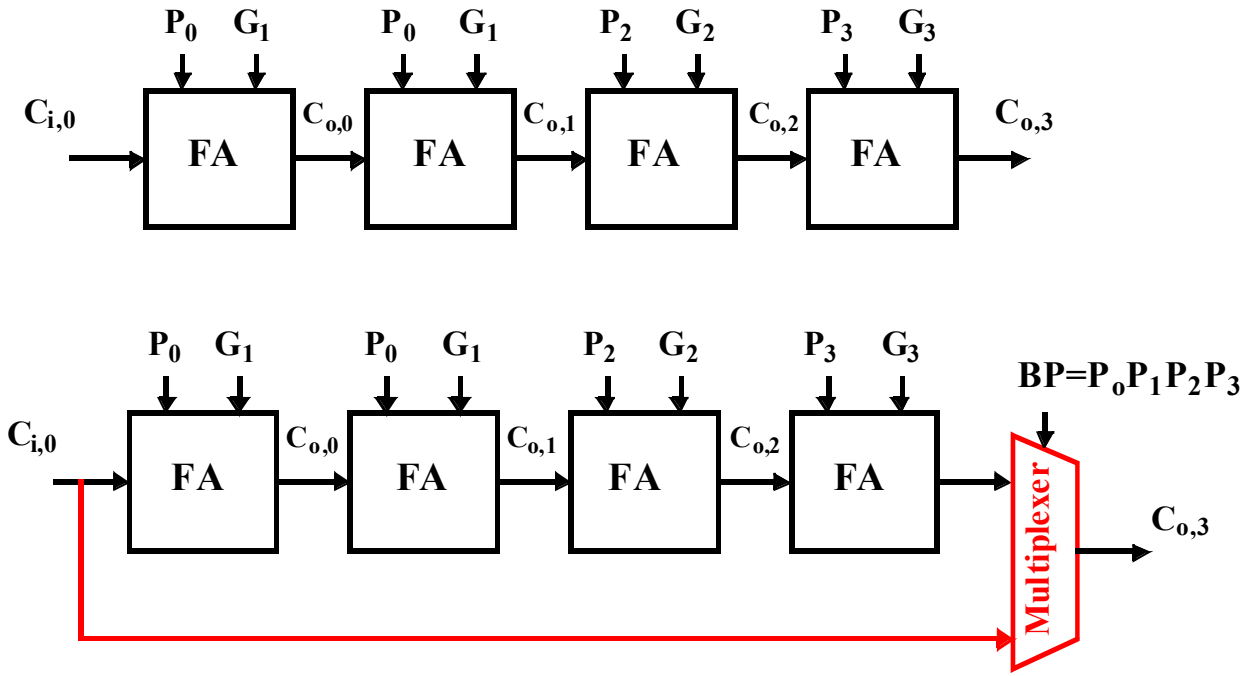
曼彻斯特进位加法器



棍状图

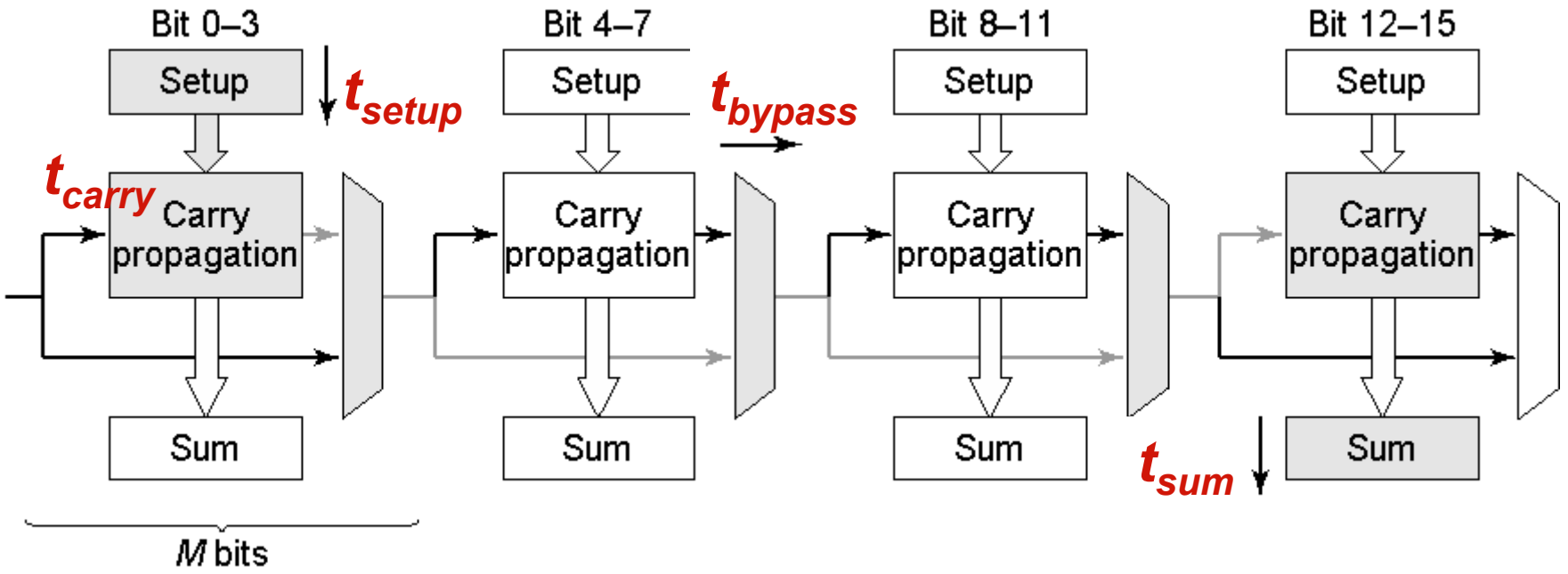
进位旁路加法器

carry-bypass adder or carry-skip adder



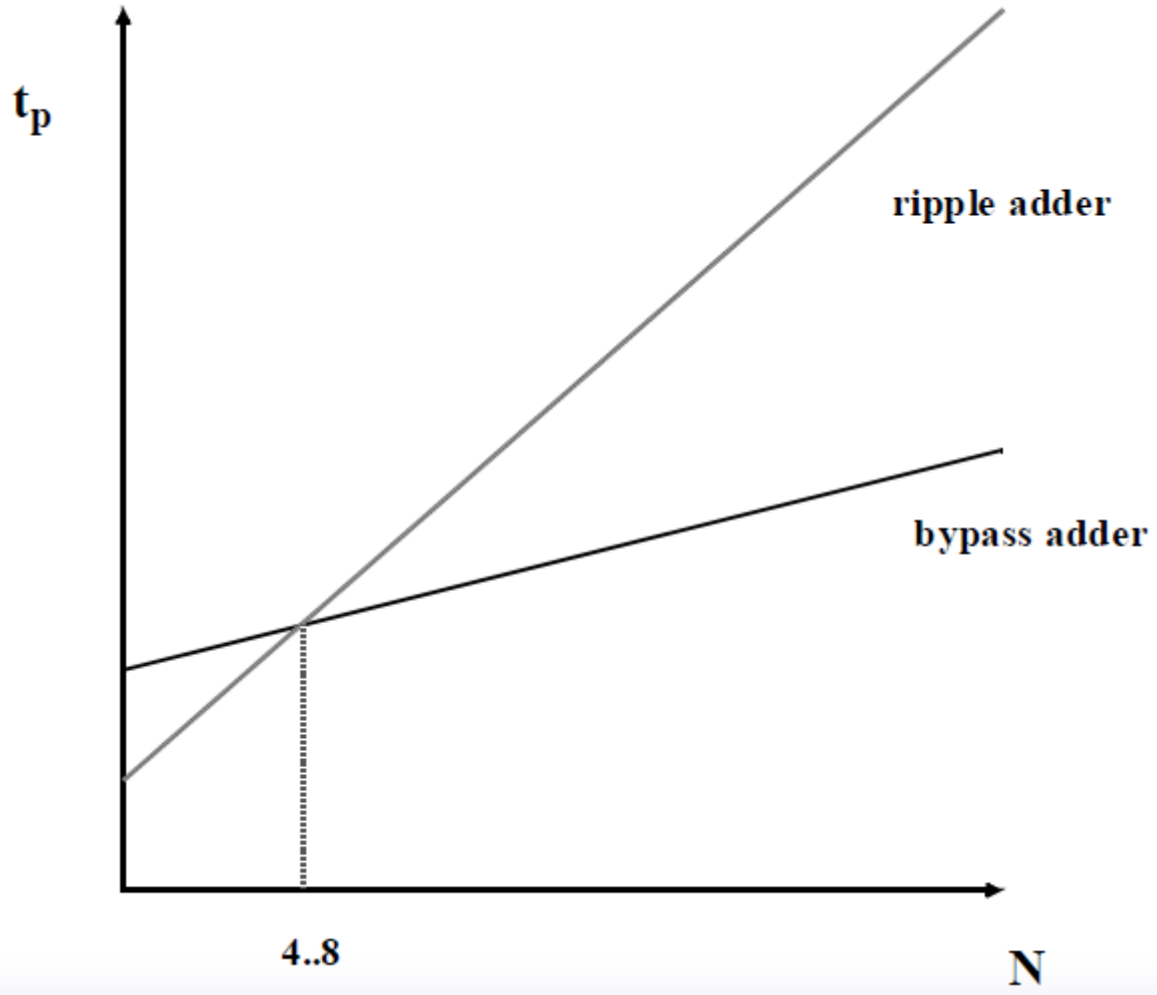
Idea: If (P0 and P1 and P2 and P3 = 1)
then $C_{o3} = C_0$, else “kill” or “generate”.

进位旁路加法器

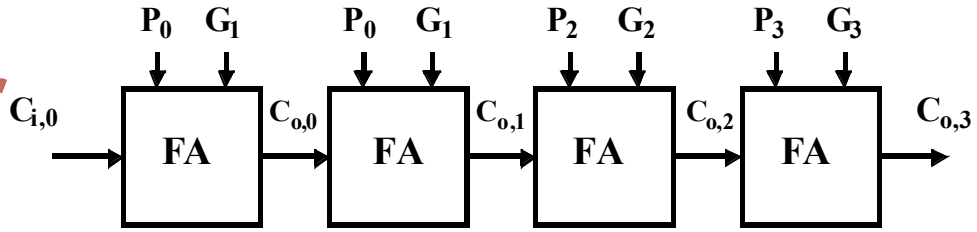


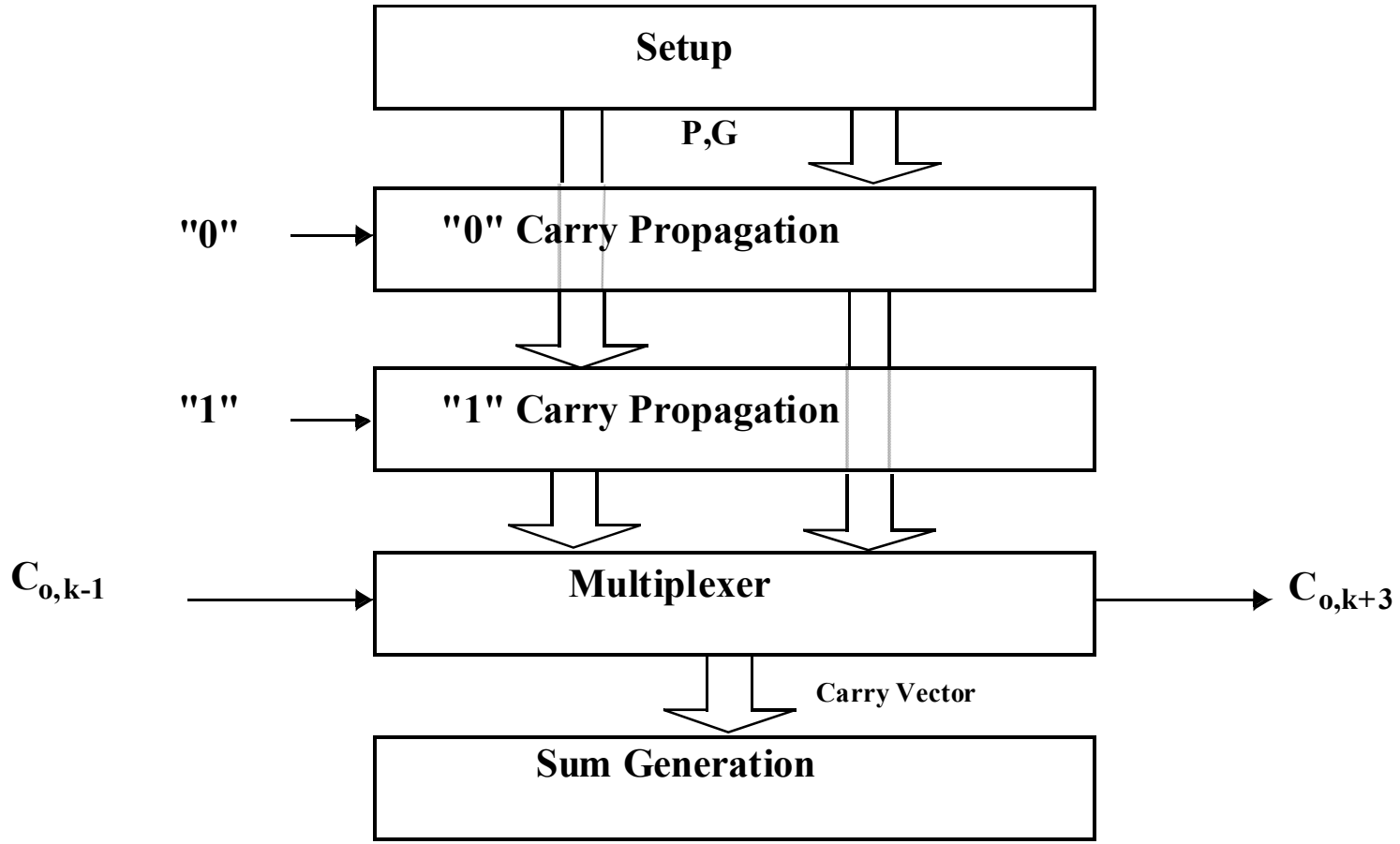
$$t_{adder} = t_{setup} + Mt_{carry} + (N/M\text{-}1)t_{bypass} + (M\text{-}1)t_{carry} + t_{sum}$$

