Digital Integrated Circuits
(83-313)

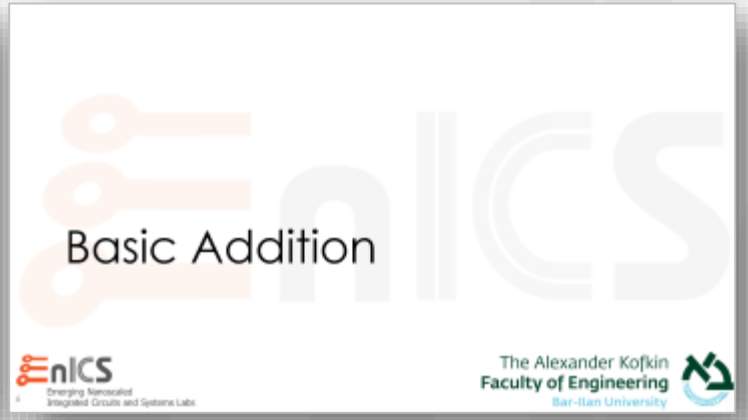
Lecture 10:
Arithmetic Circuits

Prof. Adam Teman
10 June 2021

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

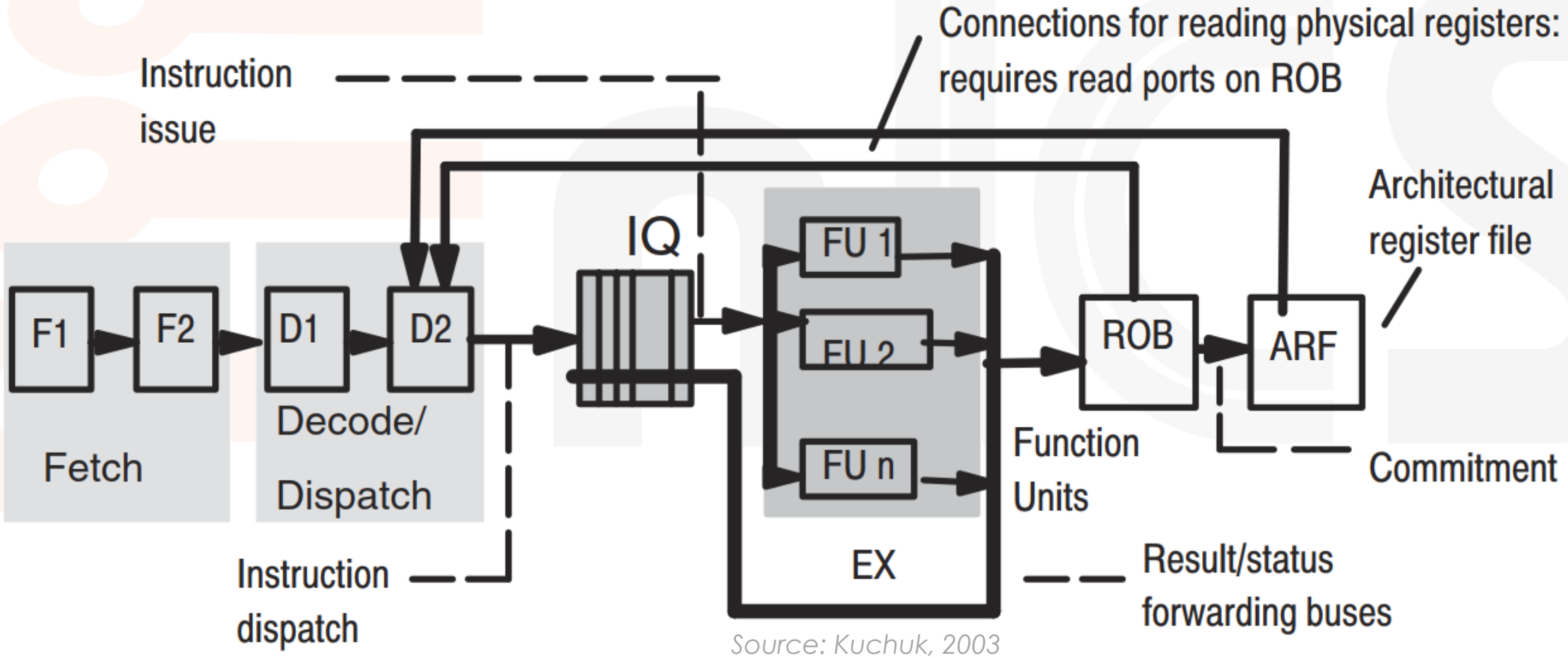
Lecture Content



DataPaths

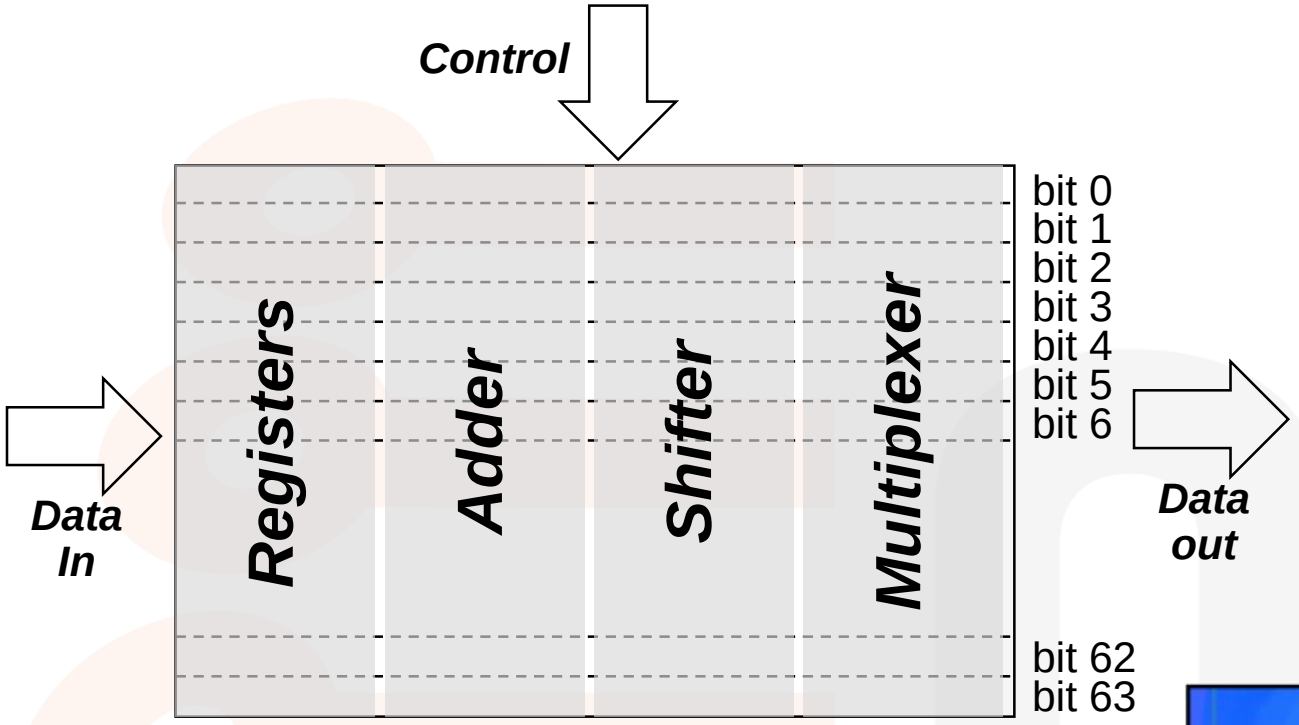
Multiple functional units

- **A complex processor may have multiple functional units working in parallel:**

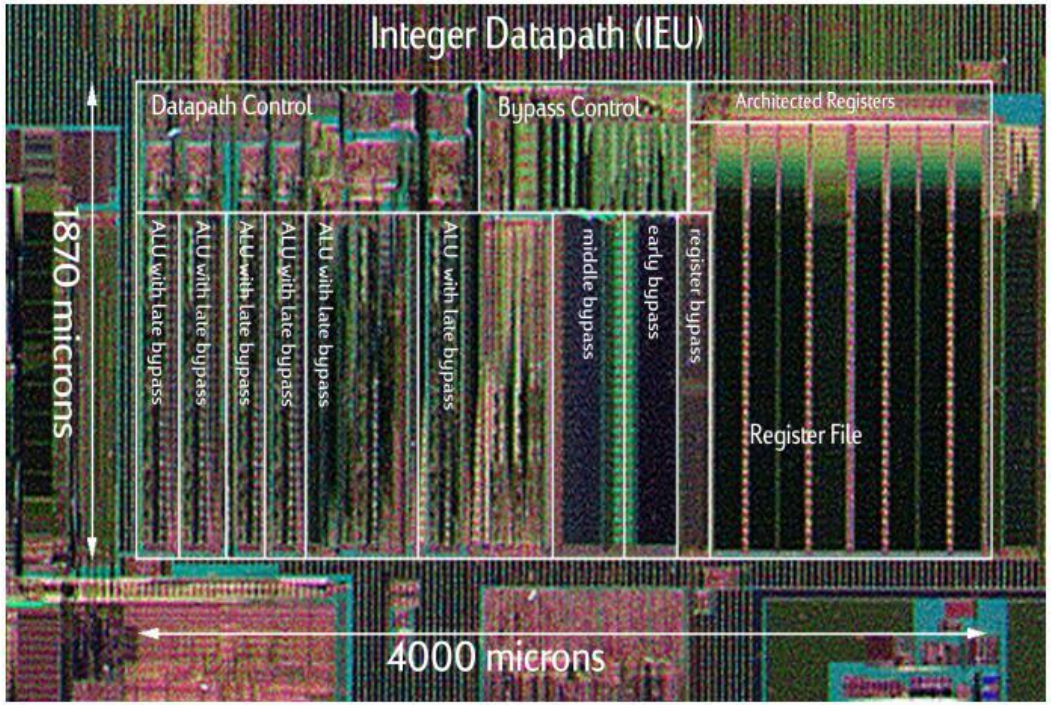


Source: Kuchuk, 2003

Bit-Sliced Design

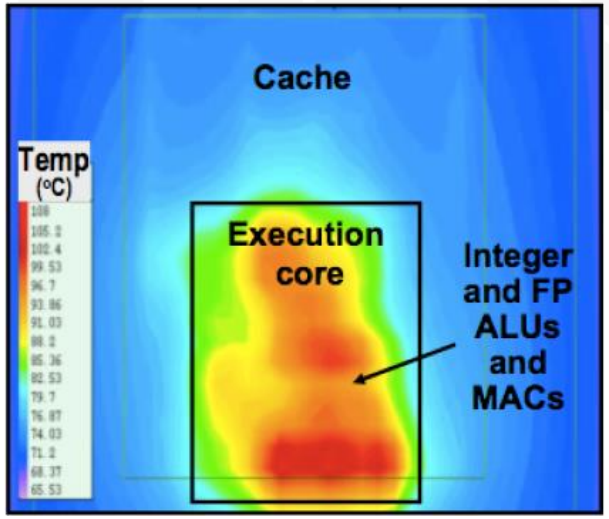


Tile identical Processor Elements



Fetzer, Orton, ISSCC'02

Processor thermal map

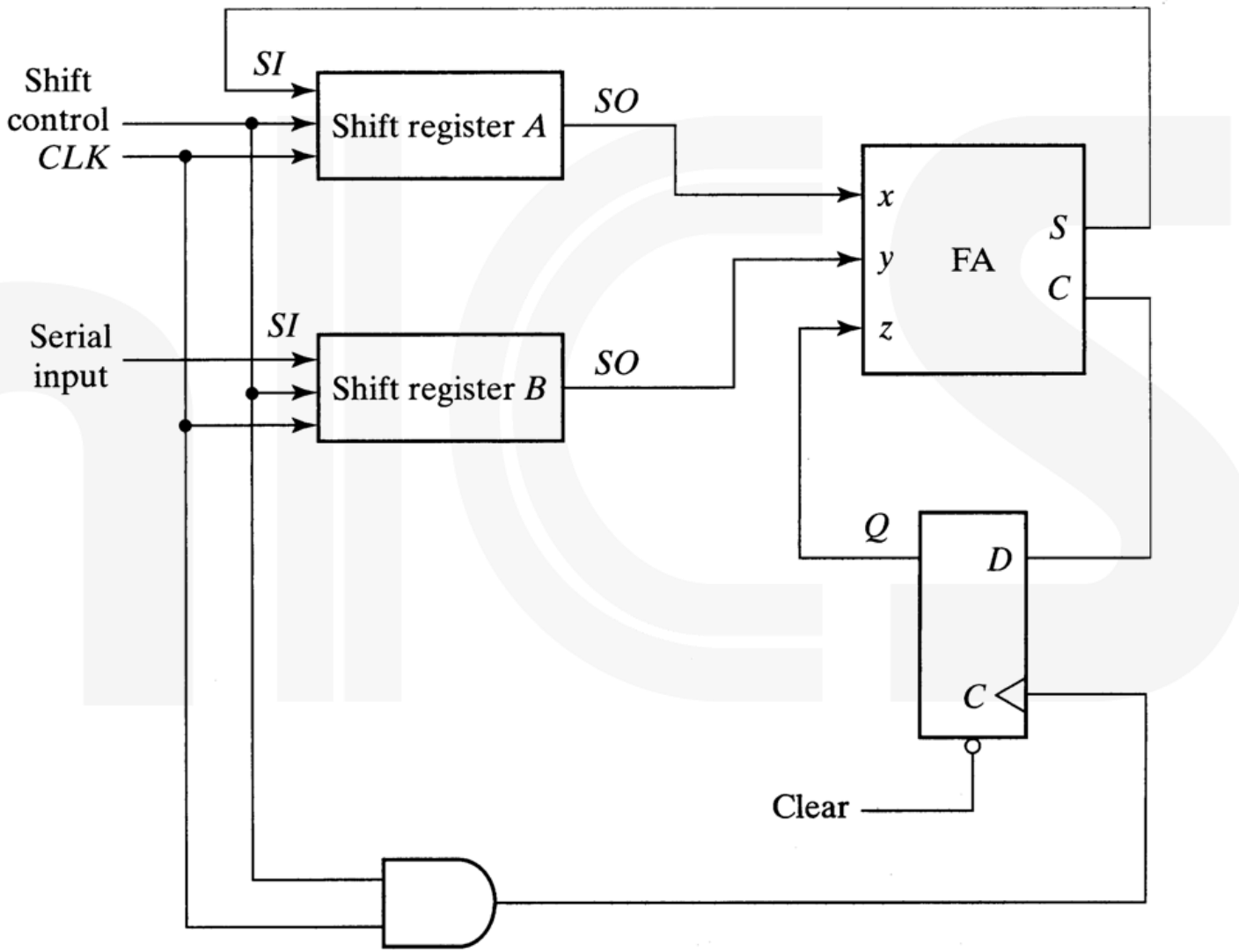


Design for energy efficiency!

Basic Addition

Serial Adder Concept

- At time i, read a_i and b_i. Produce s_i and c_i+1
- Internal state stores c_i. Carry bit c_0 is set as c_{in}

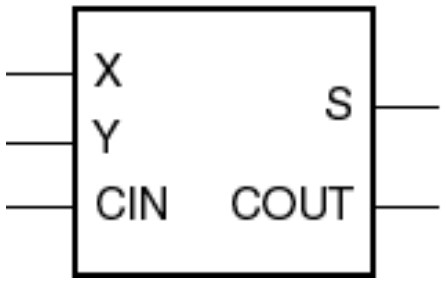


Source: Gate Overflow

Basic Addition Unit – Full Adder

X	Y	Cin	S	Cout
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

full adder



$$S = x \oplus y \oplus C_{in}$$

$$\Rightarrow S = P \oplus C_{in}$$

$$C_{out} = xy + xC_{in} + yC_{in}$$

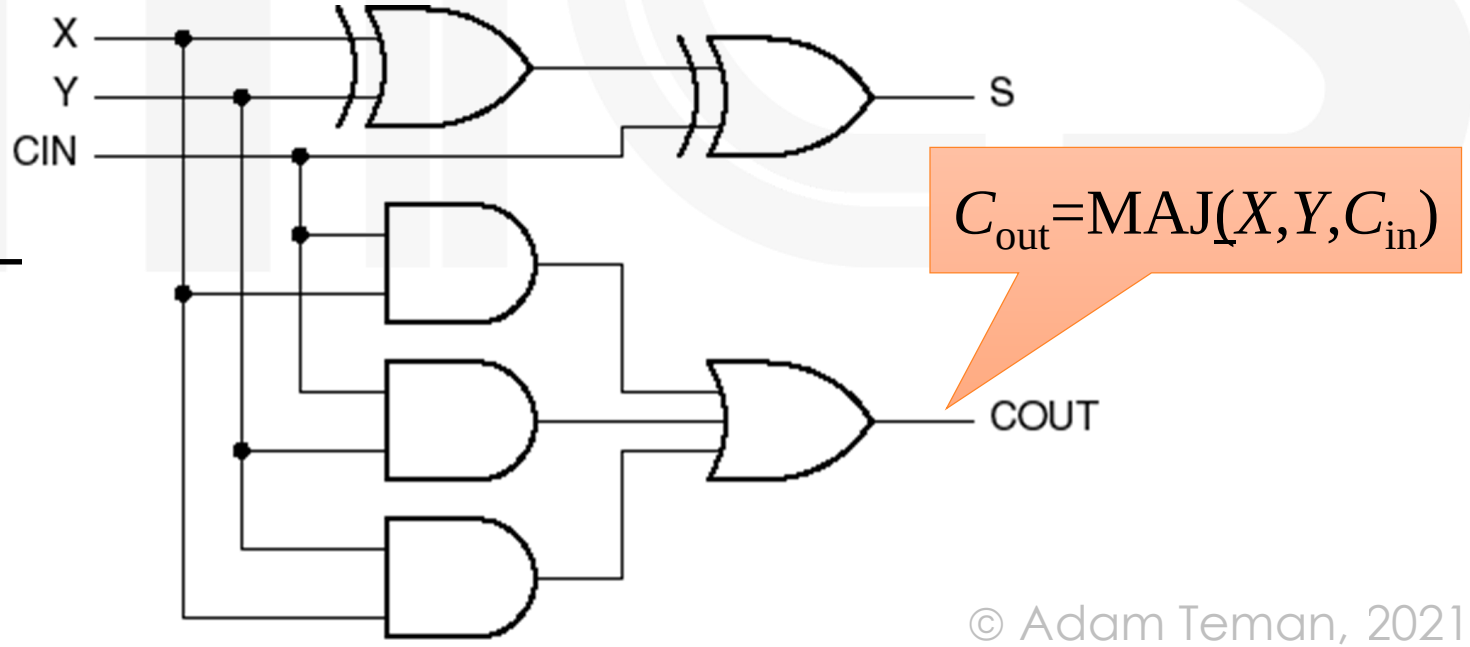
$$\Rightarrow C_{out} = G + P \cdot C_{in}$$