Carry Ripple versus Carry Bypass



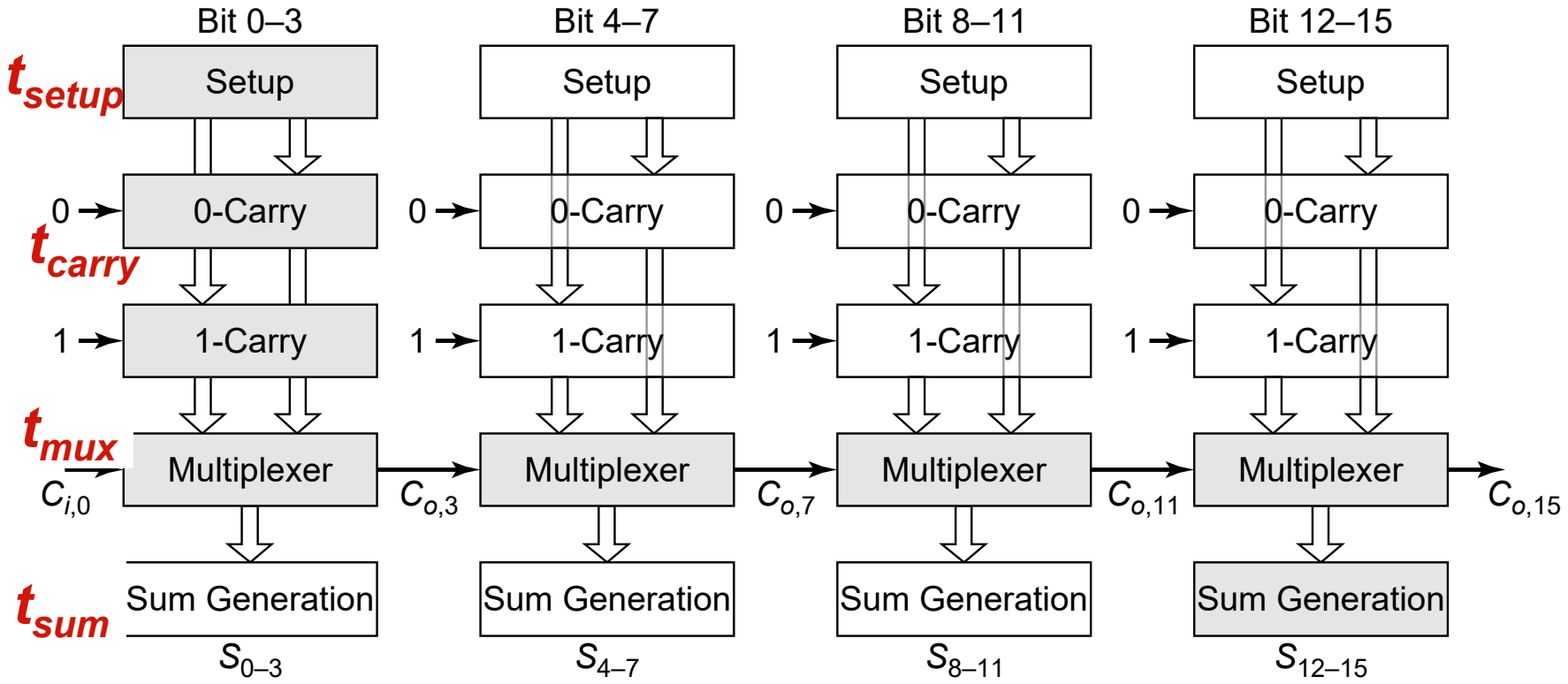
Carry-Select Adder





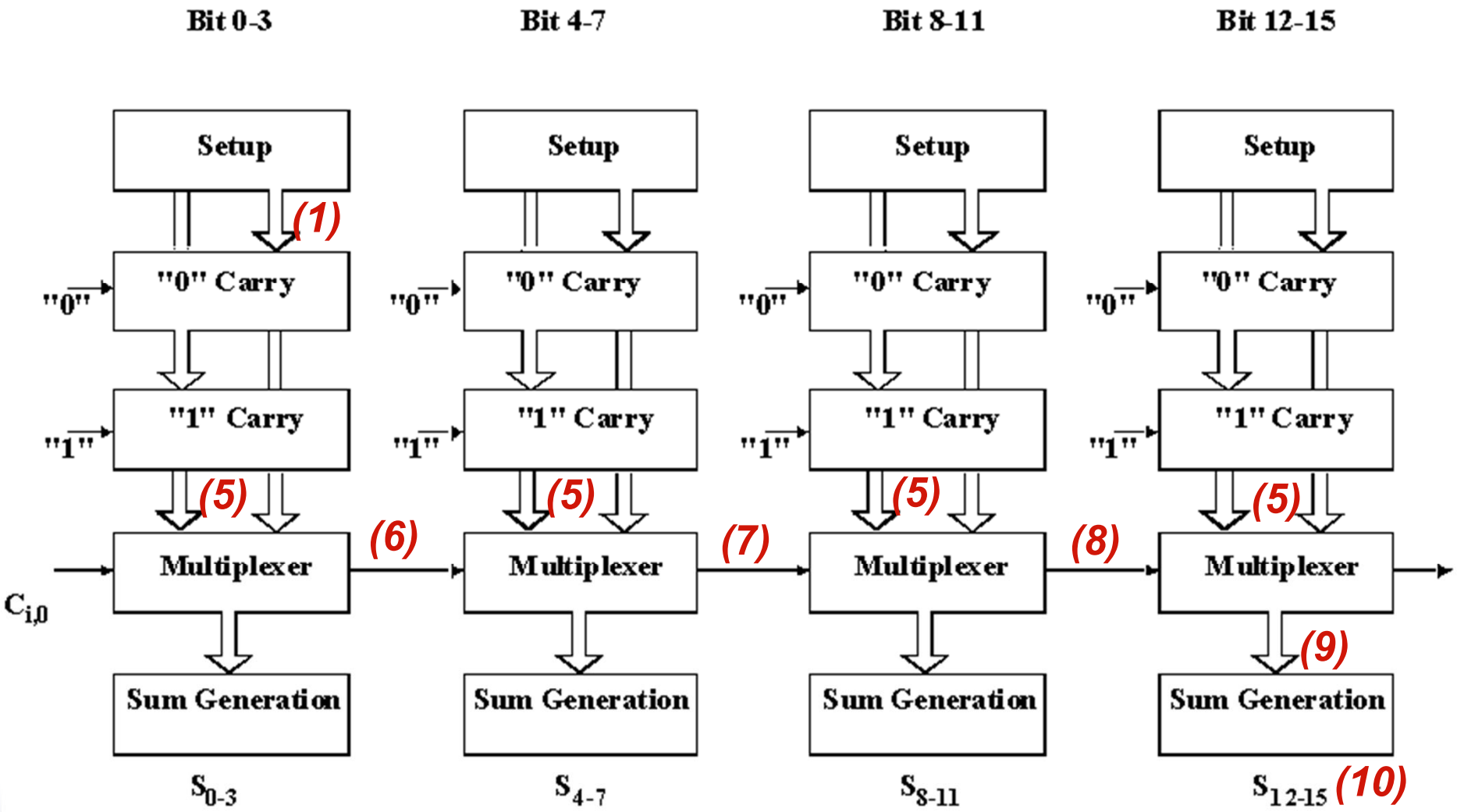
硬件开销限于一个额外的进位路径和一个多路开关，大约等于行波进位结构的30%。

Carry Select Adder: Critical Path

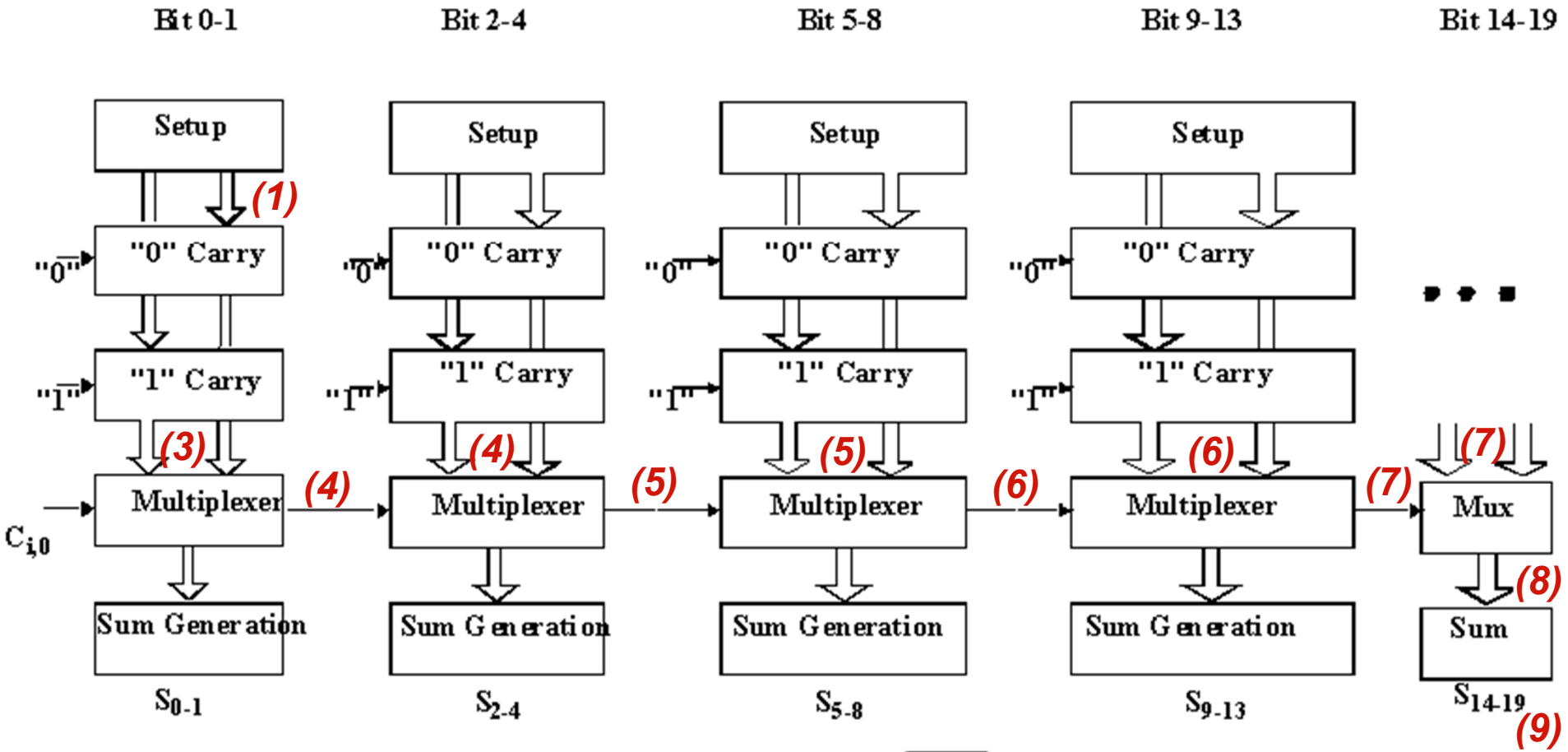


$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{mux} + M t_{carry} + t_{sum}$$

Linear Carry Select $t_{setup} = t_{carry} = t_{mux} = t_{sum} = 1$

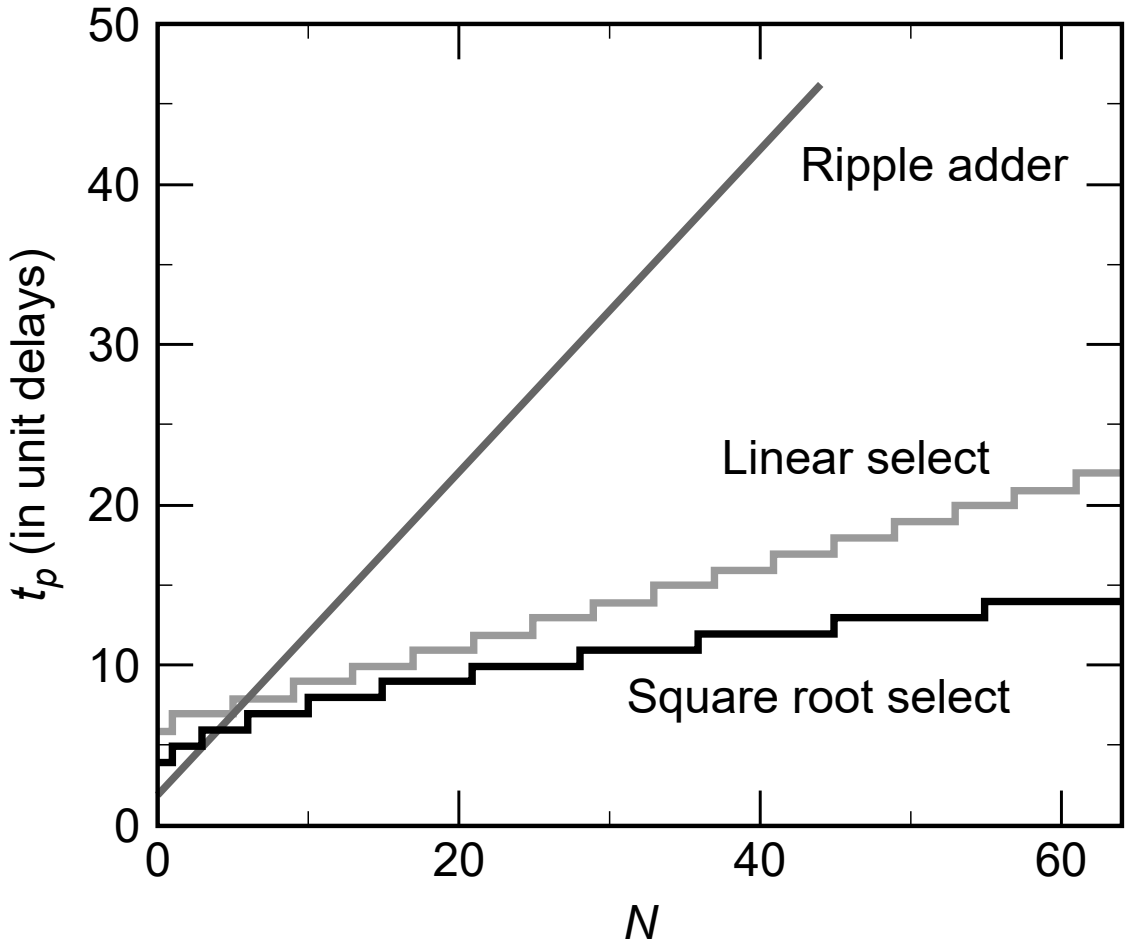


Square Root Carry Select

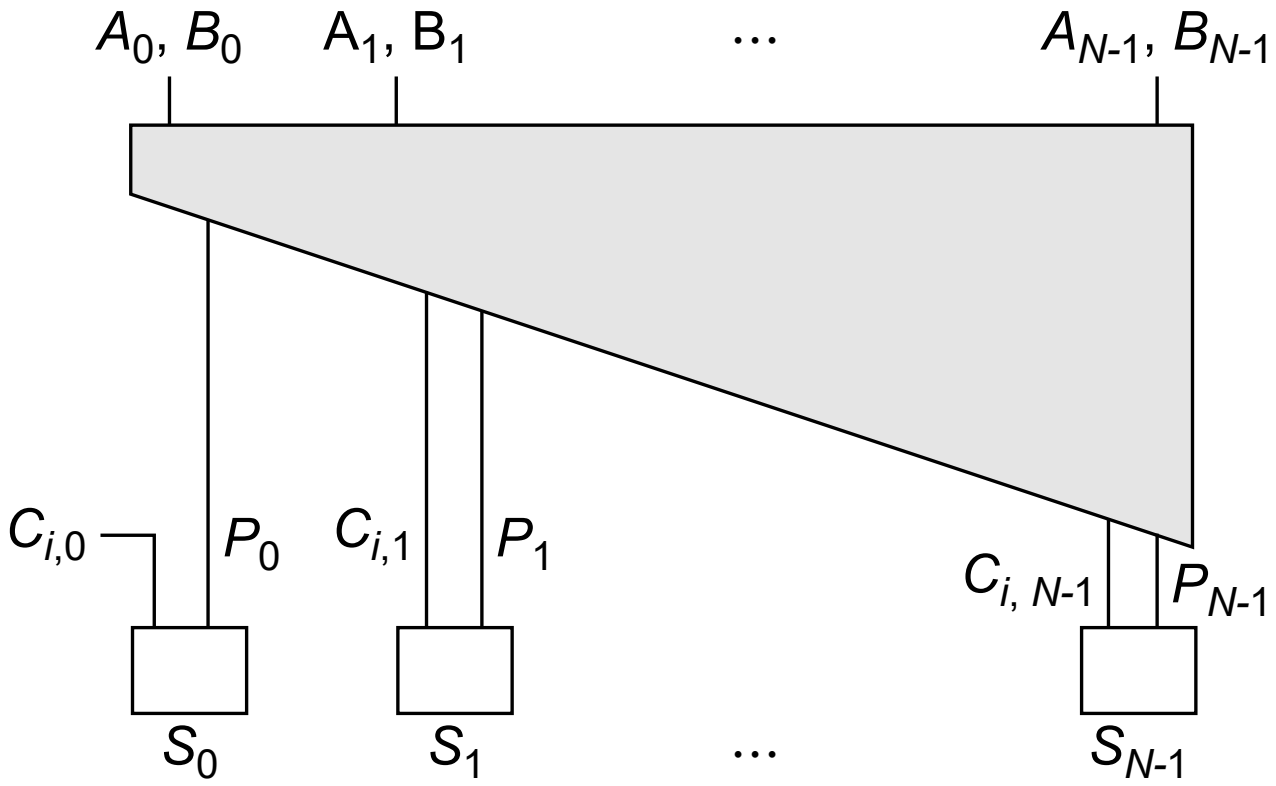


$$t_{add} = t_{setup} + M \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

加法器比较


50
40
30
20
10
0
0
20
40
60
t_p (in unit delays)
N
Ripple adder
Linear select
Square root select

超前进位加法器– 基本原理



$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1}$$

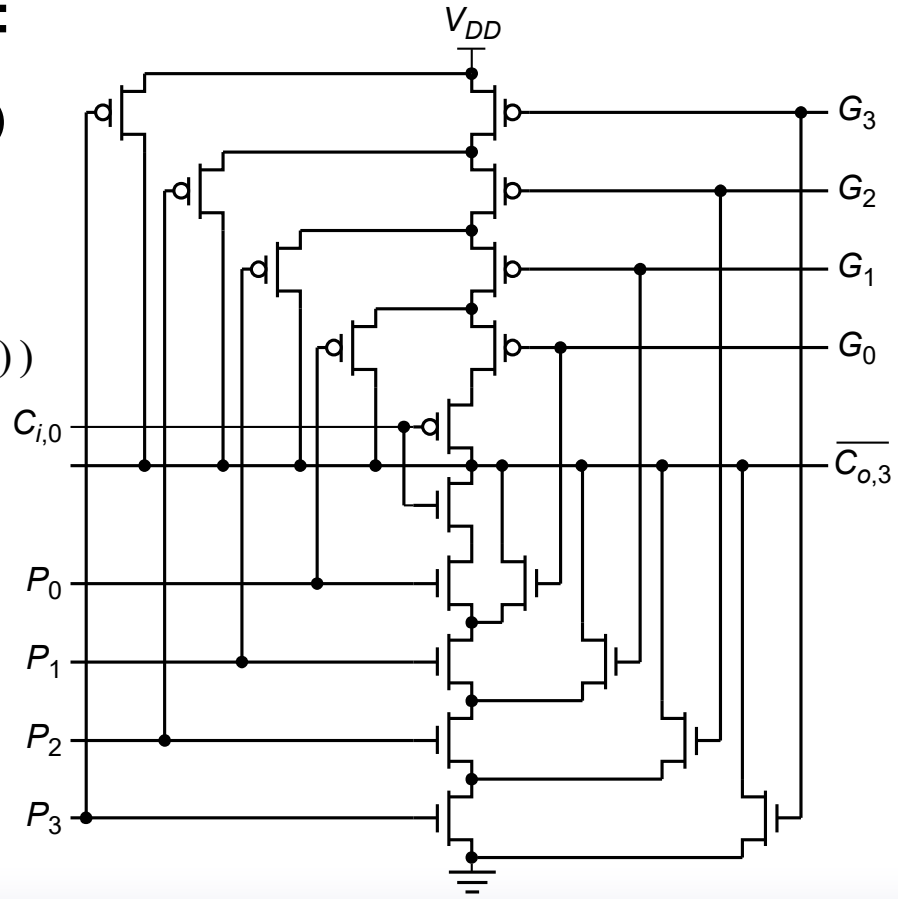
超前进位加法器: 结构

Expanding Lookahead equations:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

All the way:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0})))$$



Carry Look Ahead in CMOS?