$\text{Kill} = \overline{x} \cdot \overline{y}$

$\text{Generate} = x \cdot y$

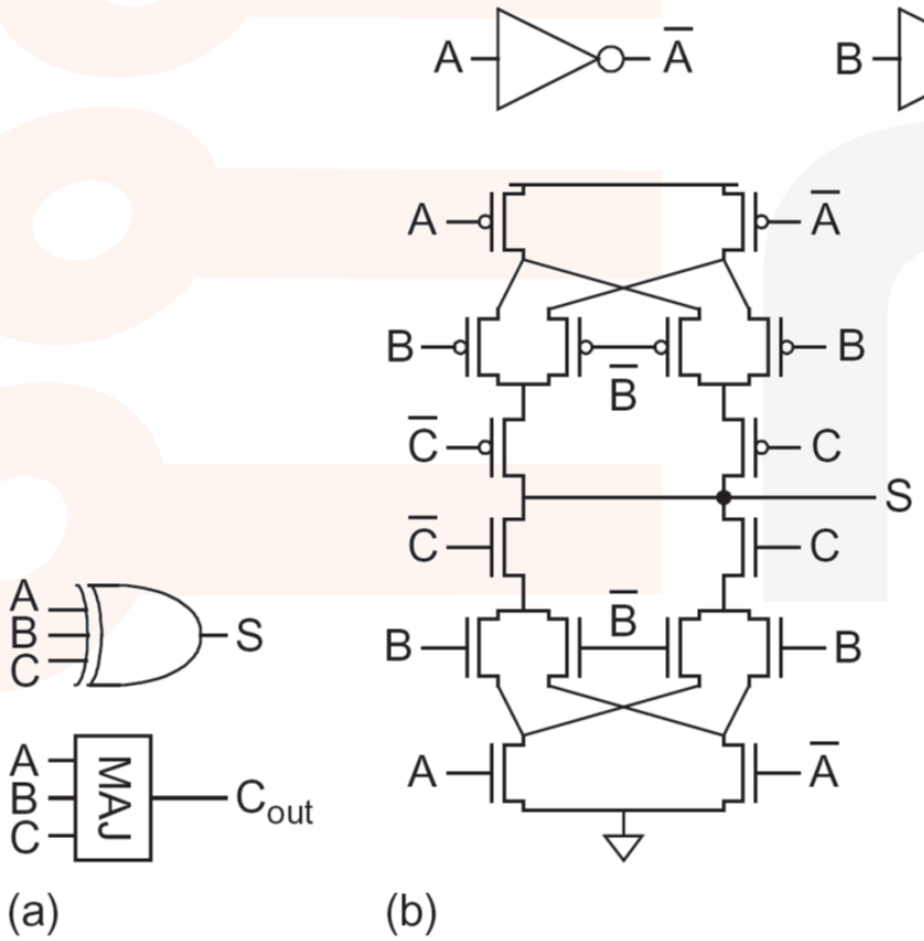
$\text{Propagate} = x \oplus y$

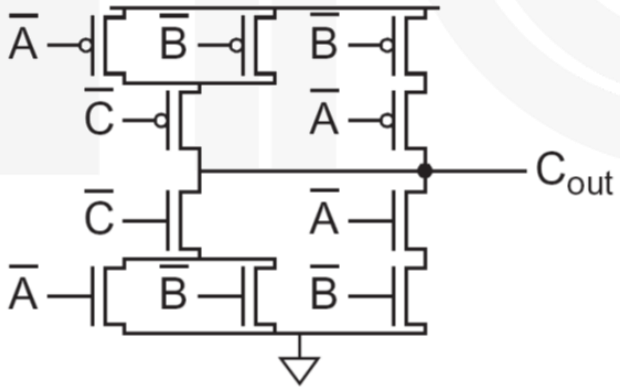
$\approx x + y$



Full-Adder Implementation

- **A full-adder is therefore a majority gate and a 3-input XOR:**



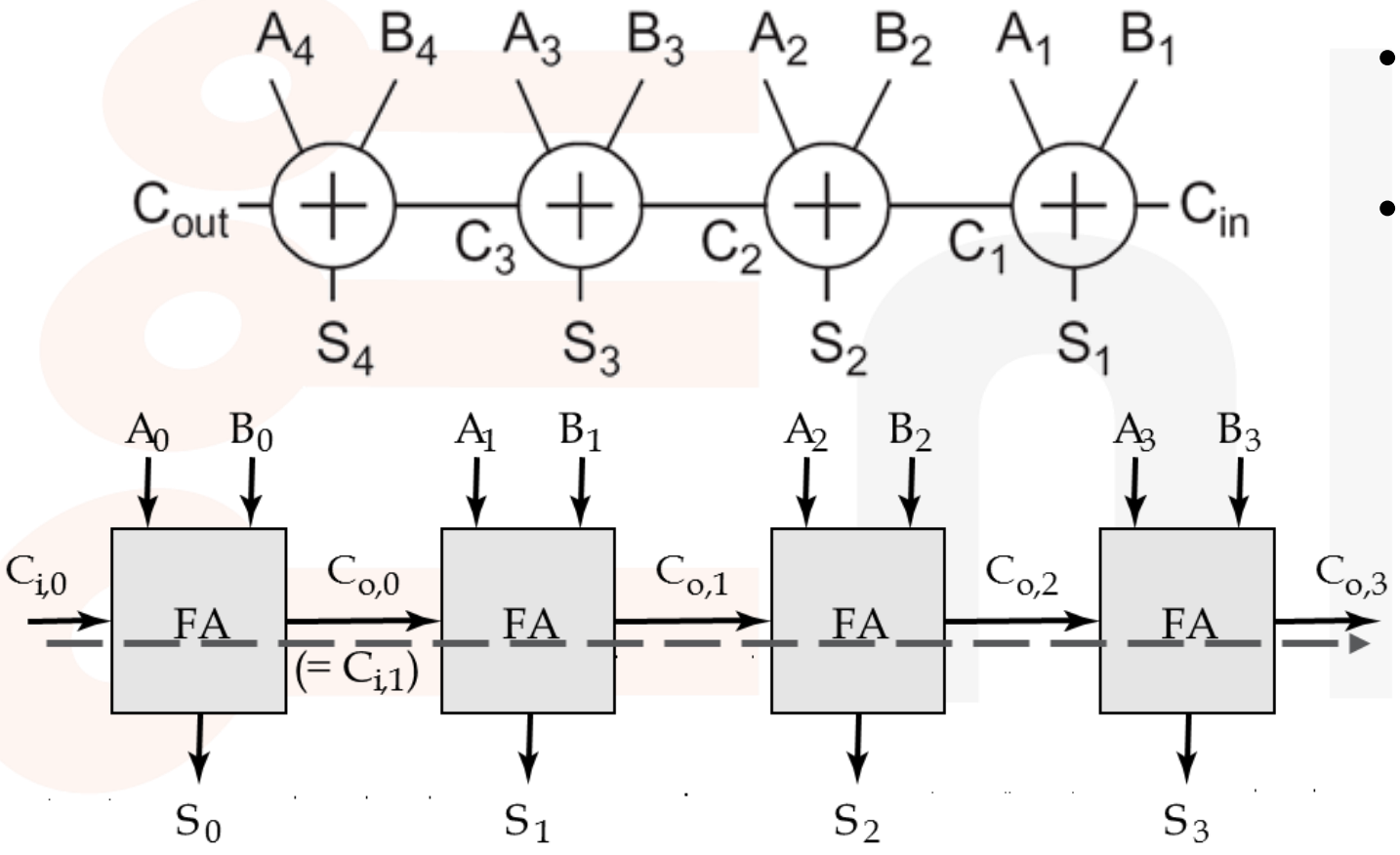


Total: 32 Transistors

X	Y	Cin	S	Cout
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

Source: CMOS VLSI Design

Ripple Carry Adder



- So, it is clear, the C_{out} output of the Full Adder is on the critical path.
- Can we exploit this to improve the design?

$$S = A \oplus B \oplus C_{in} =$$

$$= ABC_{in} + \left(A + B + C_{in} \right) \bar{C}_{out}$$

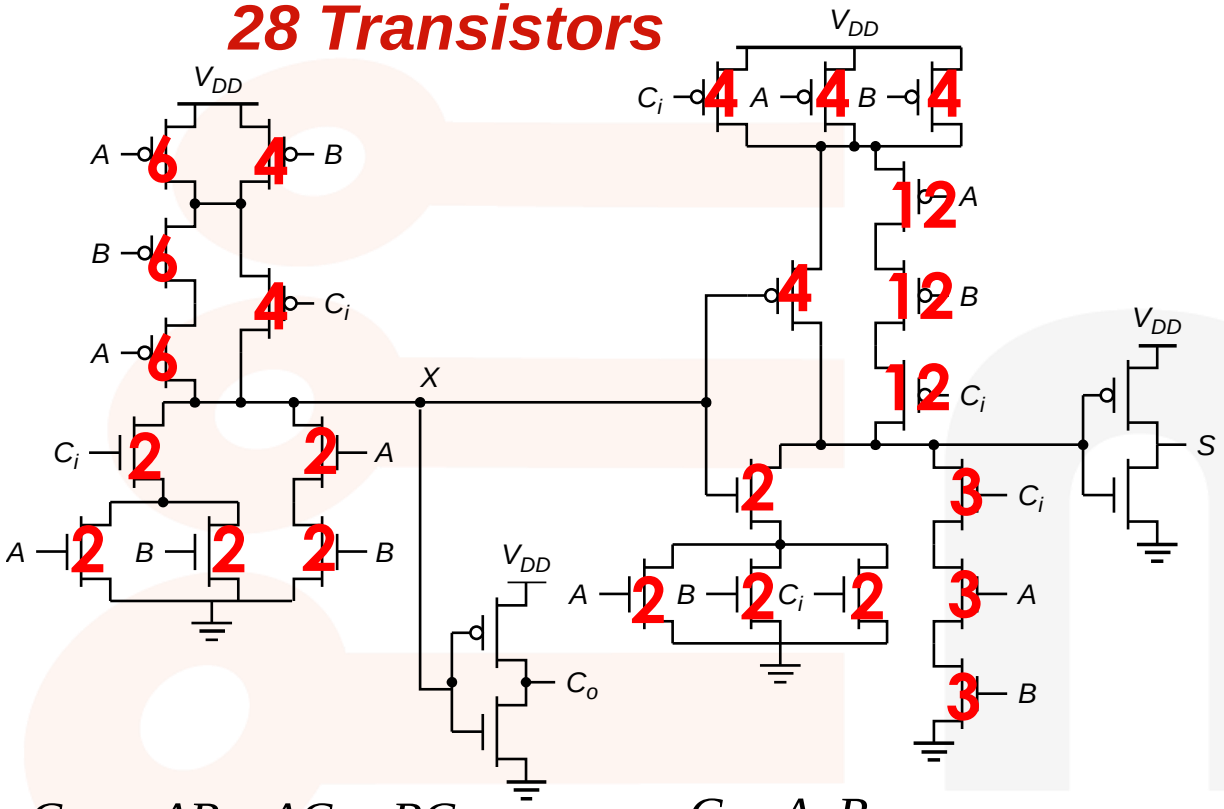
$$t_{adder} = (N\text{-}1)t_{carry} + t_{sum} \quad \Rightarrow \quad t_{pd} = O(N)$$