1. Area : triangular
2. Fanout of Pi signals

$C_i = G_i + P_i.C_{i-1}$

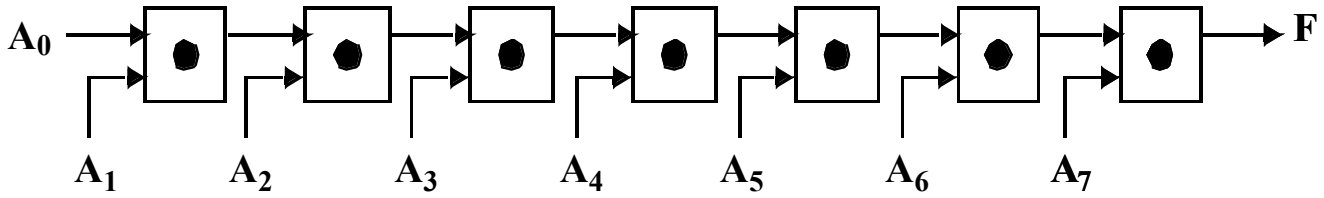
$C1 = G1 + P1.C0$	2
$C2 = G2 + P2.C1 = G2 + P2.G1 + P2.P1.C0$	3
$C3 = G3 + P3.C2 = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.C0$	4
...	...

Not a good idea...except for small groups of 4

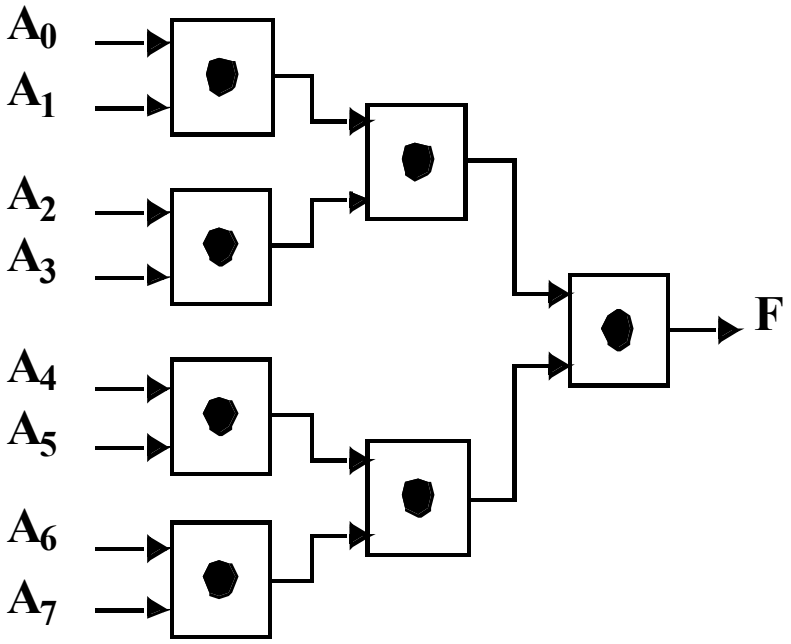
Better idea : build binary tree delay = $O(\log_2(n))$

Logarithmic adder

对数超前进位加法器



$t_p \sim N$



$t_p \sim \log_2(N)$

超前进位树

$$C_{o,0}=G_0+P_0C_{i,0}$$

$$C_{o,1}=G_1+P_1G_0+P_1P_0C_{i,0}=(G_1+P_1G_0)+(P_1P_2)C_{i,0}=G_{1:0}+P_{1:0}C_{i,0}$$

$$=G_{1:1}+P_{1:1}C_{o,0}$$

$$C_{o,2}=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_{i,0}=G_2+P_2C_{o,1}$$

$$=(G_2+P_2G_1)+P_2P_1(G_0+P_0C_{i,0})=G_{2:1}+P_{2:1}C_{o,0}$$

$$C_{o,3}=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_{i,0}$$

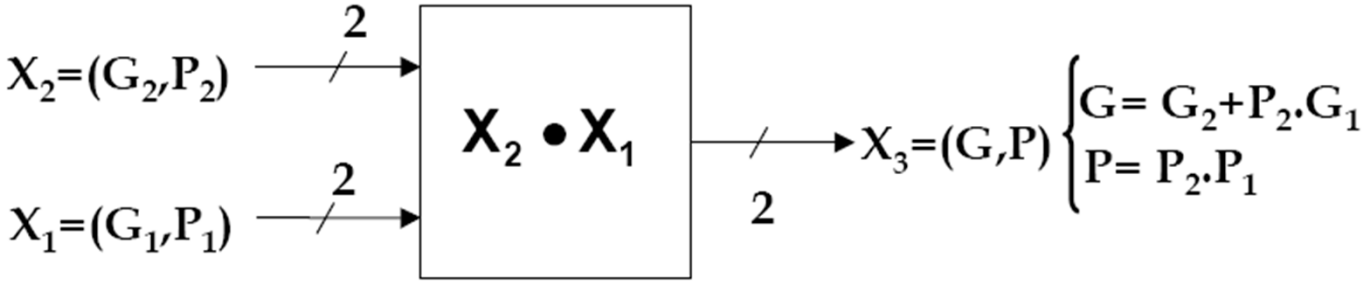
$$=(G_3+P_3G_2)+(P_3P_2)C_{o,1}=G_{3:2}+P_{3:2}C_{o,1}$$

$G_{i:j}$和$P_{i:j}$分别表示一组位（从第i位至第j位）的进位产生和进位传播函数，称为块进位产生和块进位传播信号。如果该组产生一个进位，则$G_{i:j}$等于1，而与输入进位无关。如果一个输入进位传播通过整个这一组，则$P_{i:j}$即为1。

例如，当进位产生于第3位或者当进位产生于第2位并传播通过第3位时，则$G_{3:2}$等于1（即$G_{3:2}=G_3+P_3G_2$）。

当输入进位传播通过这两位时，$P_{3:2}$为1（即$P_{3:2}=P_3P_2$）

点操作



Non-Commutative, Associative operator

$$X_2.X_1 \neq X_1.X_2$$

$$(X_3.X_2).X_1=X_3.(X_2.X_1)$$

新的表达式的形式与原来一样，只是进位产生和进位传播信号由块进位产生和块进位传播信号所代替。$G_{i:j}$和$P_{i:j}$一般化了原来的进位公式，$G_i= G_{i:i}$ ， $P_i= P_{i:i}$

$$(G_{3:2},P_{3:2})=(G_3,P_3)\cdot(G_2,P_2)$$

P426，例11.4

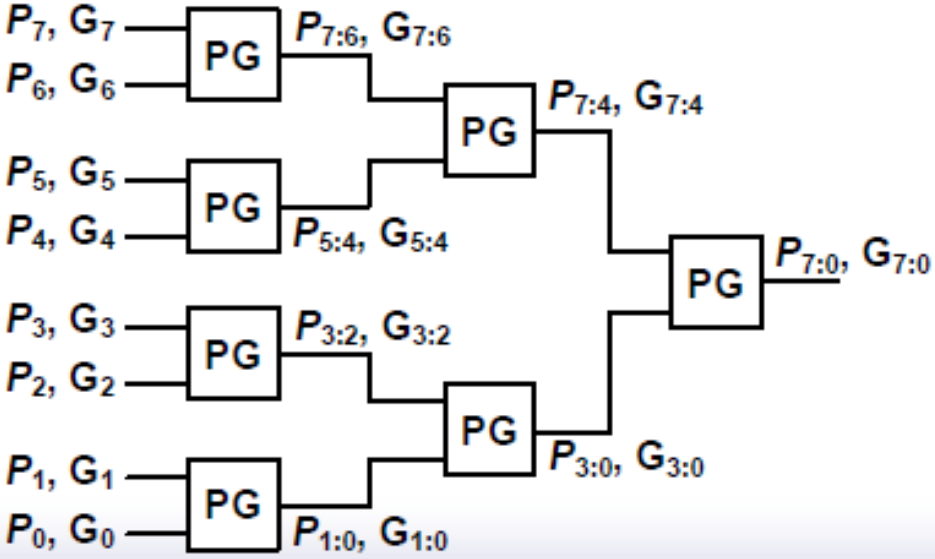
进位与点操作

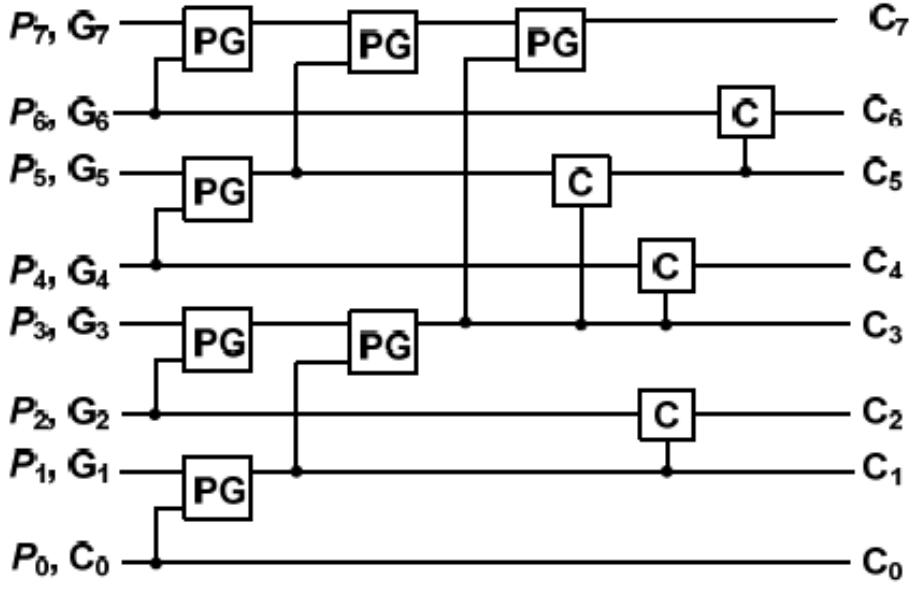
$$C_{o,0} = G_0 + P_0 C_{i,0} = (G_0, P_0) \bullet (C_{i,0}, 0)$$

$$C_{o,1} = (G_1 + P_1 G_0) + (P_1 P_0) C_{i,0} = (G_1, P_1) \bullet (G_0, P_0) \bullet (C_{i,0}, 0) = G_{1:0} + P_{1:0} C_{i,0}$$

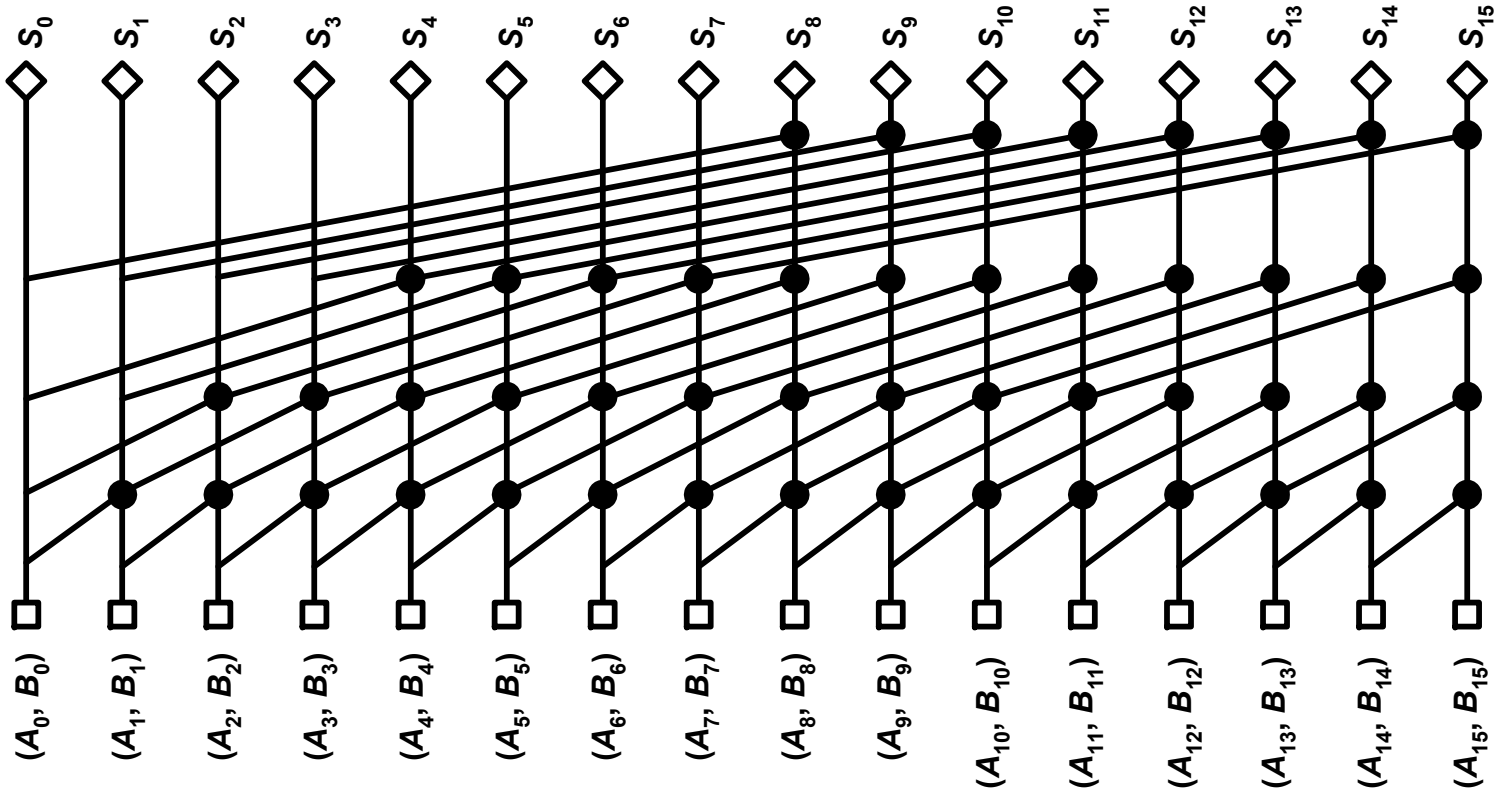
$$C_{o,2} = (G_2, P_2) \bullet (G_1, P_1) \bullet (G_0, P_0) \bullet (C_{i,0}, 0)$$

$$C_{o,3} = \ldots$$





Tree Adders



16-bit radix-2 Kogge-Stone tree

$$(G_{1:0}, P_{1:0}) = (G_1, P_1) \cdot (G_0, P_0)$$

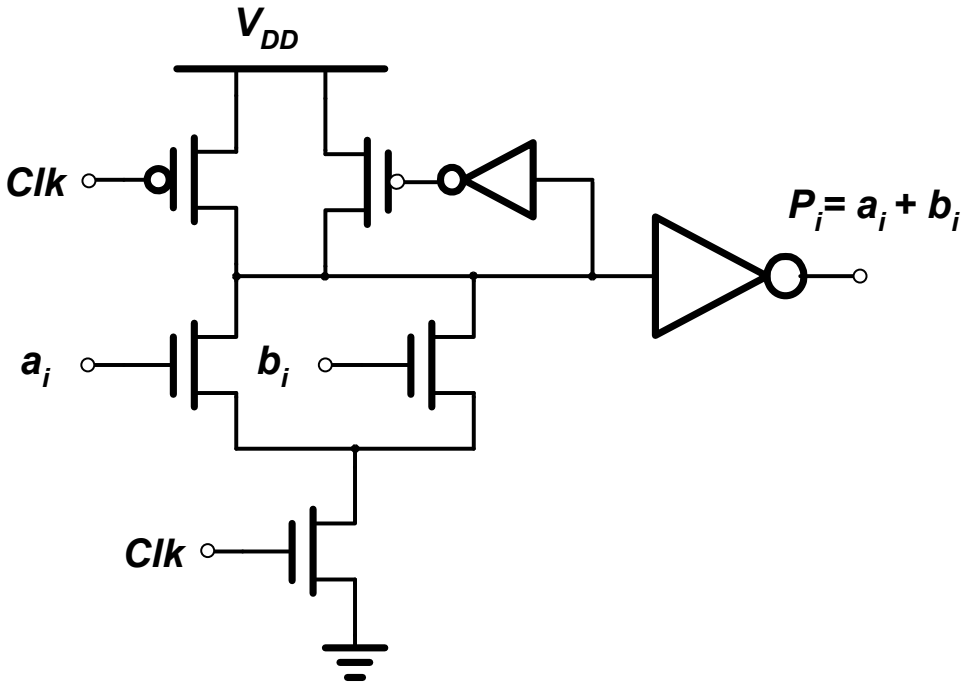
$$(G_{3:0}, P_{3:0}) = (G_{3:2}, P_{3:2}) \cdot (G_{1:0}, P_{1:0})$$

$$(G_{7:0}, P_{7:0}) = (G_{7:4}, P_{7:4}) \cdot (G_{3:0}, P_{3:0})$$

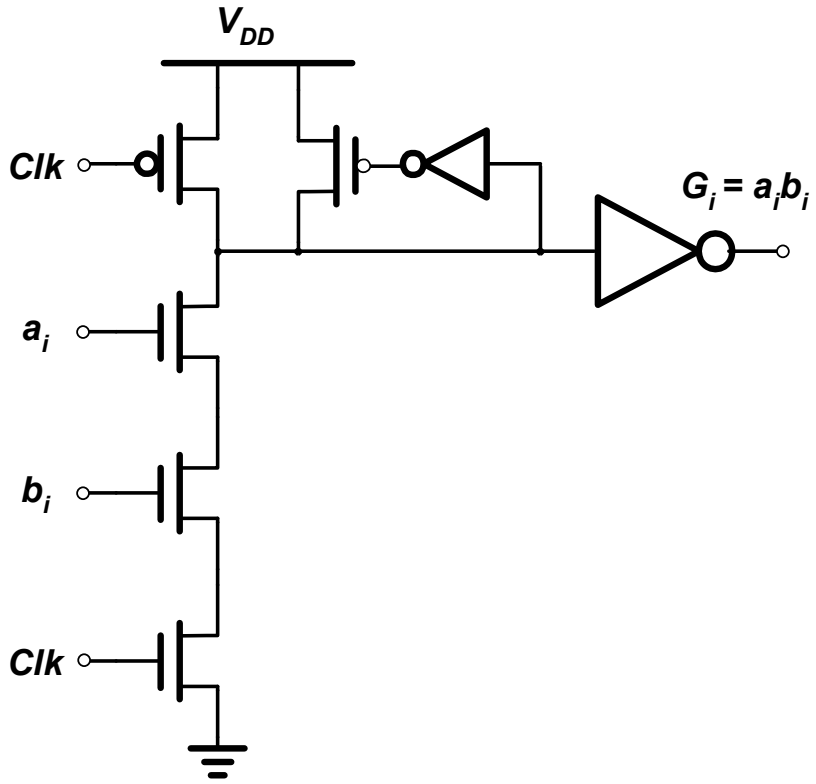
$$(G_{15:0}, P_{15:0}) = (G_{15:8}, P_{15:8}) \cdot (G_{7:0}, P_{7:0})$$

动态电路实现超前进位加法器

Domino Adder



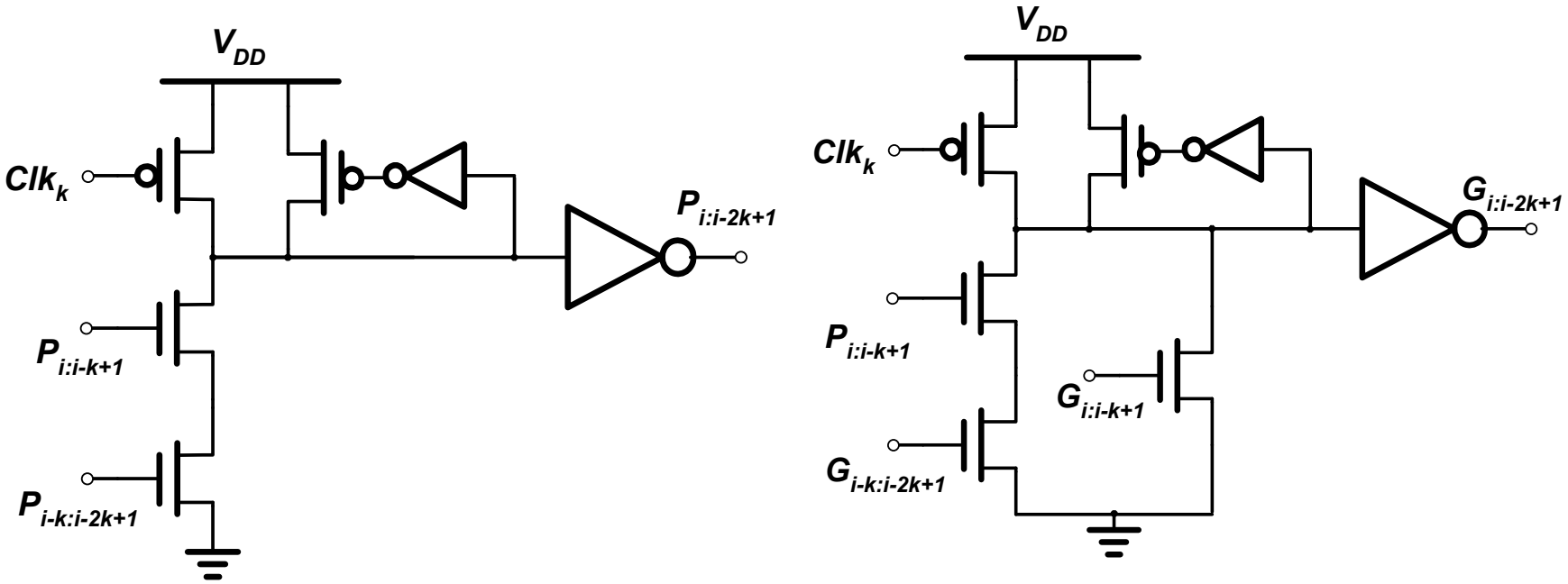
Propagate



Generate

动态电路实现超前进位加法器

Domino Adder

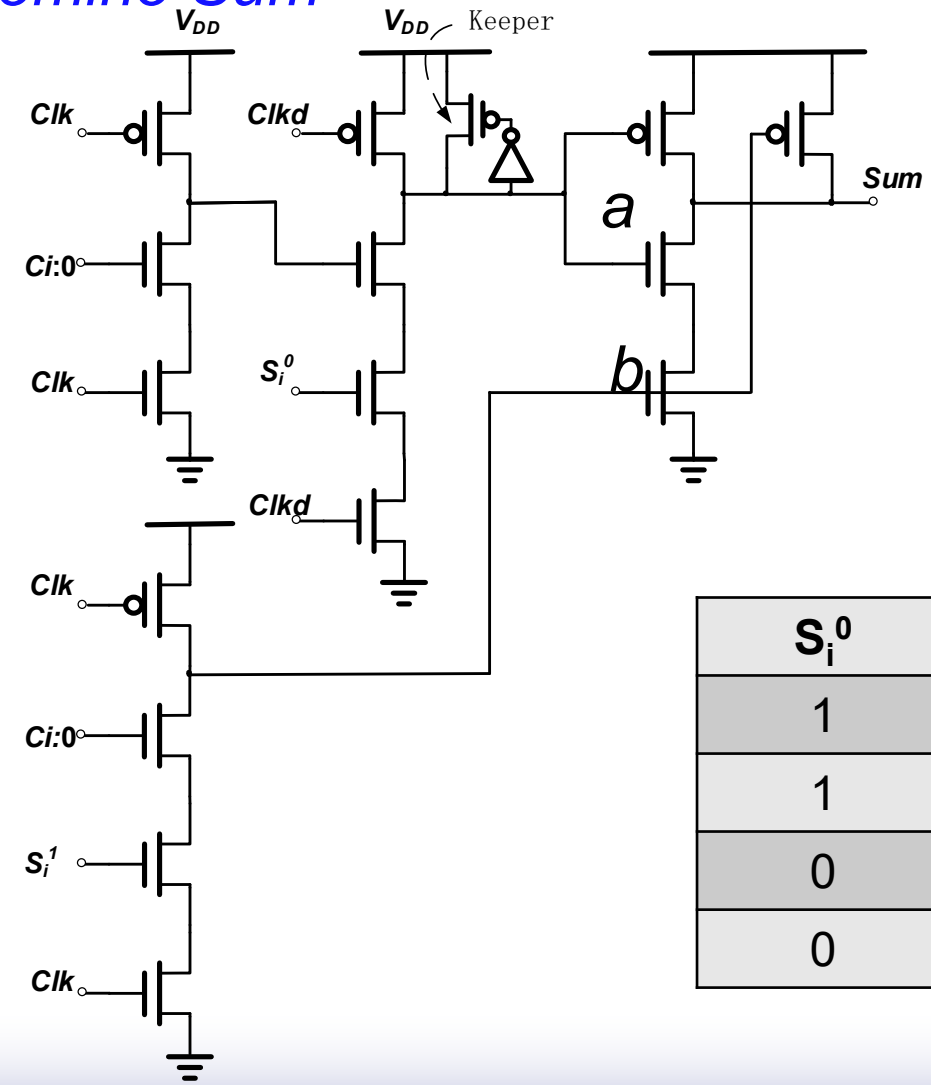


$$(G_{i:i-2k+1}, P_{i:i-2k+1}) = (G_{i:i-k+1}, P_{i:i-k+1}) \cdot (G_{i-k:i-2k+1}, P_{i-k:i-2k+1})$$

$$= (G_{i:i-k+1} + P_{i:i-k+1} G_{i-k:i-2k+1},\ P_{i:i-k+1} P_{i-k:i-2k+1})$$

动态电路实现超前进位加法器

Domino Sum



采用多米诺电路通过求和选择电路实现求和。

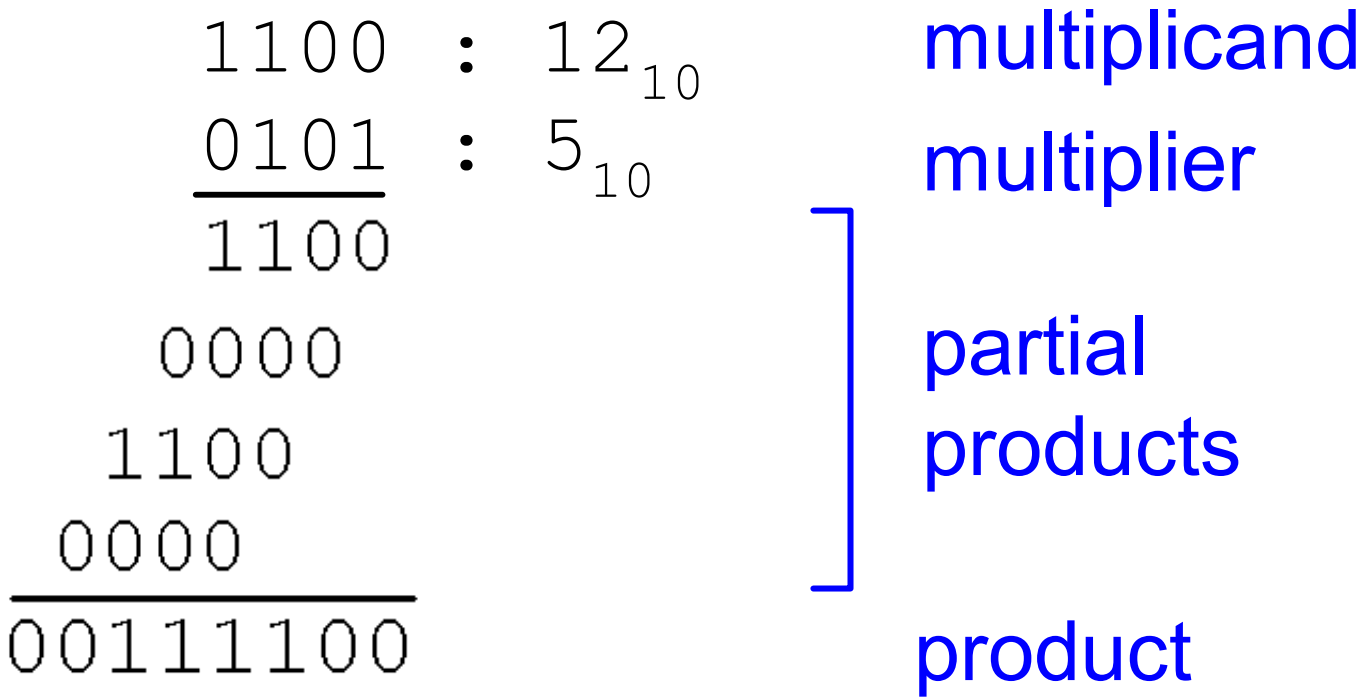
$$s_i^0 = \overline{a \oplus b}$$

$$s_i^1 = a \oplus b$$

S_i^0	S_i^1	C_i^0	~S
1	0	0	0
1	0	1	1
0	1	0	1
0	1	1	0

乘法器的基本原理

乘法的示例


1100 : 12_{10}
0101 : 5_{10}
1100
0000
1100
0000
00111100
multiplicand
multiplier
partial products
product