Source: CMOS VLSI Design

Full-Adder Implementation

$$S = ABC_{in} + (A+B+C_{in})\bar{C}_{out}$$

28 Transistors

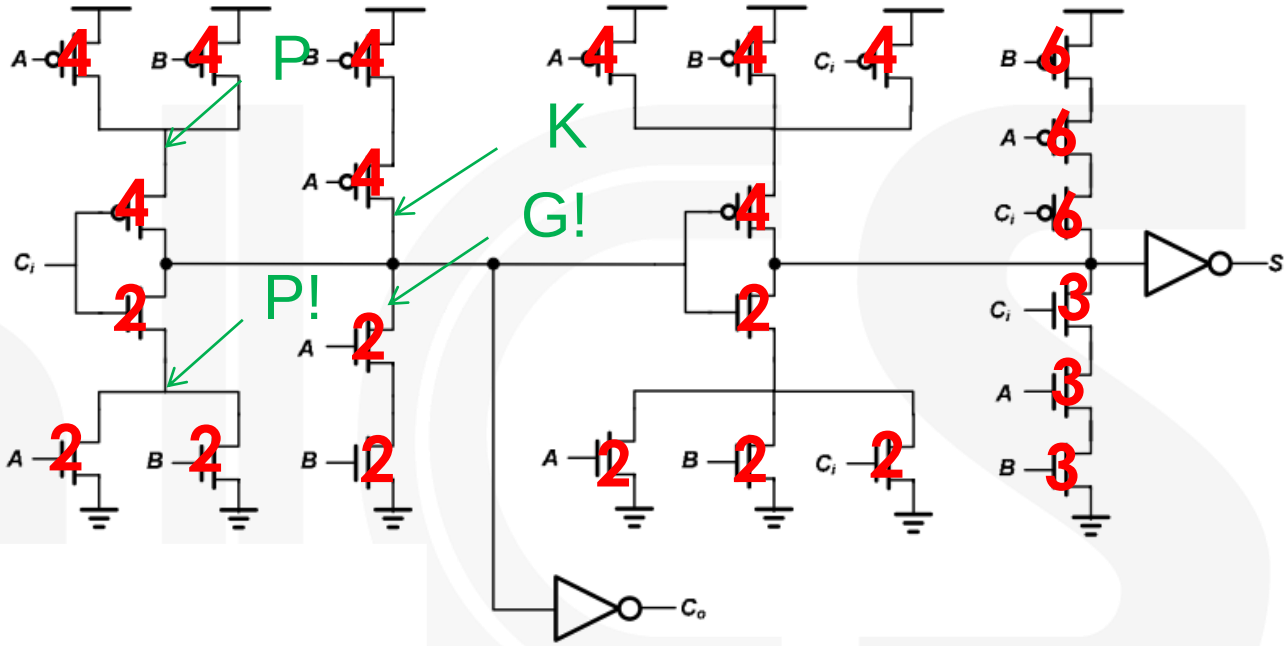


$$C_{out} = AB + AC_i + BC_i \qquad G = A \cdot B$$

$$S = ABC_{in} + (A+B+C_{in})\bar{C}_{out} \qquad P = A \oplus B$$

$$LE_{C_i} = \frac{2+4+2+3+12+4}{3} = 9$$

24 Transistors

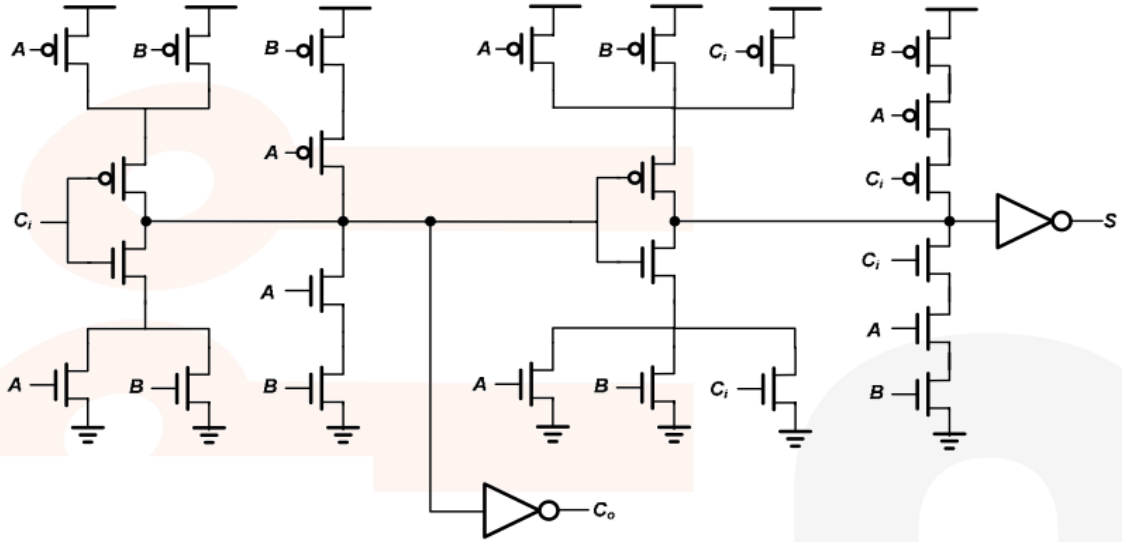


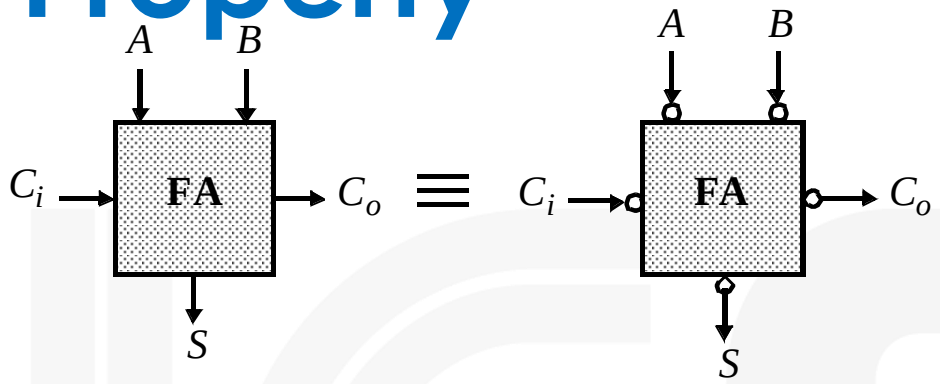
$$LE_{C_i} = \frac{2+4+2+4+6+3}{3} = 7$$

…BUT ~64 stages to propagate

i.e., $PE_{opt} = 4^{64}$

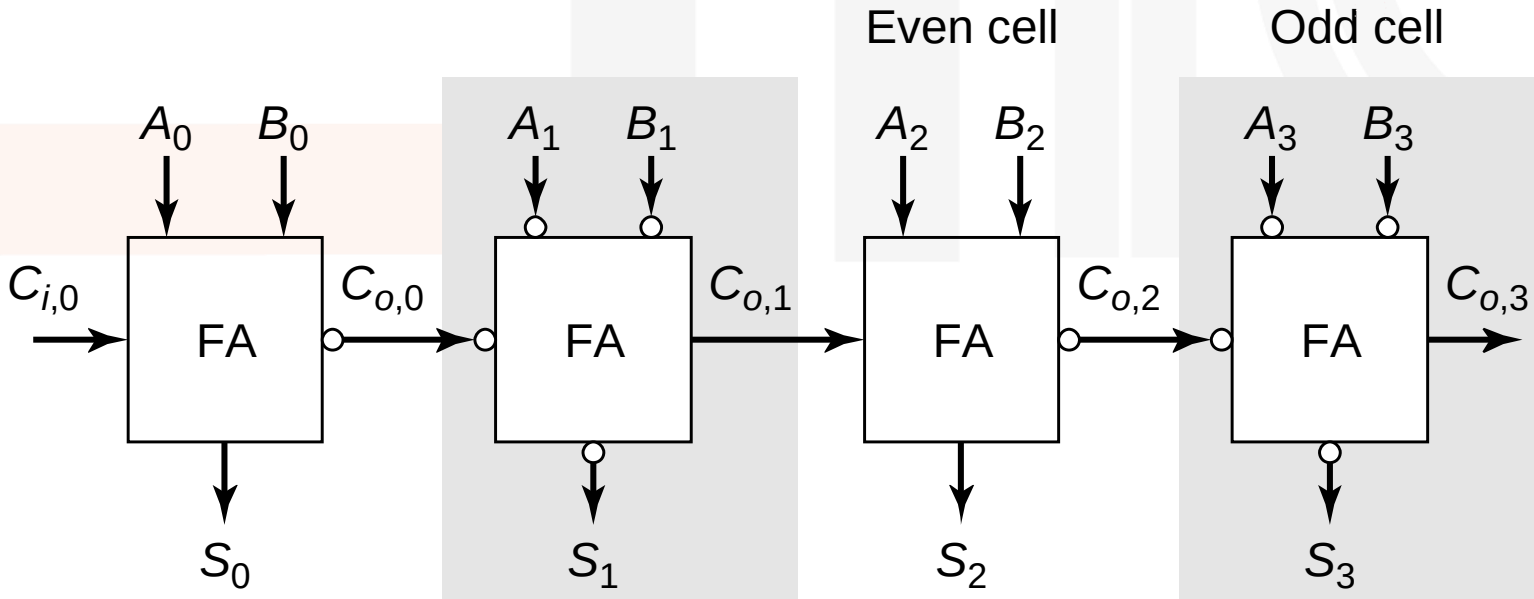
Exploiting the Inversion Property





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

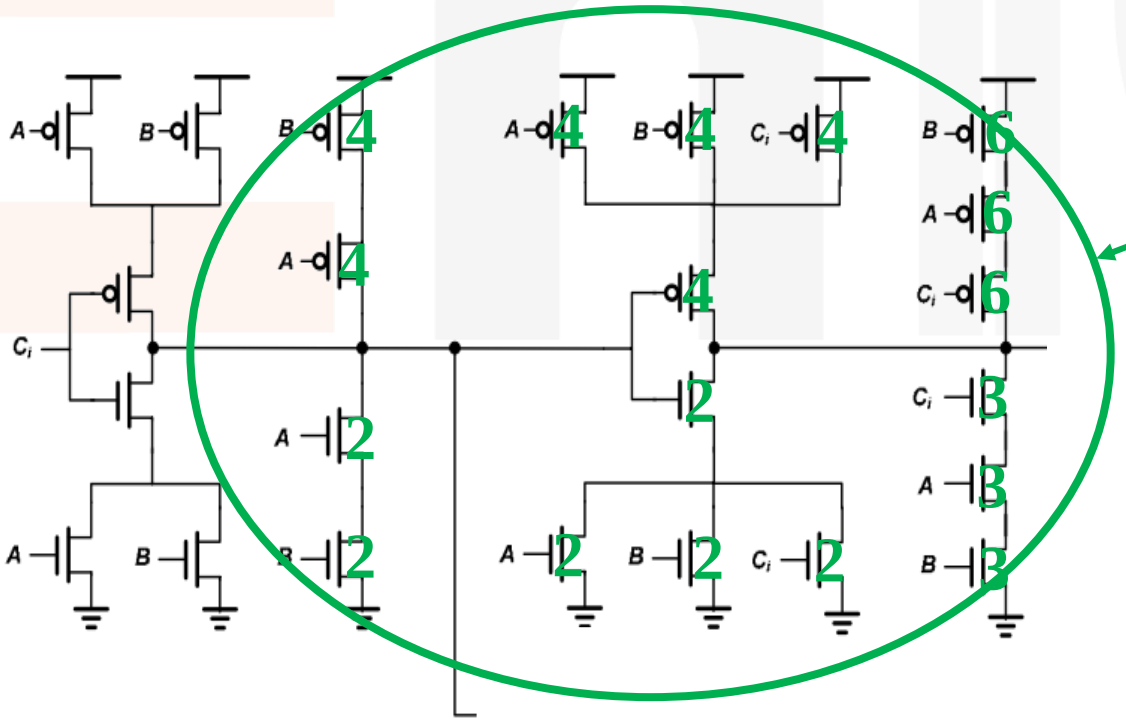
$$\overline{C_o}(A,B,C_i) = C_o(\bar{A},\bar{B},\bar{C_i})$$



We saved the inverter, so $PE_{opt}=4^{32}$

Sizing the Mirror Adder

- **Problem: How can we make a high speed bitslice layout?**
  - If we upsize each stage according to Logical Effort, we will have non-identical bitslices.
  - Such upsizing will result in huge gates.
- **Why not design the adder to inherently achieve optimal Electrical Effort ($EF_{opt}=4$)?**
  - Assume everything *not on the carry path* can be sized like a minimum inverter!



Not on the critical path!

Sizing the Mirror Adder

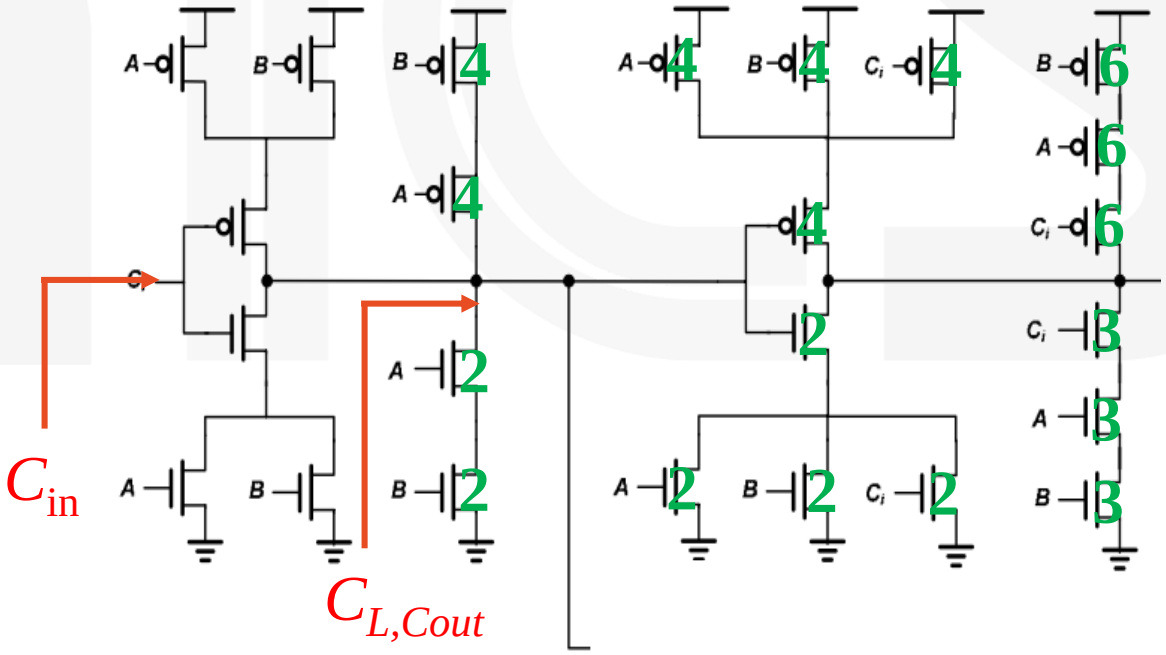
- **Now, let's try to size the first stage to get *EF*=4:**
  - Remember, logical effort is a function of gate topology and not sizing!
  - Therefore, we can temporarily size the first stage as a minimum sized inverter, giving us:

$$LE_{Cin} = \frac{4+2}{3} = 2$$

  - So to get *EF*=4:

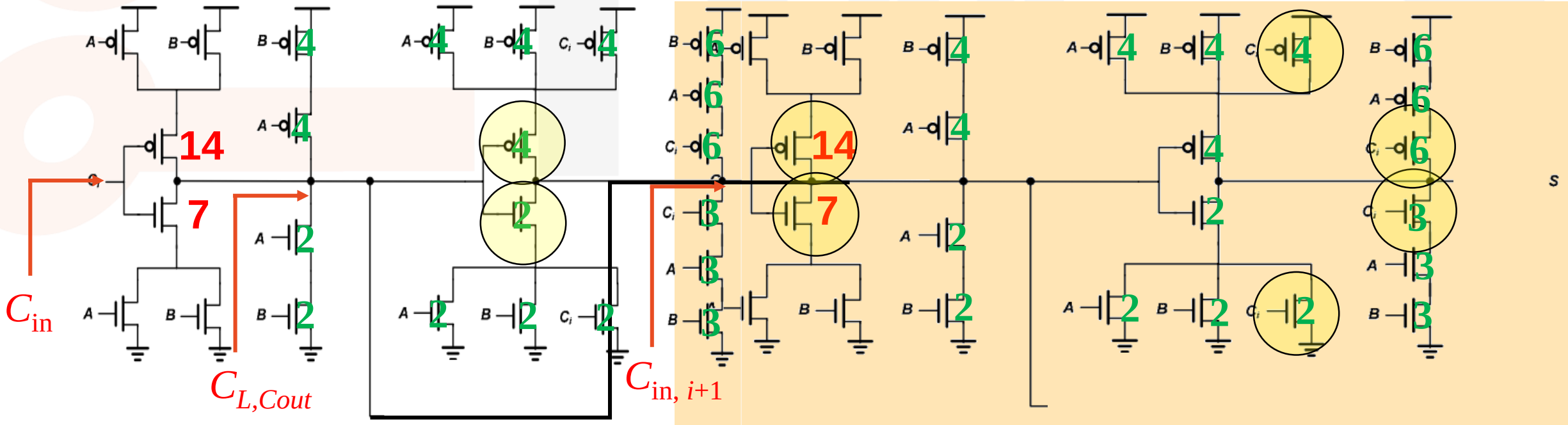
$$EF_{FA,Cin} = \frac{LE_{Cin} \cdot C_{L,Cout}}{C_{Cin}} \Rightarrow \left.\frac{C_{L,Cout}}{C_{Cin}}\right|_{EF=4} = 2$$

  - But what is $C_{L,Cout}$?



Sizing the Mirror Adder

- **What is $C_{L,Cout}$?**
  - Obviously, we have the second stage…
  - But don't forget the next full adder!
- **So $C_{L,Cout}$ is:** $C_{L,Cout} = 6 + C_{Cin} + 6 + 9 = C_{Cin} + 21$
- **And now, we can find C_{in} using the *EF* constraint we found:** $\left.\frac{C_{L,Cout}}{C_{Cin}}\right|_{EF=4} = 2 \Rightarrow C_{Cin} = 21$