乘法的基本形式

- M x N位的乘法
  - 生成N个M位的部分积
  - 将这N个部分积

  求和生成M+N位最终积

$$X = \sum_{i=0}^{M-1} X_i 2^i \qquad Y = \sum_{j=0}^{N-1} Y_j 2^j$$

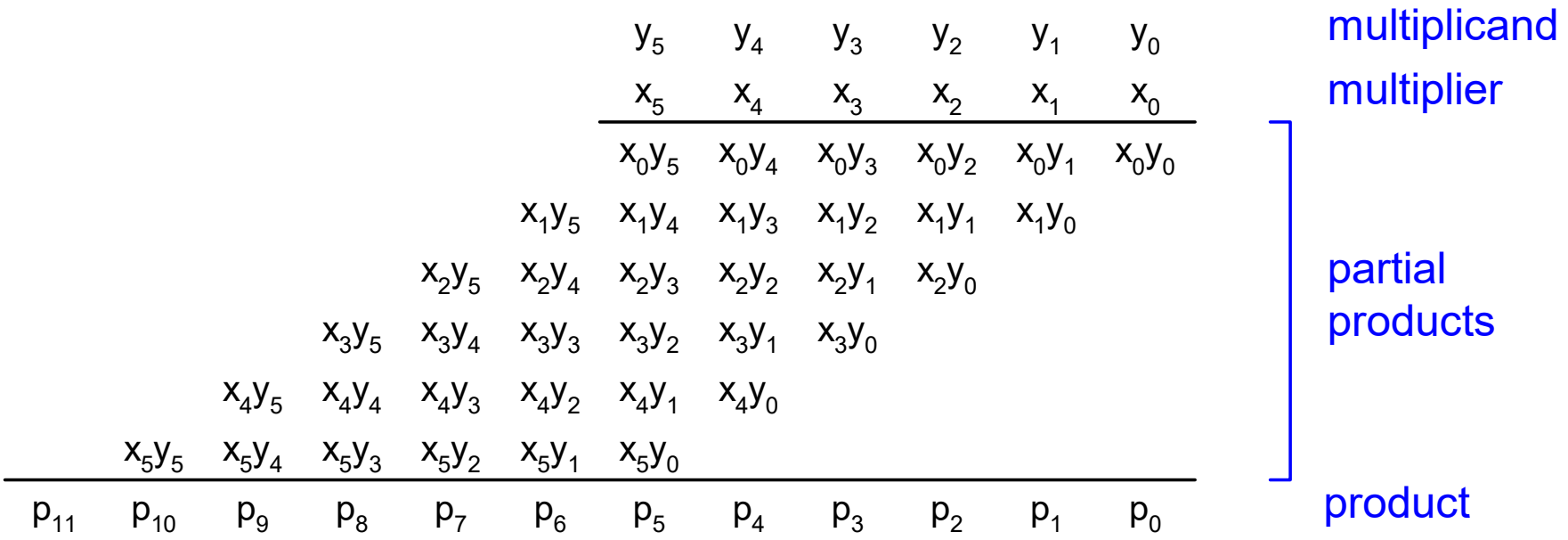
$$Z = X \times Y = \sum_{k=0}^{M+N-1} Z_k 2^k$$

$$= \left(\sum_{i=0}^{M-1} X_i 2^i \right) \left(\sum_{j=0}^{N-1} Y_j 2^j \right)$$

$$= \sum_{i=0}^{M-1} \left(\sum_{j=0}^{N-1} X_i Y_j 2^{i+j} \right)$$

乘法的基本形式

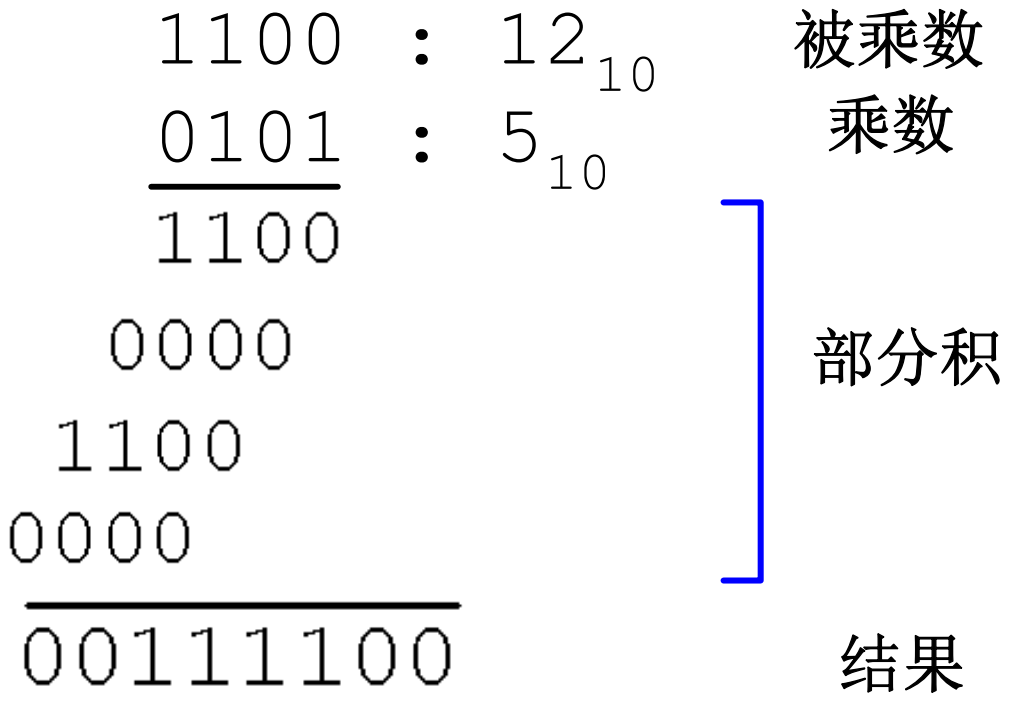
- 被乘数: $Y = (y_{M-1}, y_{M-2}, \cdots, y_1, y_0)$
- 乘数: $X = (x_{N-1}, x_{N-2}, \cdots, x_1, x_0)$
- 积:



部分积的产生——波兹编码

- 陈列乘法器需要N个部分积
- 如果将乘数的每r位配对成组一次乘以被乘数,则部分积会减少至N/r个
  - 乘法器会不会更快,规模会不会更少?
  - 被称为基-2^r 编码
- 假设r=2则有
  - 形成部分积 0, Y, 2Y, 3Y
  - 前面三个比较容易实现, 但是3Y 需要加法器。

波兹编码



01111110

$10000\bar{1}0$

波兹(Booth)编码：保证在每两个连续位中最多只有一个**1**或**-1**，使部分积的数目至少可以减少到原来的一半。

波兹编码

从形式上说，这一变换相当于把乘数字变换为一个四进制形式，而不是通常的二进制形式：

$$Y=\sum_{j=0}^{(N-1)/2} Y_j 4^j \quad (Y_j \in \{-2,-1,0,1,2\})$$

虽然{0,1}相乘相当于一个AND操作，但乘以{-2,-1,0,1,2}还需要结合反相逻辑和以为逻辑。

大小不同的不同的阵列对乘法器设计并不合理，因此经常使用的是改进波兹编码。

改进波兹编码

- 乘数按三位一组进行划分，相互重叠一位。每一组的三位按下表编码，并形成部分积。
- 进入编码的输入位是两个当前位和一个来自相邻低位组的最高位。

Table 10.12 Radix-4 modified Booth encoding values

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	X_i	$2X_i$	M_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

数学证明

$$\mathrm{A} = -A_{N-1}\times 2^{N-1} + A_{N-2}\times 2^{N-2} + \cdots A_1\times 2^1 + A_0\times 2^0$$

$$= (-A_{N-1}+A_{N-2})\times 2^{N-1} + (-A_{N-2}+A_{N-3})\times 2^{N-2} + \cdots + (-A_2+A_1)\times 2^2$$

$$+ (-A_1+A_0)\times 2^1 + (-A_0+0)\times 2$$

$$= (-2A_{N-1}+A_{N-2}+A_{N-3})\times 2^{N-2} + (-2A_{N-3}+A_{N-4}+A_{N-5})\times 2^{N-4} + \cdots$$

$$+ (-2A_3+A_2+A_1)\times 2^2 + (-2A_1+A_0+0)\times 2$$

$$B \times A = B \times \left(\sum_{i=0, A_{-1}=0}^{\frac{N}{2}-1} (-2A_{2i+1}+A_{2i}+A_{2i-1}) \times 2^{2i} \right)$$

$$= \sum_{i=0, A_{-1}=0}^{\frac{N}{2}-1} B \times (-2A_{2i+1}+A_{2i}+A_{2i-1}) \times 2^{2i}$$

改进波兹编码

- 从msb(最高有效位)至lsb(最低有效位)，可以把它们分为三位一组，首尾重叠的四组。
- 如：011=1+1=2，即为10；

  111=-2+1+1=0，即为00；

  110=-2+0+0=-2,即为$\bar{1}0$；

01111110

10

000

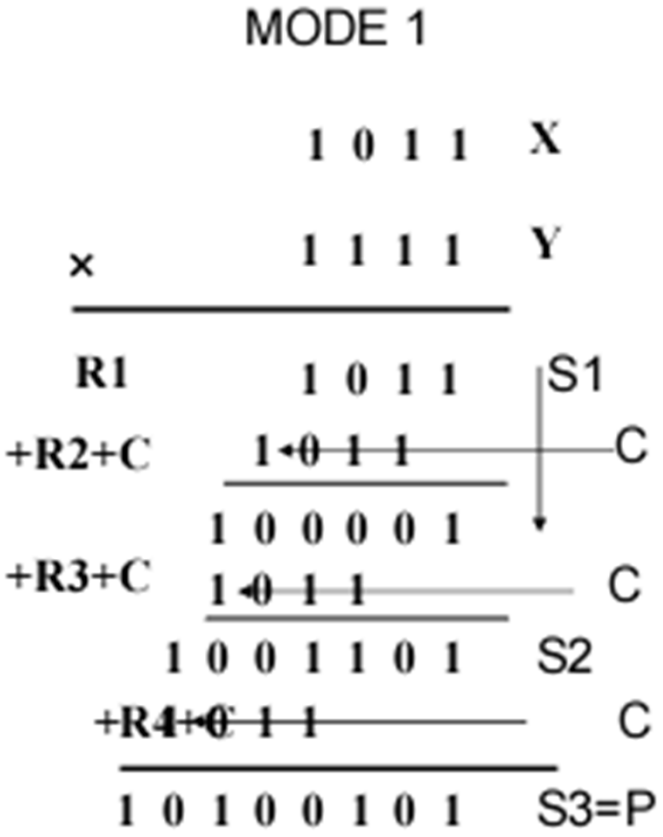
000

0$\bar{1}$0

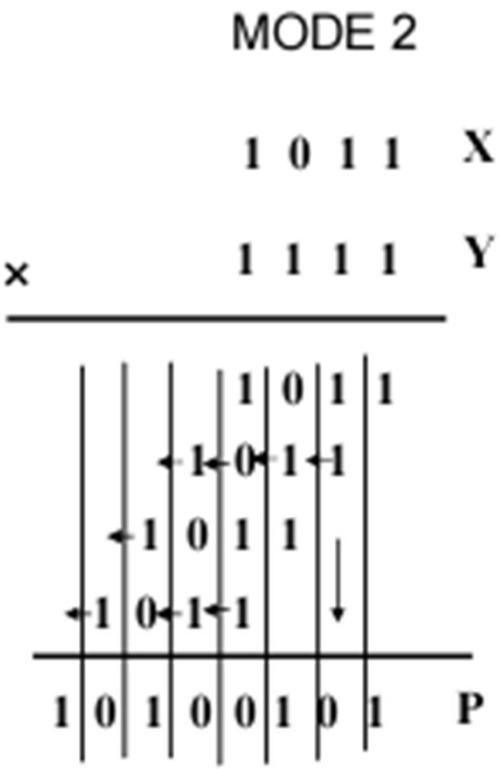
10000$\bar{1}$0

$$A=\left(\sum_{i=0,A_{-1}=0}^{\frac{N}{2}-1}(-2A_{2i+1}+A_{2i}+A_{2i-1})x2^{2i}\right)$$

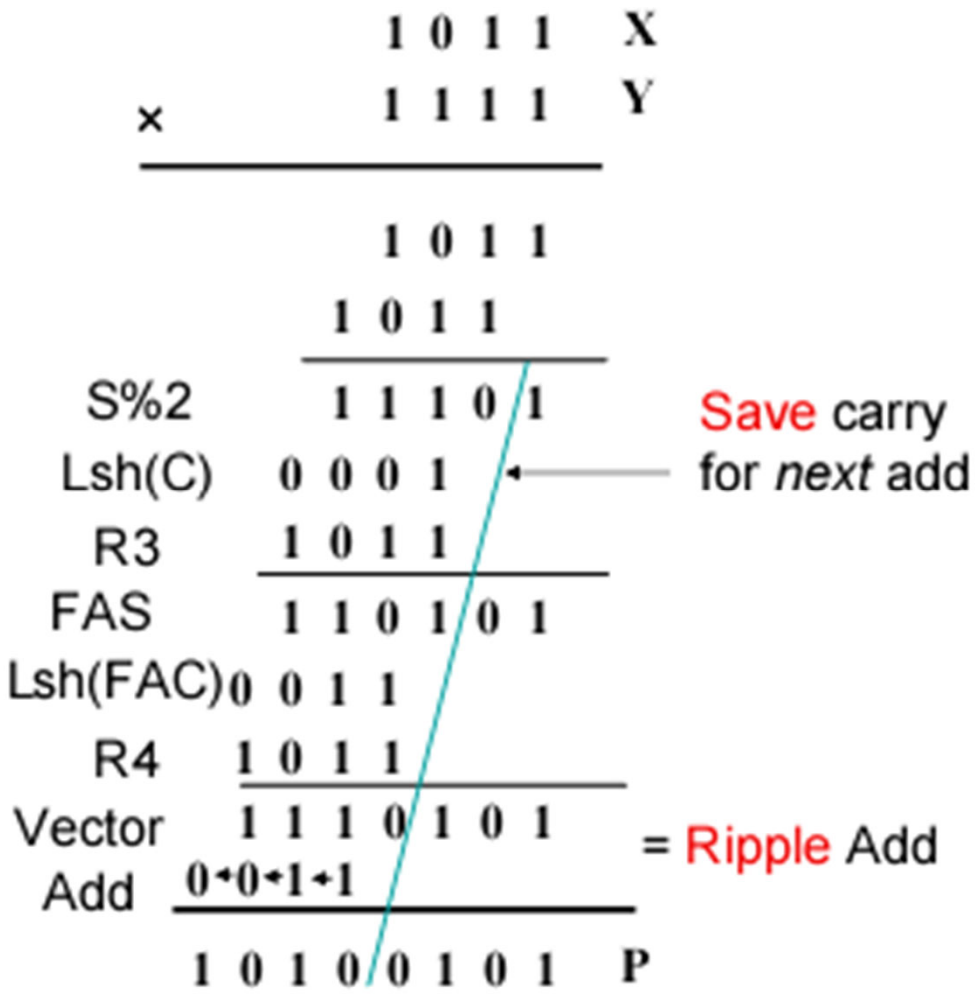
乘法的三种实现模式



Row by row carry horizontal



Column-by-Column, carry horizontal
Pen and paper

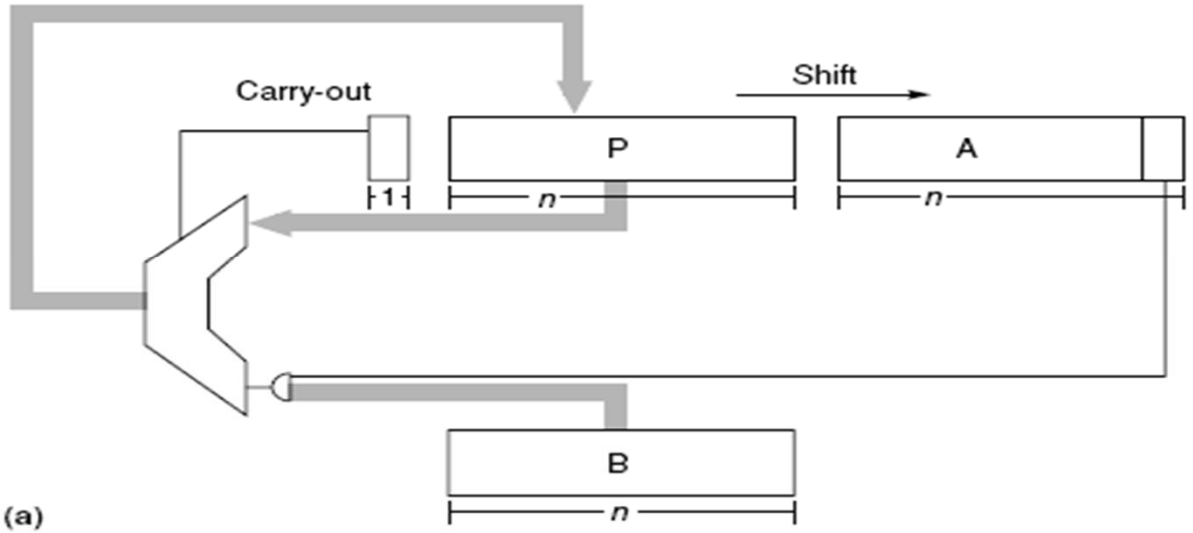

1 0 1 1 X
× 1 1 1 1 Y
1 0 1 1
1 0 1 1
S%2 1 1 1 0 1
Lsh(C) 0 0 0 1
R3 1 0 1 1
FAS 1 1 0 1 0 1
Lsh(FAC) 0 0 1 1
R4 1 0 1 1
Vector 1 1 1 0 1 0 1
Add 0 0 1 1
1 0 1 0 0 1 0 1 P
Save carry
for next add
= Ripple Add