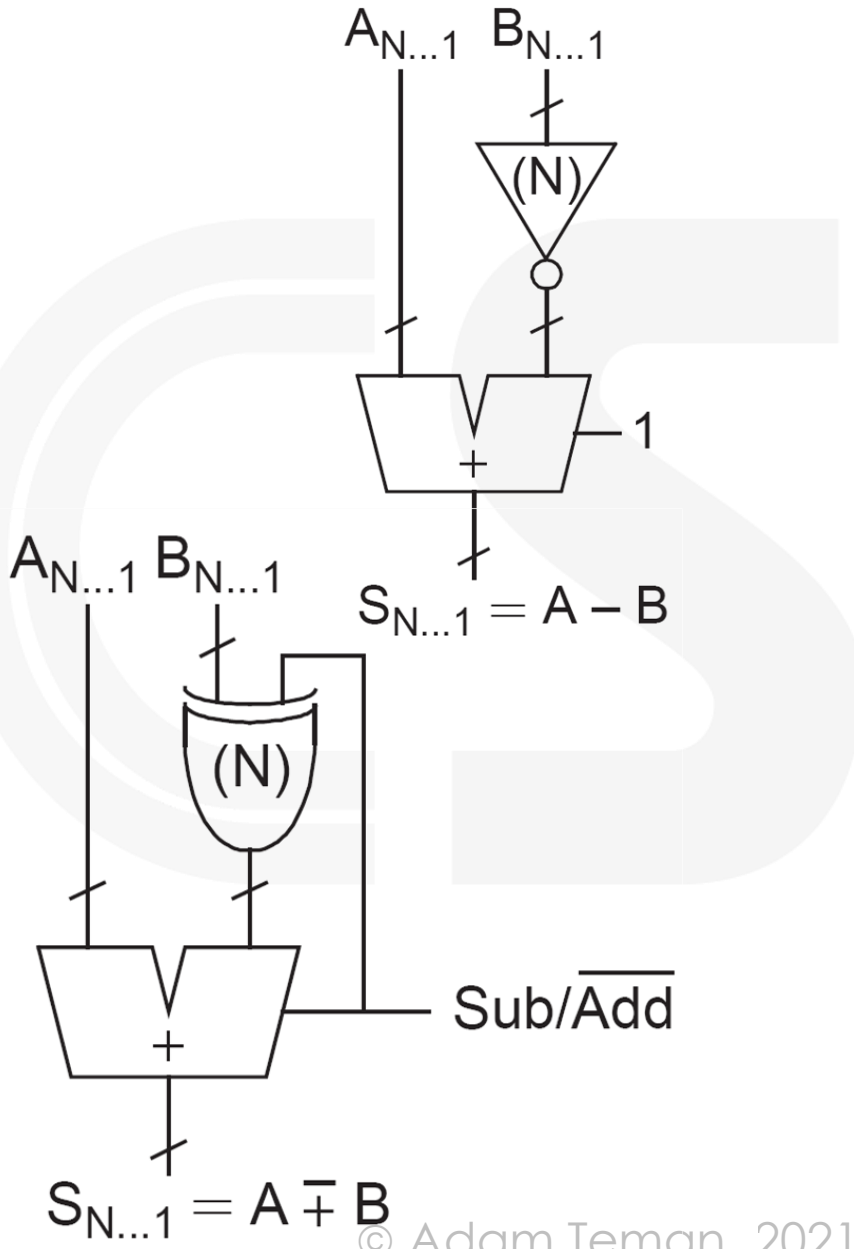


Subtraction

- **To subtract two's complement, just remember that:**

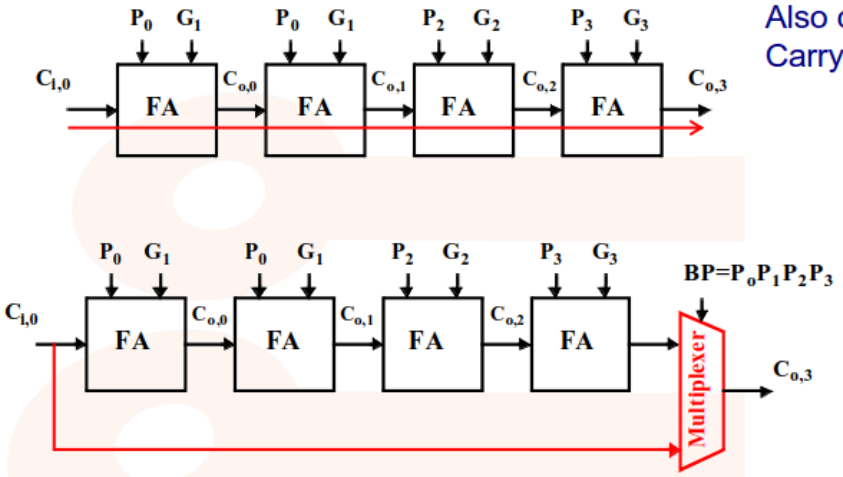
$$-x = \bar{x} + 1 \quad \Rightarrow \quad A - B = A + \bar{B} + 1$$

- **So, to subtract:**
  - Invert one of the operands.
  - Add a carry in to the first bit.
- **Therefore, to provide an adder/subtractor:**
  - Add an XOR gate to the B-input
  - Use the sub/add selector to the XOR and carry in.



Faster Adders

Carry-Skip (Carry Bypass) Adder



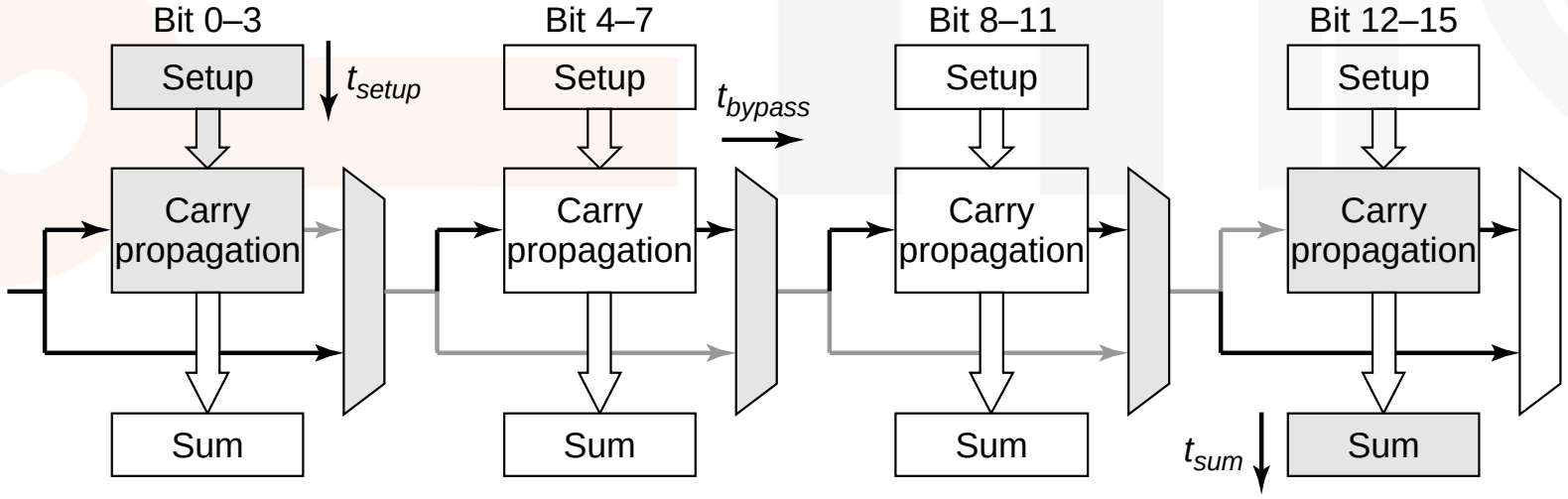
Also called Carry-Skip

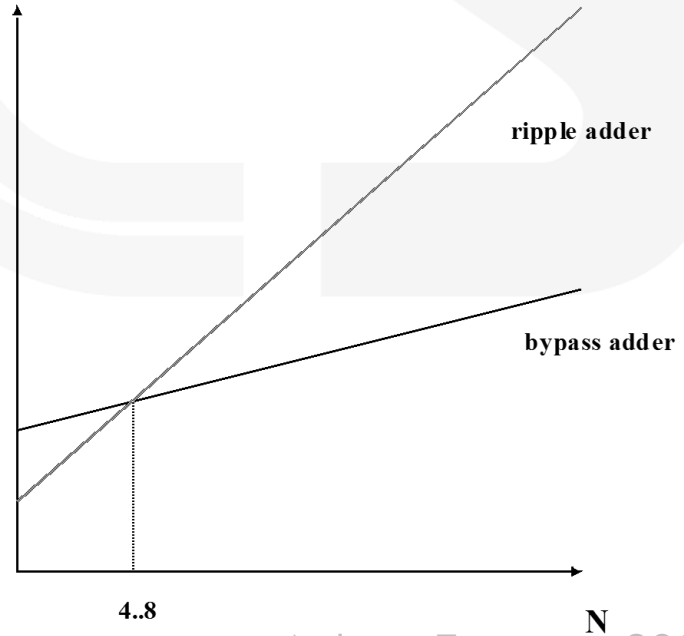
M Sections of (N/M) Bits Each

$$t_{\text{skip}} = t_{\text{p/g}} + \left(\frac{N}{M} - 1\right) t_{\text{carry}} + (M-1)\, t_{\text{bypass}} + t_{\text{sum}}$$

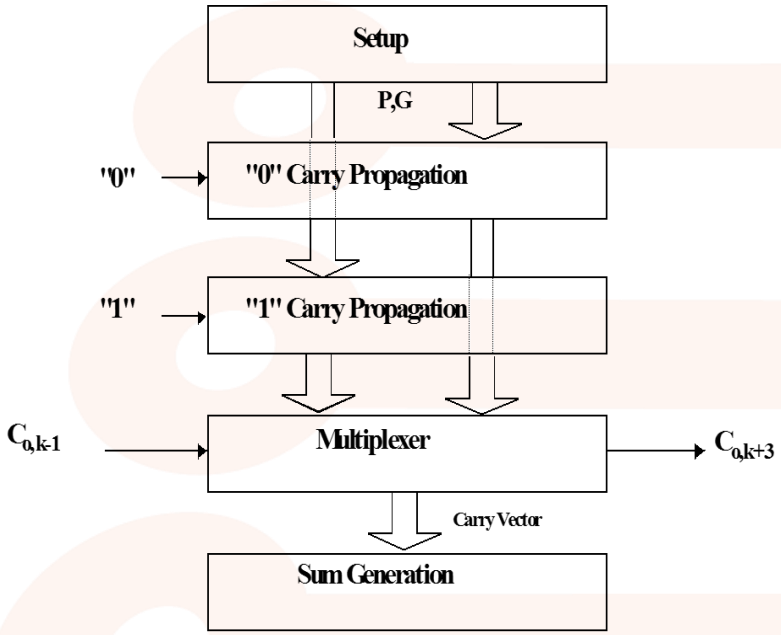
$$\Rightarrow \mathrm{O}\left(N/M\right)$$

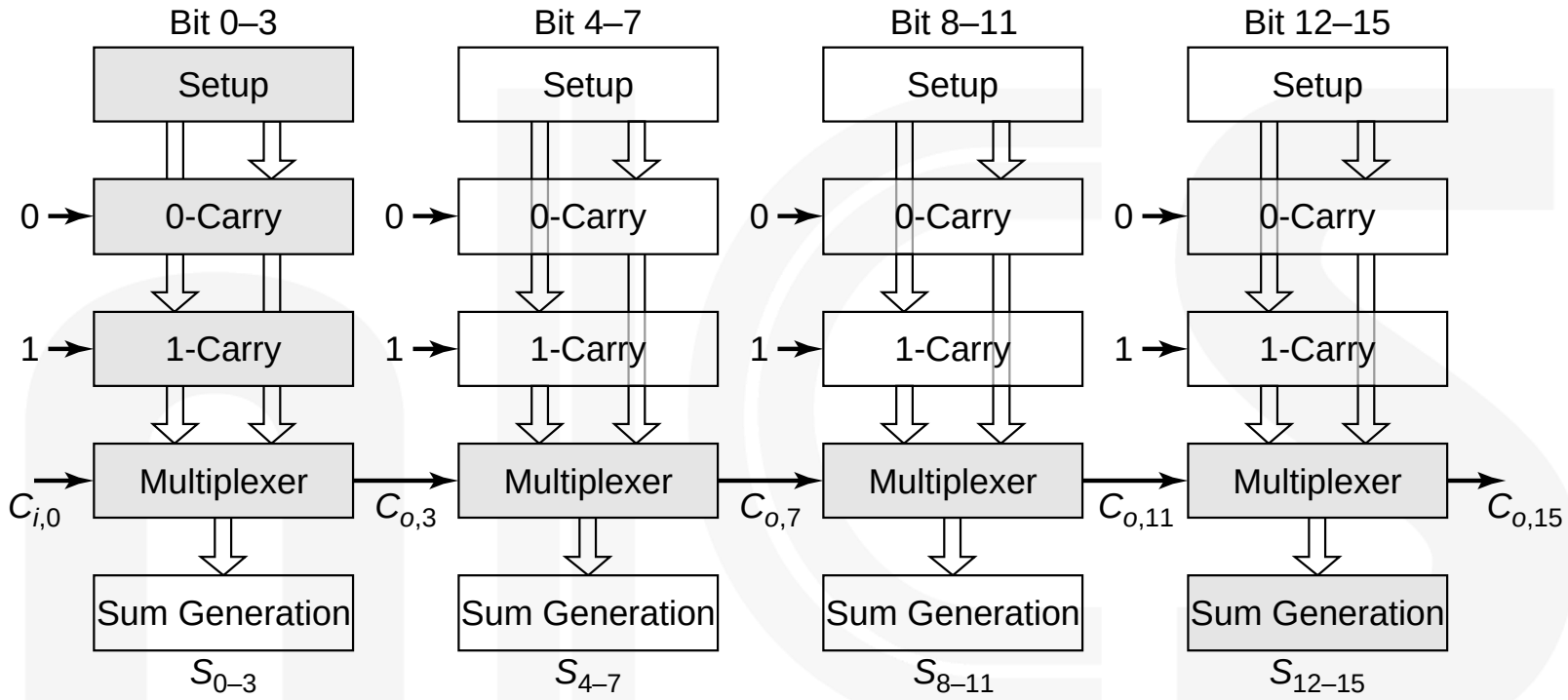
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else "kill" or "generate".





Carry-Select Adder



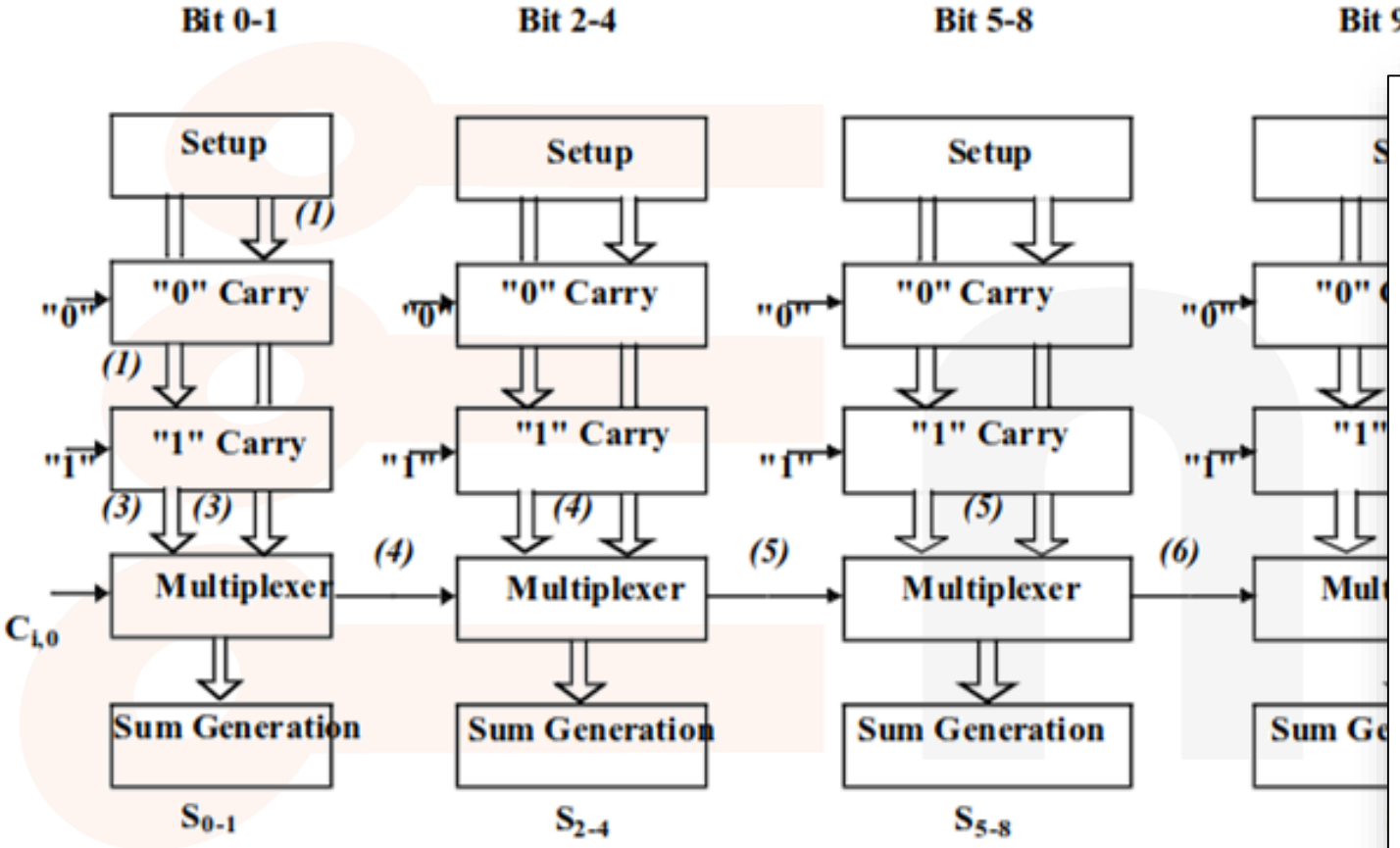


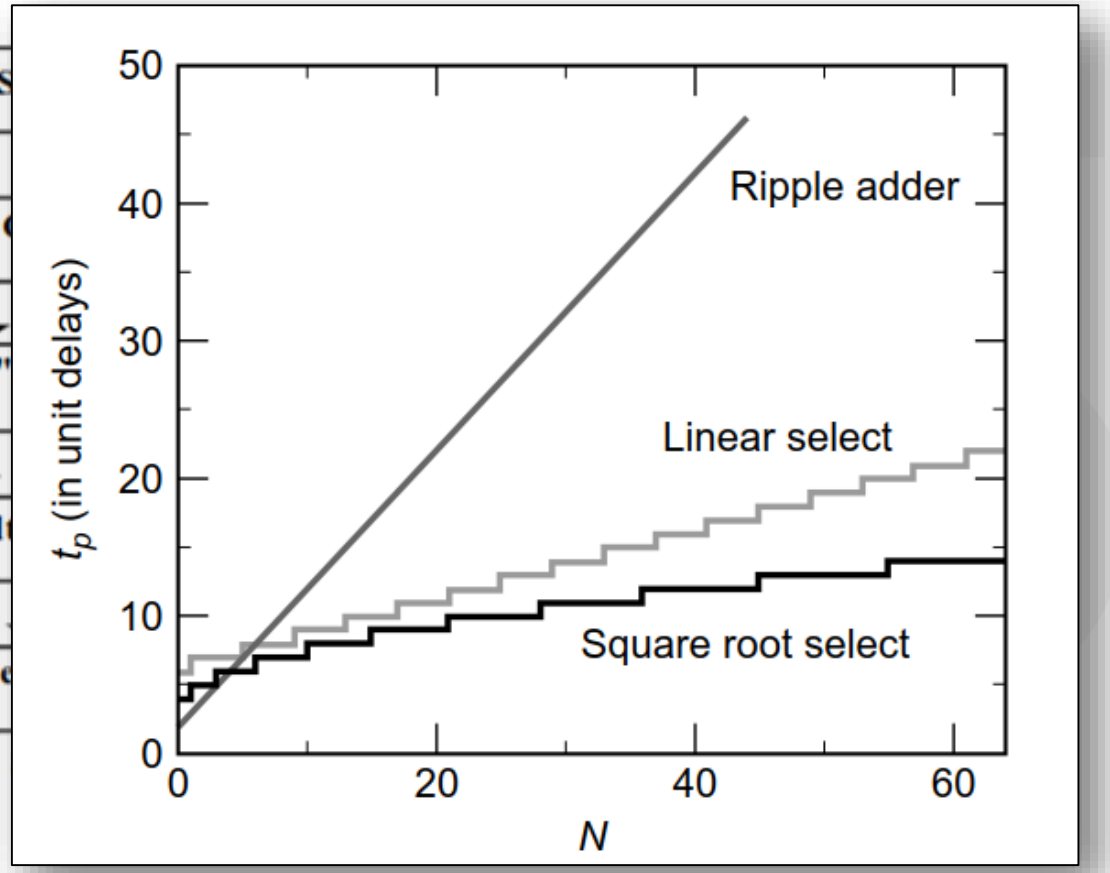
Let's guess the answer for each value of the carry.

N-bit input with M CSA blocks

$$t_{\text{select}} = t_{\text{p/g}} + \frac{N}{M} t_{\text{carry}} + M \cdot t_{\text{mux}} + t_{\text{sum}} \quad \Rightarrow \mathrm{O}\left(N/M\right)$$

Square Root Carry Select





$$t_{\text{sqrt}} = t_{\text{p/g}} + Mt_{\text{carry}} + \sqrt{2N}t_{\text{mux}} + t_{\text{sum}} \Rightarrow \text{O}\left(\sqrt{2N}\right)$$

Carry Lookahead Adder – Basic Idea

- **Problem – $C_{\text{out},k}$ takes approximately k gate delays to ripple.**
- **Question – can we calculate the carry without any ripple?**