乘法器

- 串行乘法器
- 并行乘法器

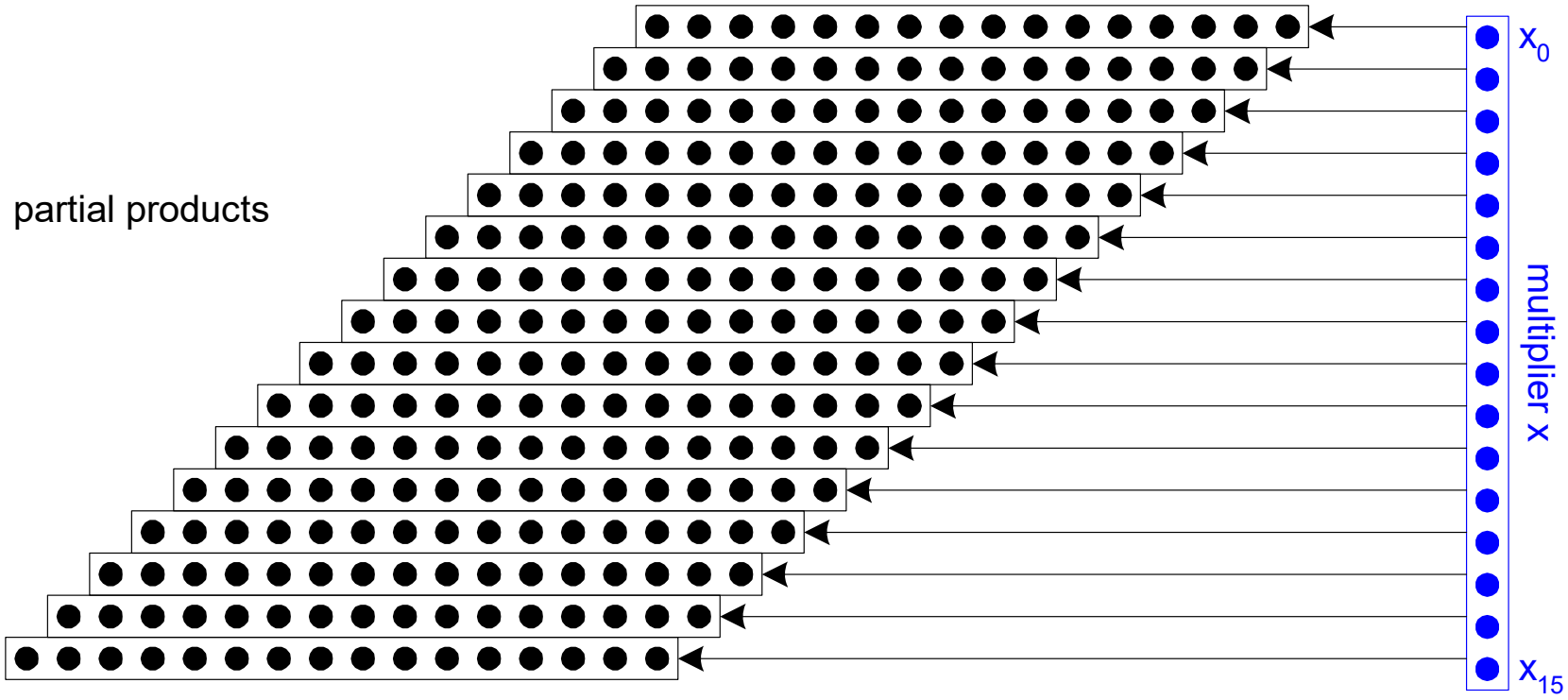
串行乘法器(移位然后相加)

- 标准加法器，移位器
- 移之后与上次产生的结果相加
- 简单但速度慢(需要N个周期)



点图

- 每个点代表一位

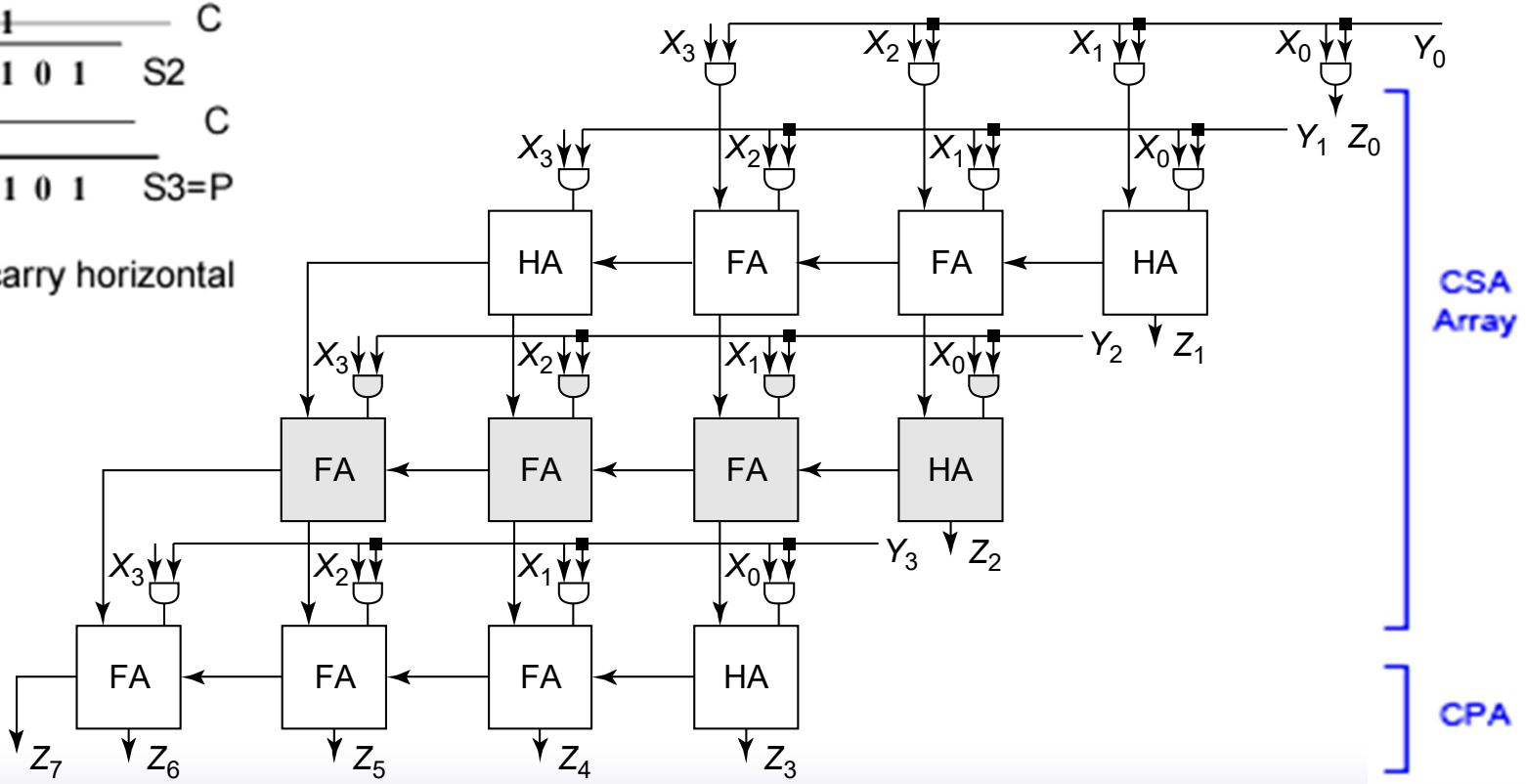


陈列乘法器

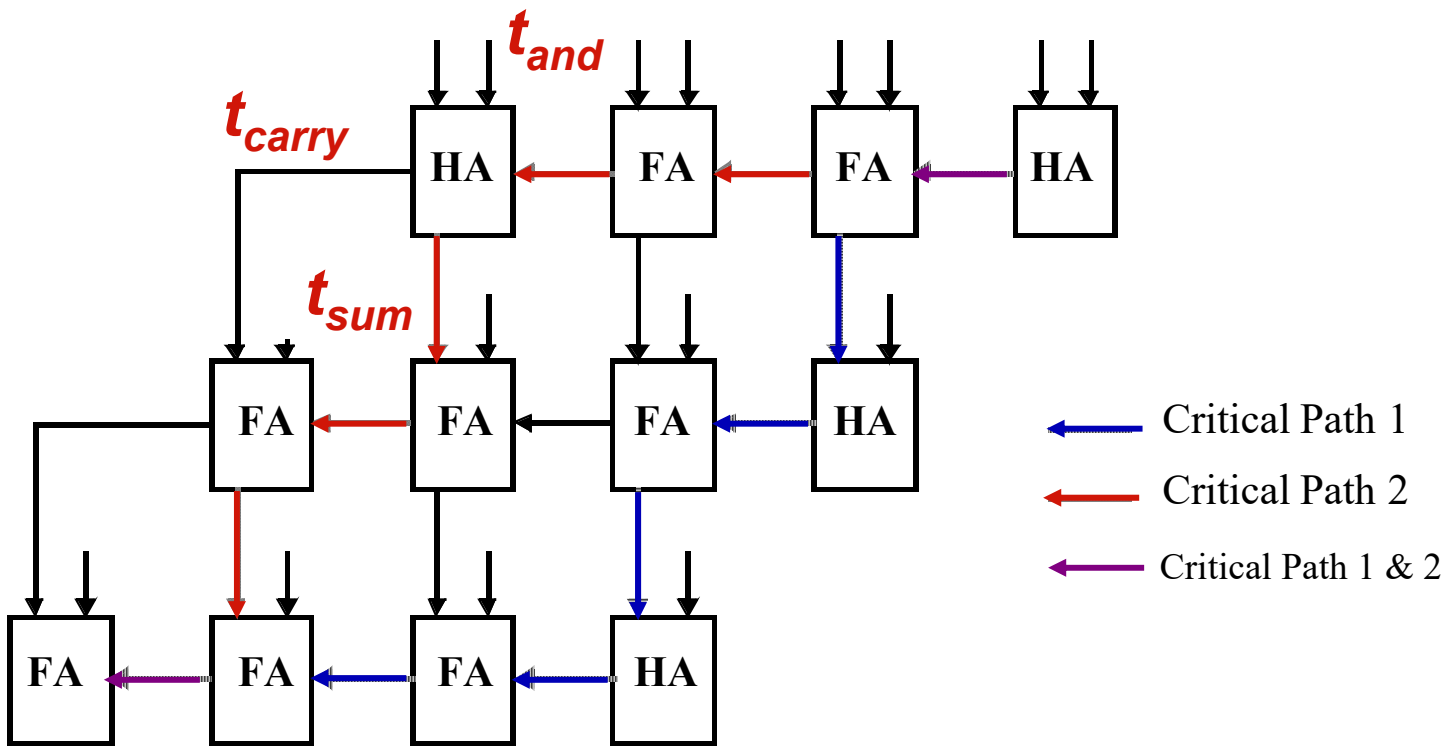
MODE 1

```
          1 0 1 1   X
      ×   1 1 1 1   Y
      ---------------
R1        1 0 1 1   S1
+R2+C   1 0 1 1     C
      ---------
    1 0 0 0 0 1
+R3+C 1 0 1 1       C
      ---------
  1 0 0 1 1 0 1     S2
+R4+C 1 0 1 1       C
      ---------
1 0 1 0 0 1 0 1     S3=P
```

Row by row carry horizontal



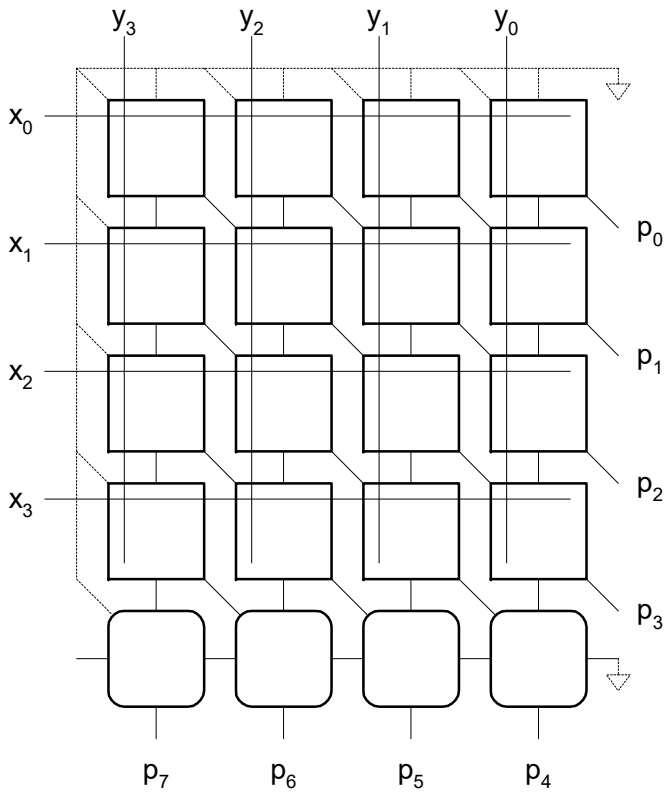
陈列乘法器的关键路径

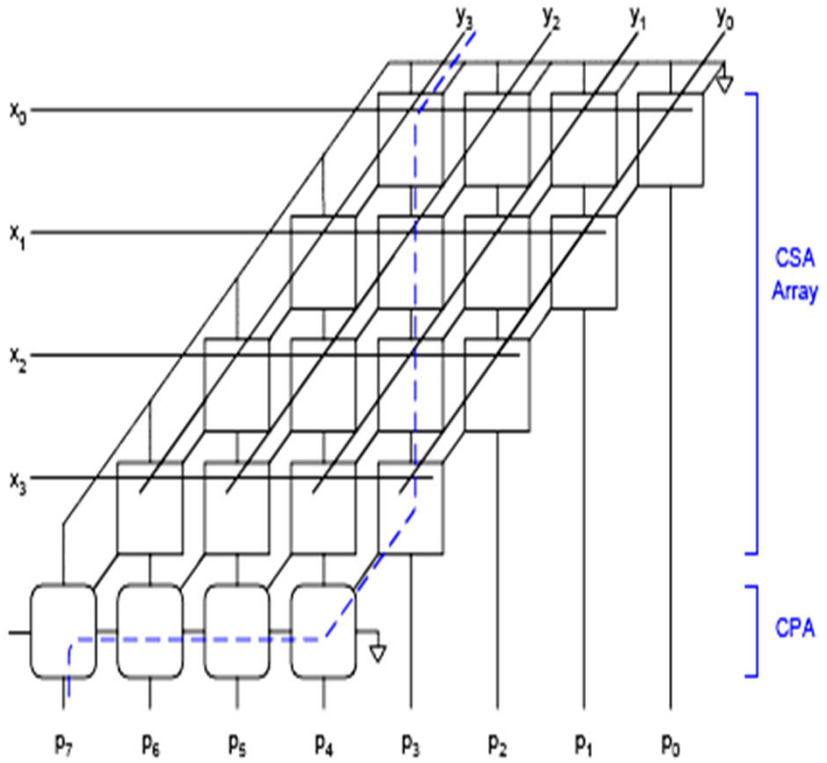


$$t_{mult} \approx [(M-1)+(N-2)]t_{carry} + (N-1)t_{sum} + t_{and}$$

矩形陈列乘法器

- 压制陈列以保证形成矩形的布图

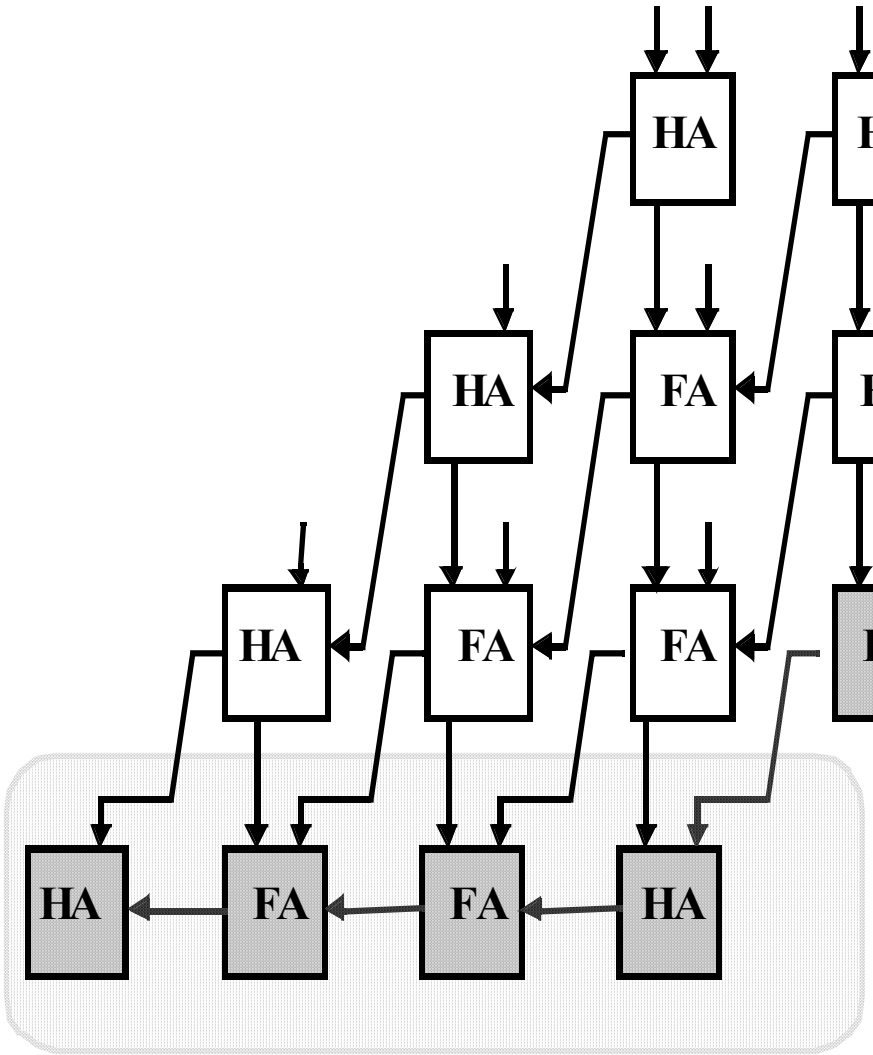




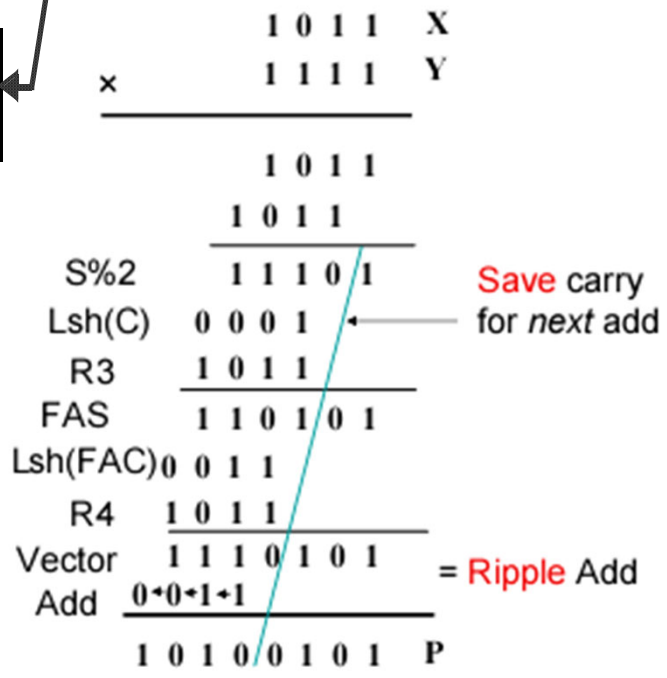
阵列乘法器的特点

- 阵列乘法器的组织结构规则性强，标准化程度高。适合用超大规模集成电路实现，且能获得很高的运算速度。
- 集成电路的价格不断下降，阵列乘法器在某些数字系统中，例如在信号及数据处理系统中受到重视，它不需要时钟脉冲，而其乘法速度仅决定于门和加法器的传输延迟。

进位保留乘法器

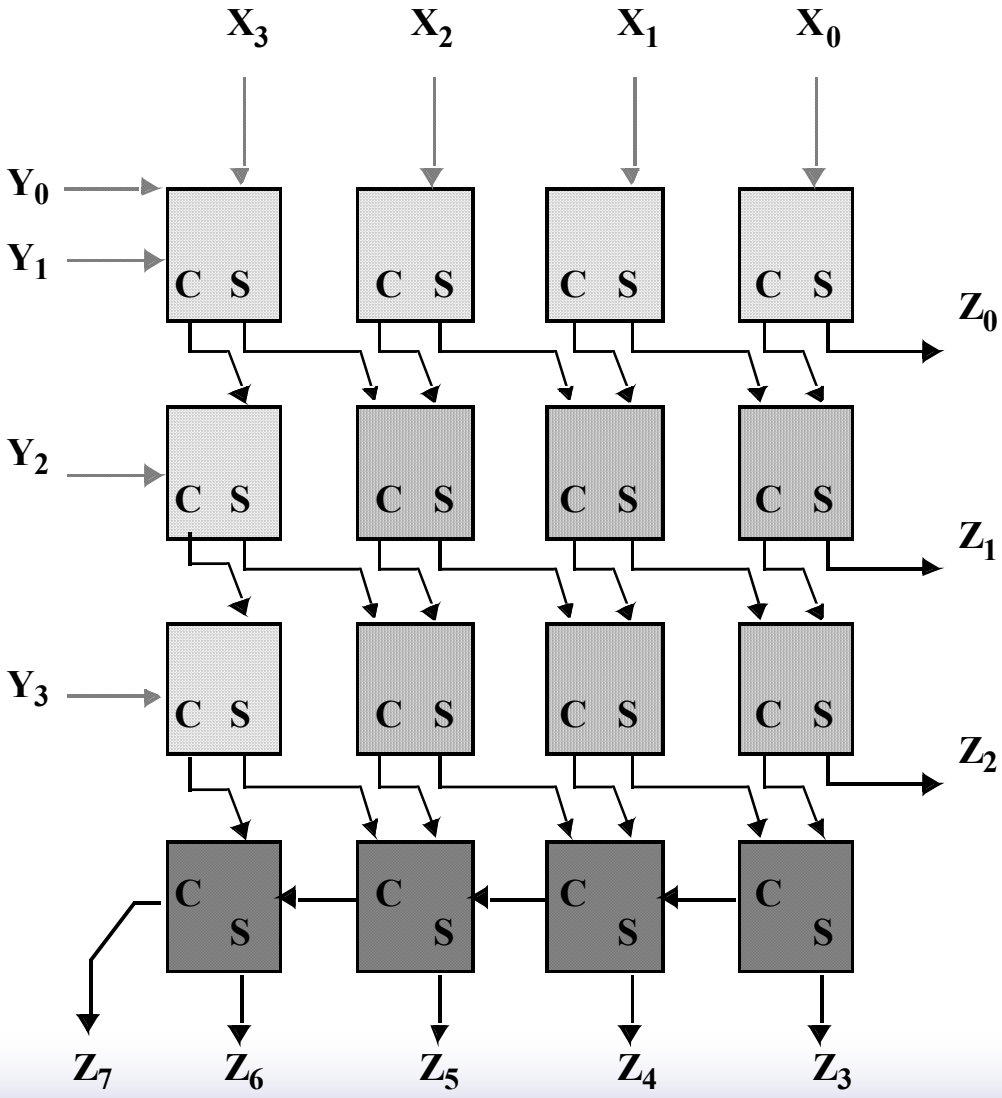


Vector Merging Adder



$$t_{mult} = (N-1)t_{carry} + t_{and} + t_{merge}$$

进位保留加法器平面布图



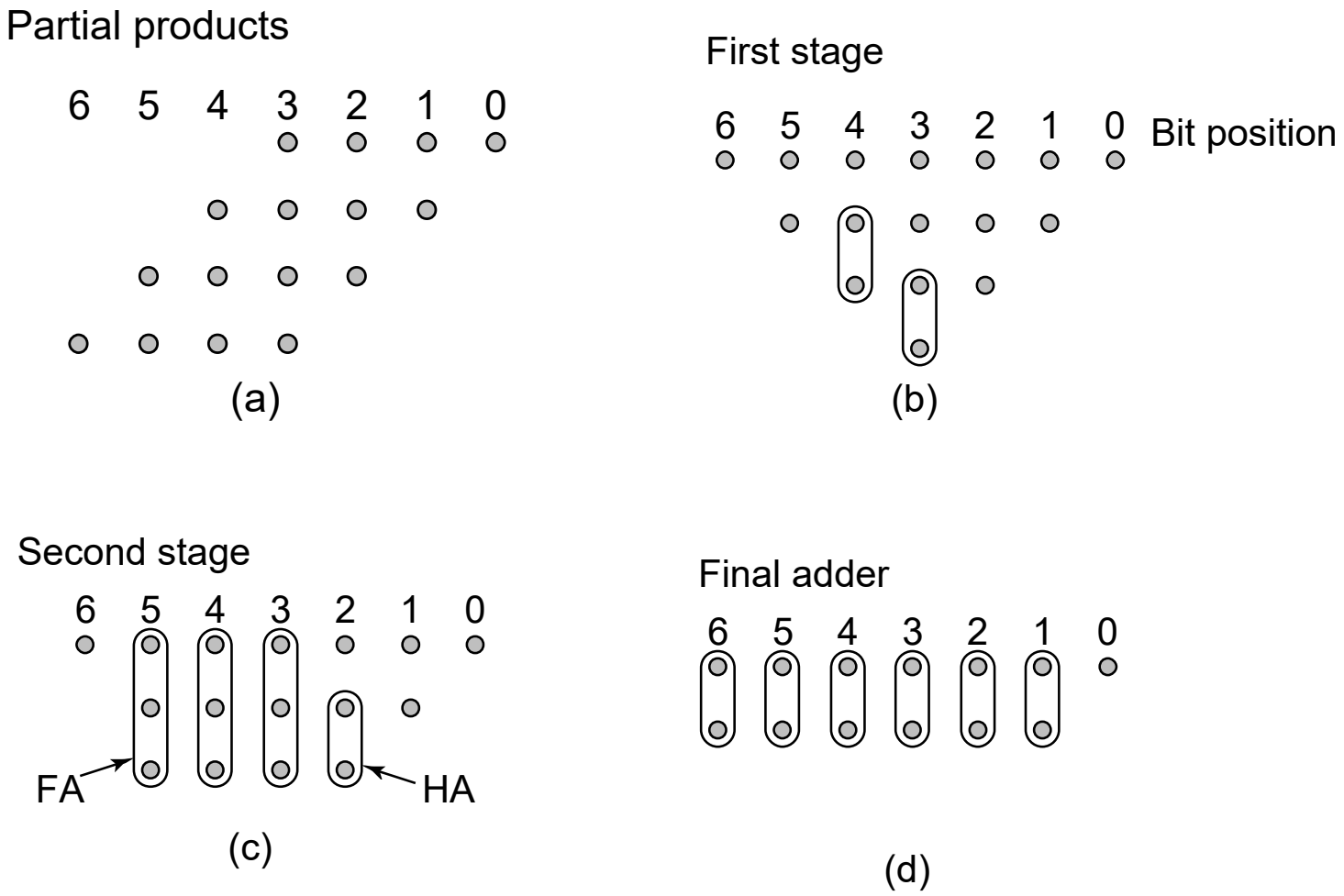
HA Multiplier Cell

FA Multiplier Cell

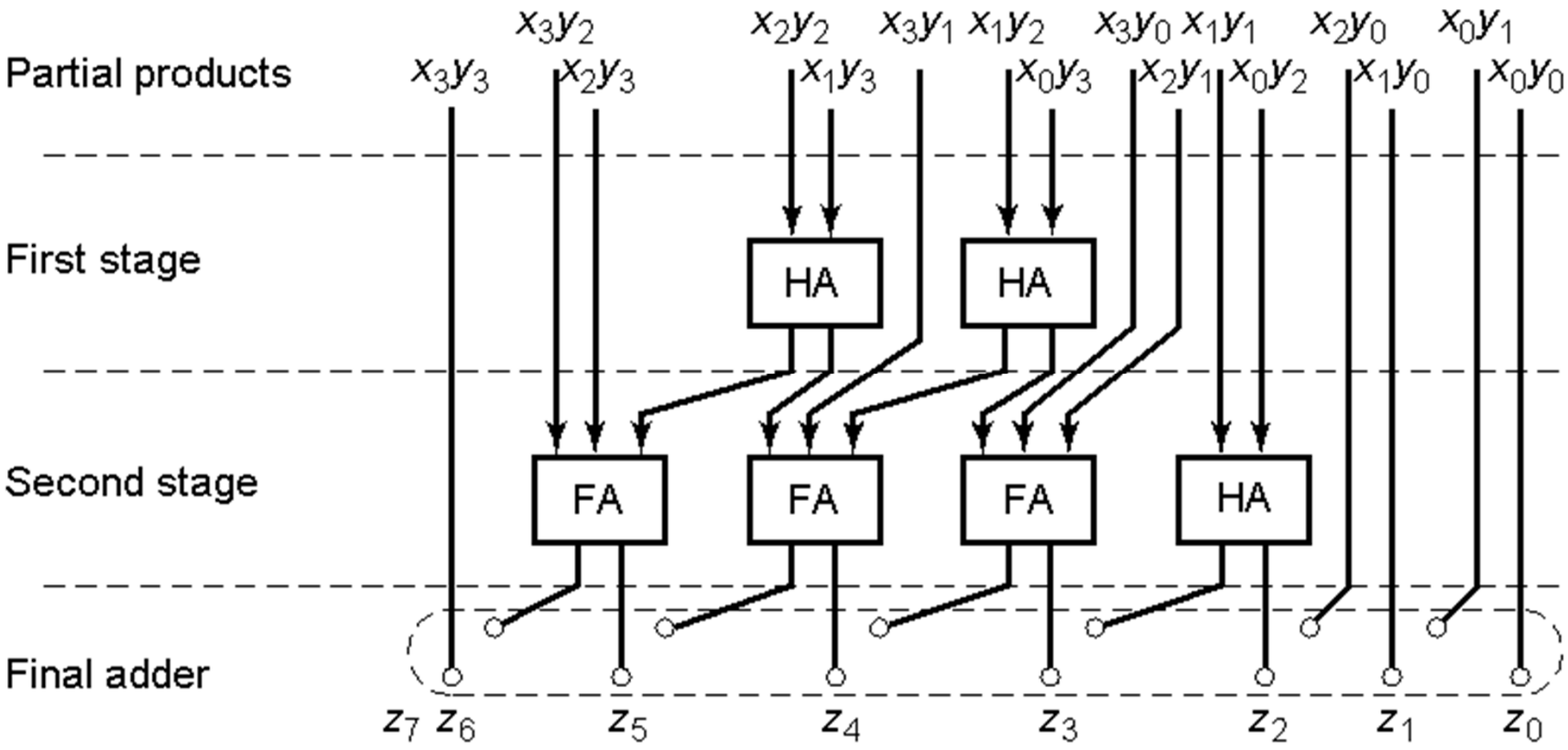
Vector Merging Cell

X and Y signals are broadcasted through the complete array.
(→)

树形乘法器


Partial products
6 5 4 3 2 1 0
(a)
First stage
6 5 4 3 2 1 0
Bit position
(b)
Second stage
6 5 4 3 2 1 0
FA
HA
(c)
Final adder
6 5 4 3 2 1 0
(d)

Wallace树乘法器



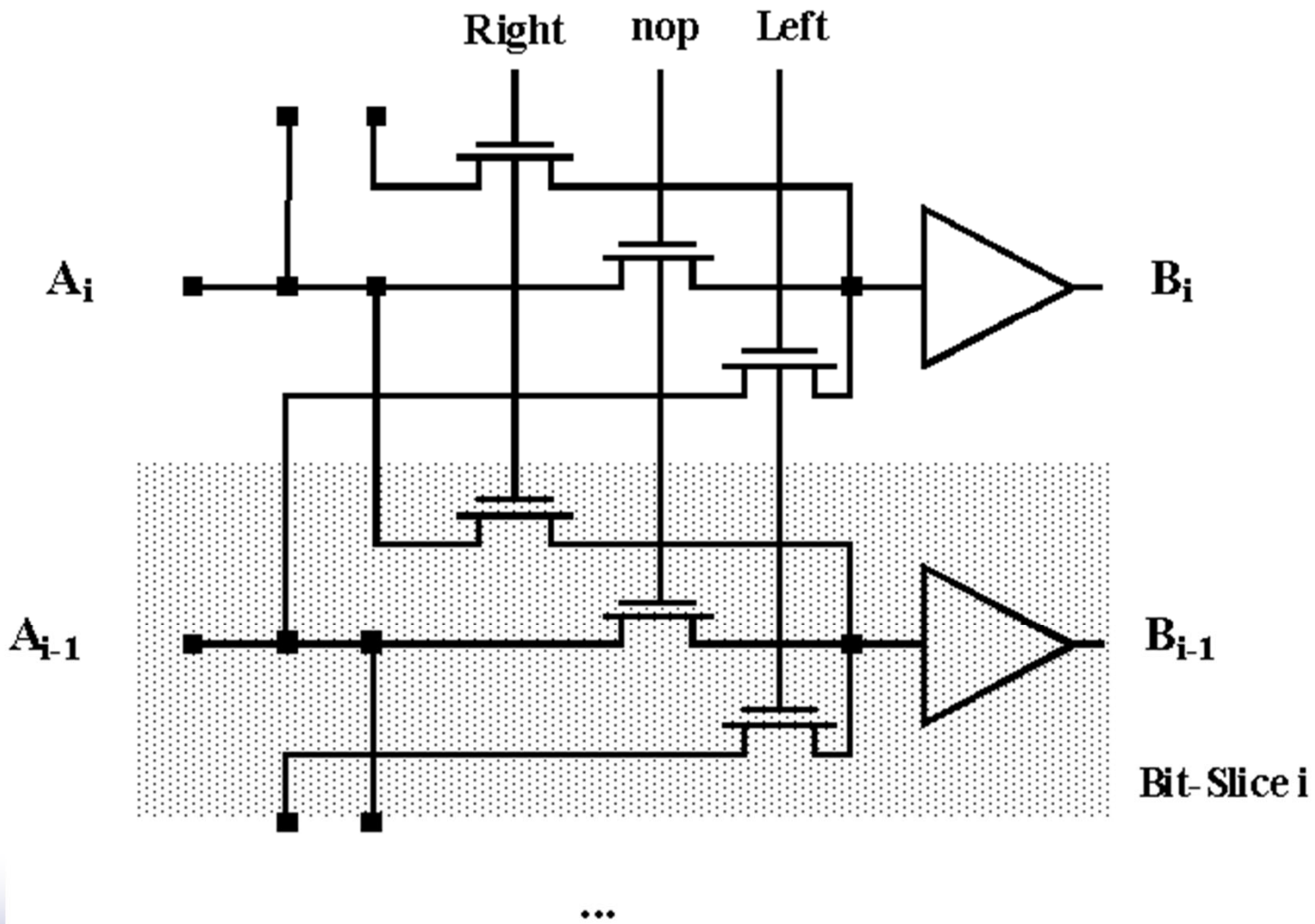
乘法器—总结

- 乘法器的优化目标与加法器不同
  - 需要确定关键路径
  - 采用并行处理的方法减小关键路径延时
- 其它可能的技术
  - 对数延时 VS. 线性延时（Wallace树乘法器）
  - 编码技术（Booth编码）
  - 流水线技术

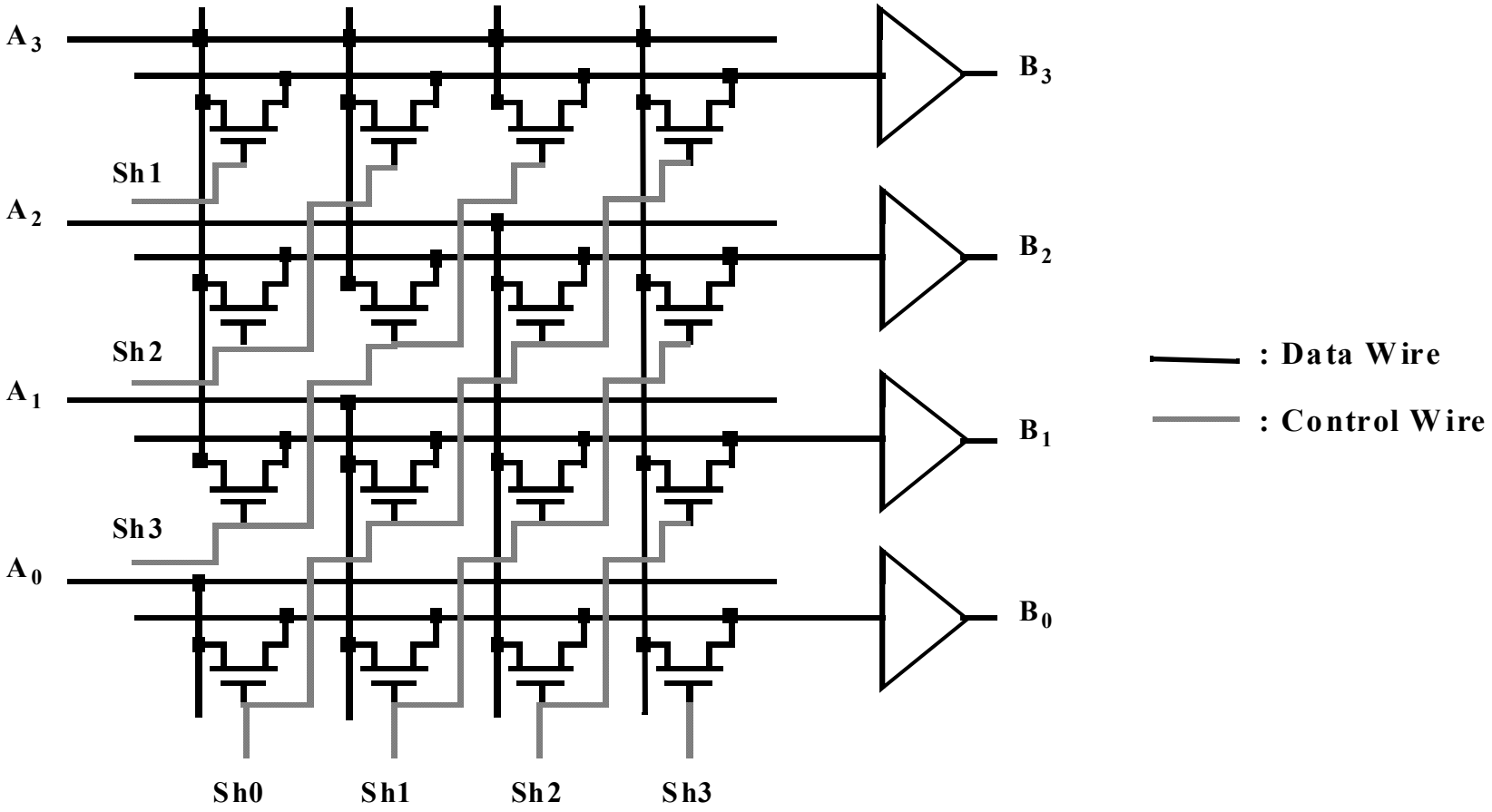
First glimpse at system level optimization.

移位器

一位移位器

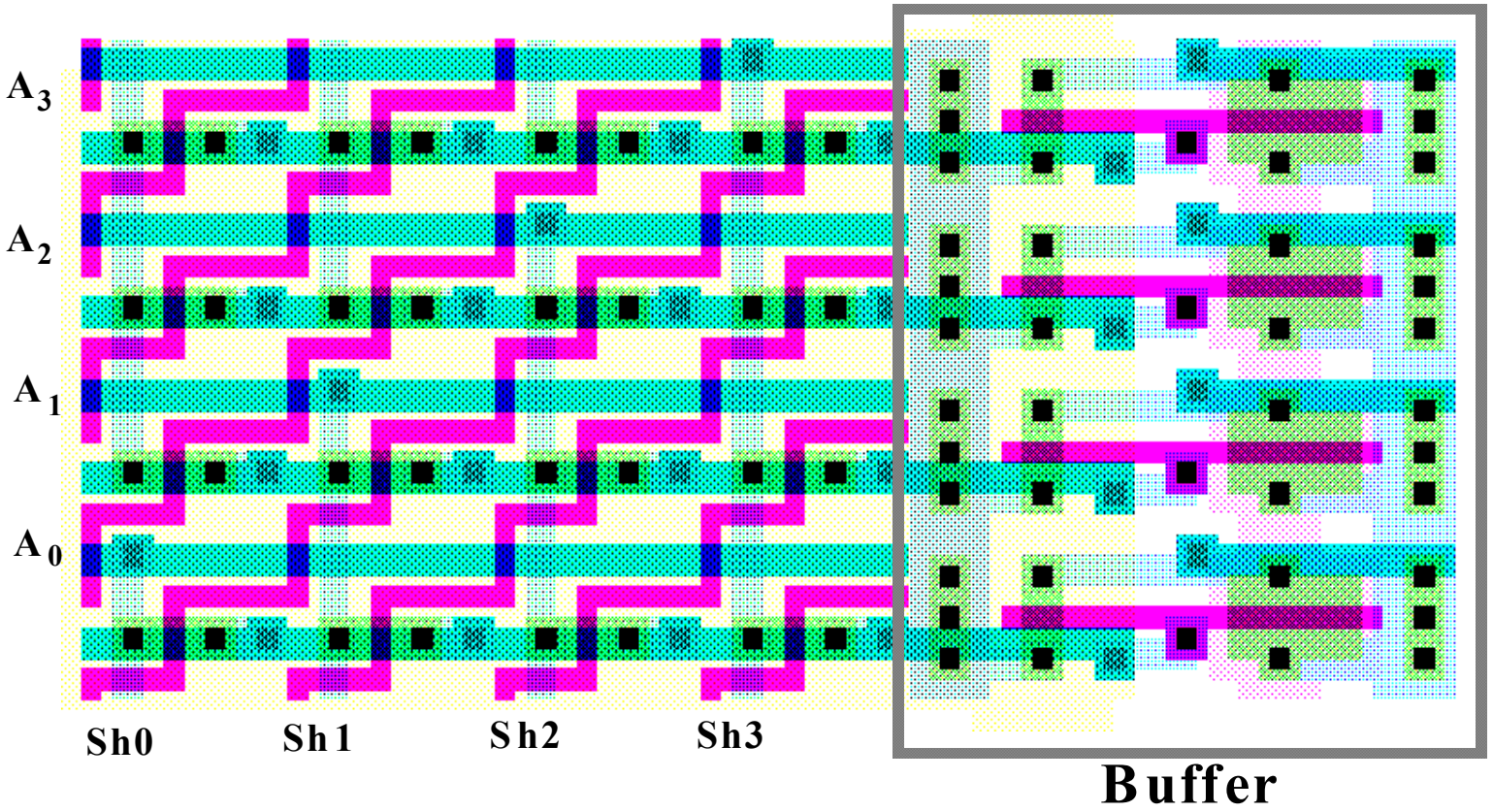


The Barrel Shifter 桶形移位器



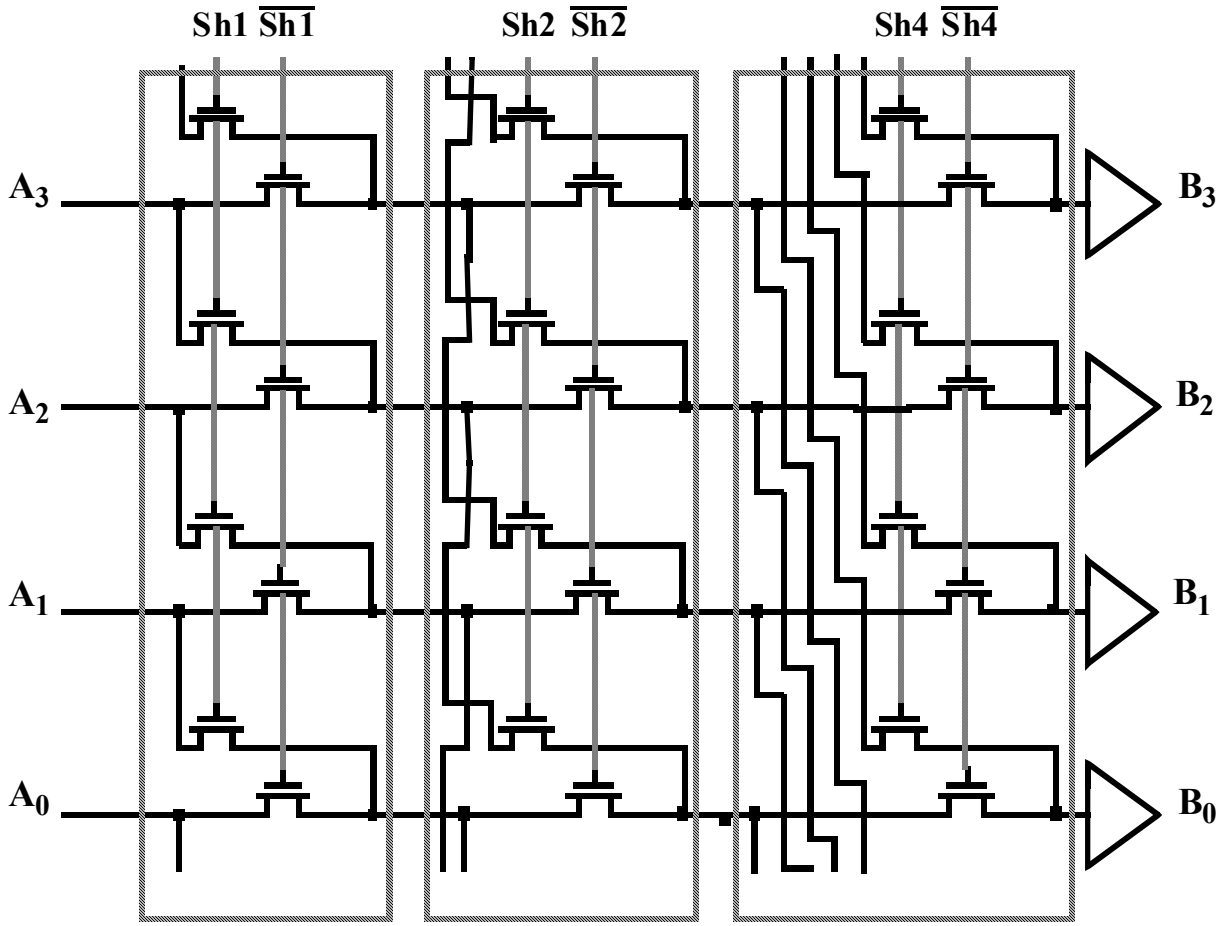
Area Dominated by Wiring

4x4 barrel shifter

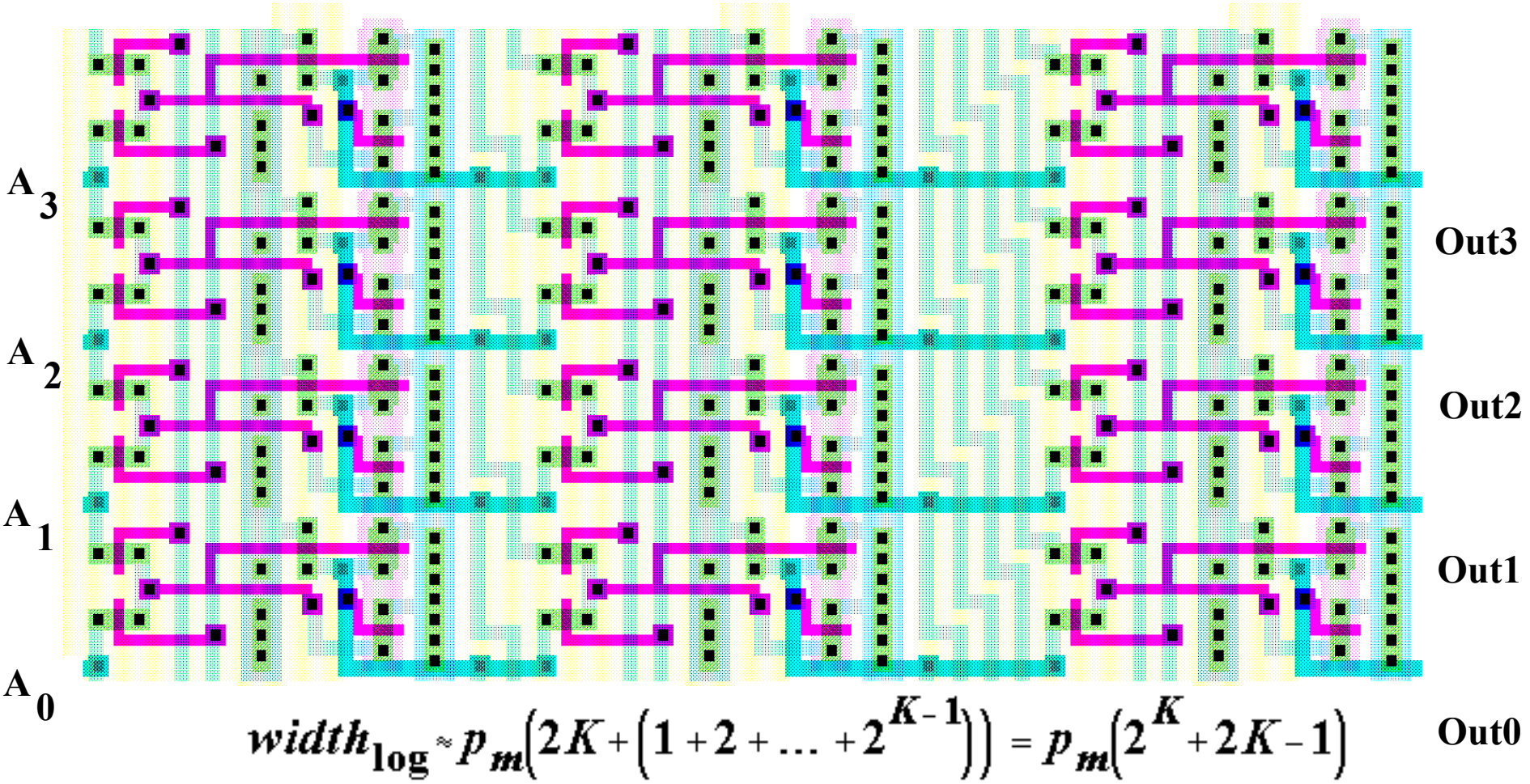


$$Width_{barrel} \sim 2\, p_m\, M$$

Logarithmic Shifter 对数移位器



0-7 bit Logarithmic Shifter



$$width_{\log} \approx p_m\left(2K + \left(1 + 2 + \ldots + 2^{K-1}\right)\right) = p_m\left(2^K + 2K - 1\right)$$