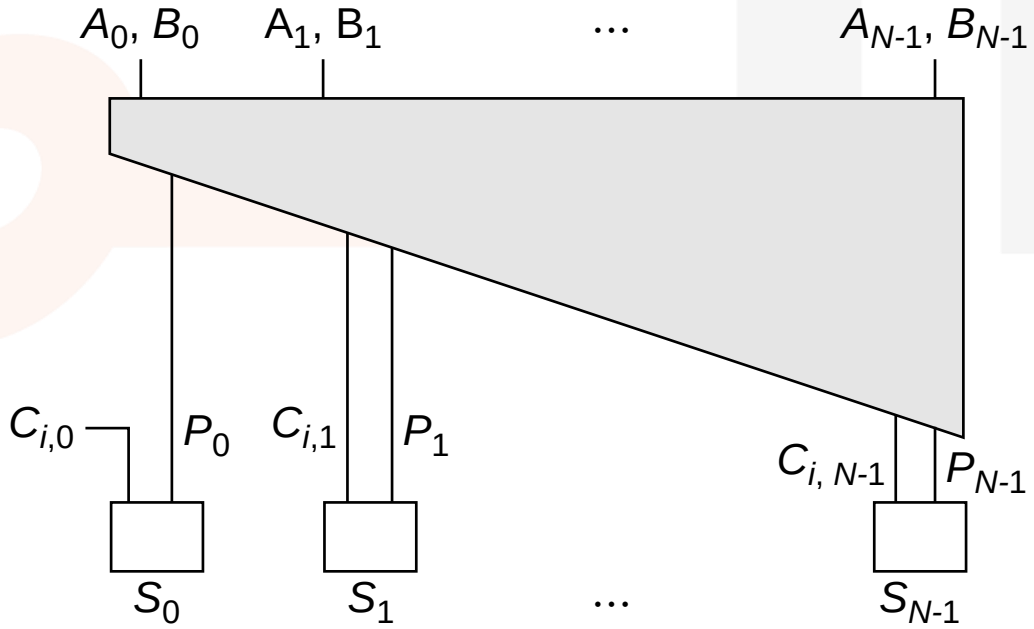
$$G_i = A_i \cdot B_i$$

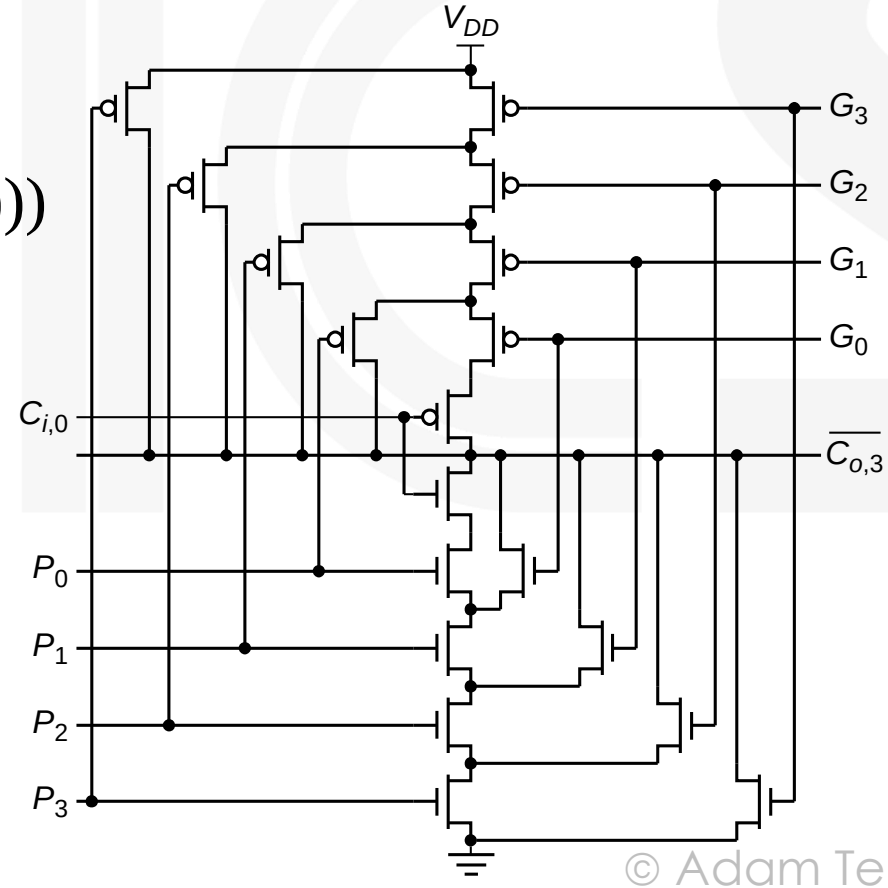
$$P_i = A_i \oplus B_i$$

$$C_{\text{out},k} = f(A_k, B_k, C_{\text{out},k-1}) = G_k + P_k \cdot C_{\text{out},k-1}$$

$$C_{\text{out},k} = G_k + P_k \cdot (G_{k-1} + P_{k-1} \cdot C_{\text{out},k-2})$$

$$C_{\text{out},k} = G_k + P_k \cdot (G_{k-1} + P_{k-1} \cdot (\ldots + P_1(G_0 + P_0 C_{\text{in},0})))$$





Tree Adders (Logarithmic CLA)

$$G_i = A_i \cdot B_i \qquad P_i = A_i \oplus B_i$$

$$S = P \oplus C_{\text{in}}$$

$$C_{\text{out}} = G + P \cdot C_{\text{in}}$$

- **Can we reduce the complexity of calculating P_i, G_i ?**

$$P_{1:0} = P_1 \cdot P_0 \qquad G_{1:0} = G_1 + P_1 \cdot G_0$$

$$\Rightarrow C_{\text{out},1} = G_{1:0} + P_{1:0} C_{\text{in},0}$$

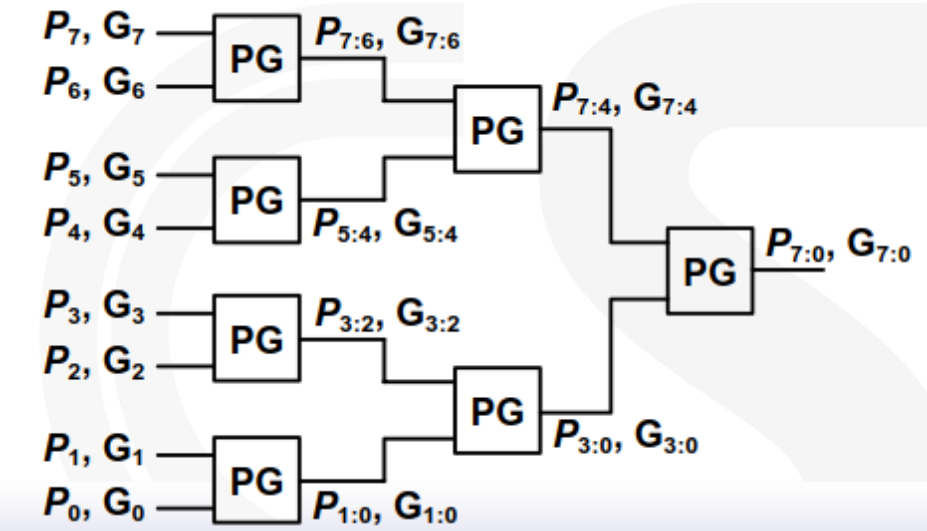
$$P_{3:2} = P_3 \cdot P_2 \qquad G_{3:2} = G_3 + P_3 \cdot G_2$$

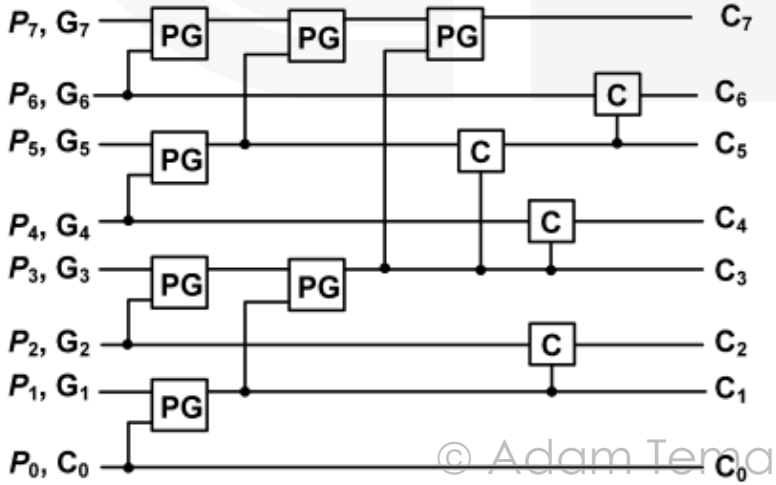
$$\Rightarrow C_{\text{out},3} = G_{3:2} + P_{3:2} C_{in,2}$$

$$P_{3:0} = P_{3:2} \cdot P_{1:0} \qquad G_{3:0} = G_{3:2} + P_{3:2} \cdot G_{1:0}$$

$$\Rightarrow C_{\text{out},3} = G_{3:0} + P_{3:0} C_{\text{in},0}$$

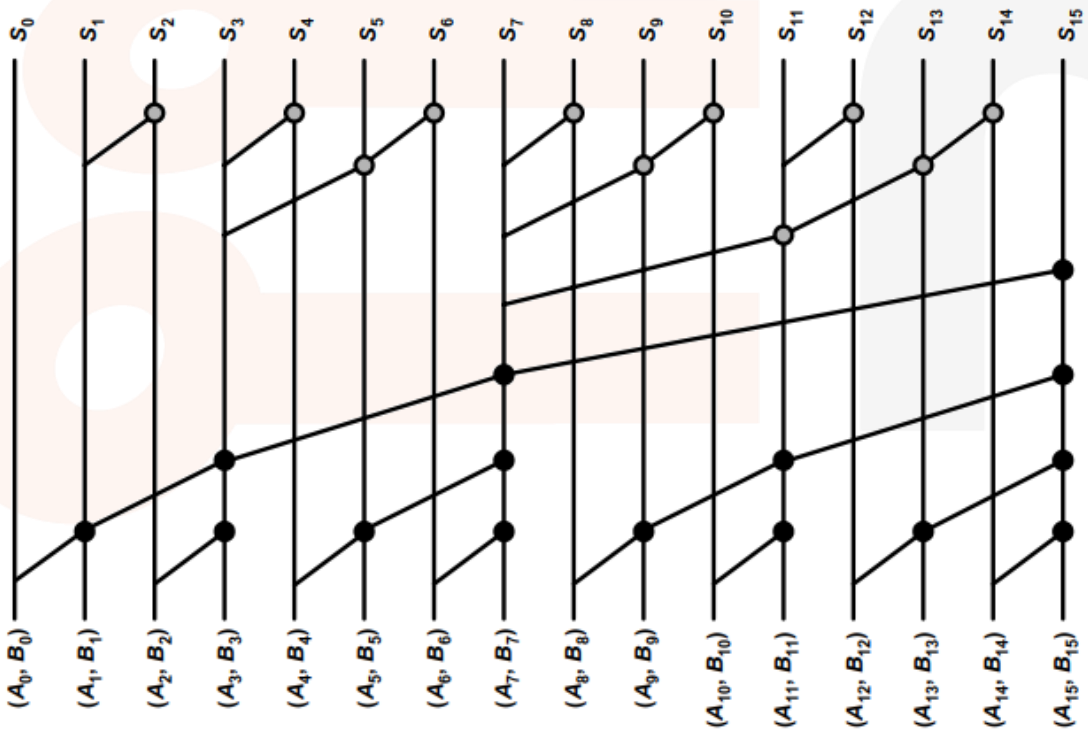
$$t_{\text{tree}} = t_{\text{p/g}} + \lceil \log_2 N \rceil t_{\text{AND/OR}} + t_{\text{sum}} \Rightarrow O(\log_2 N)$$



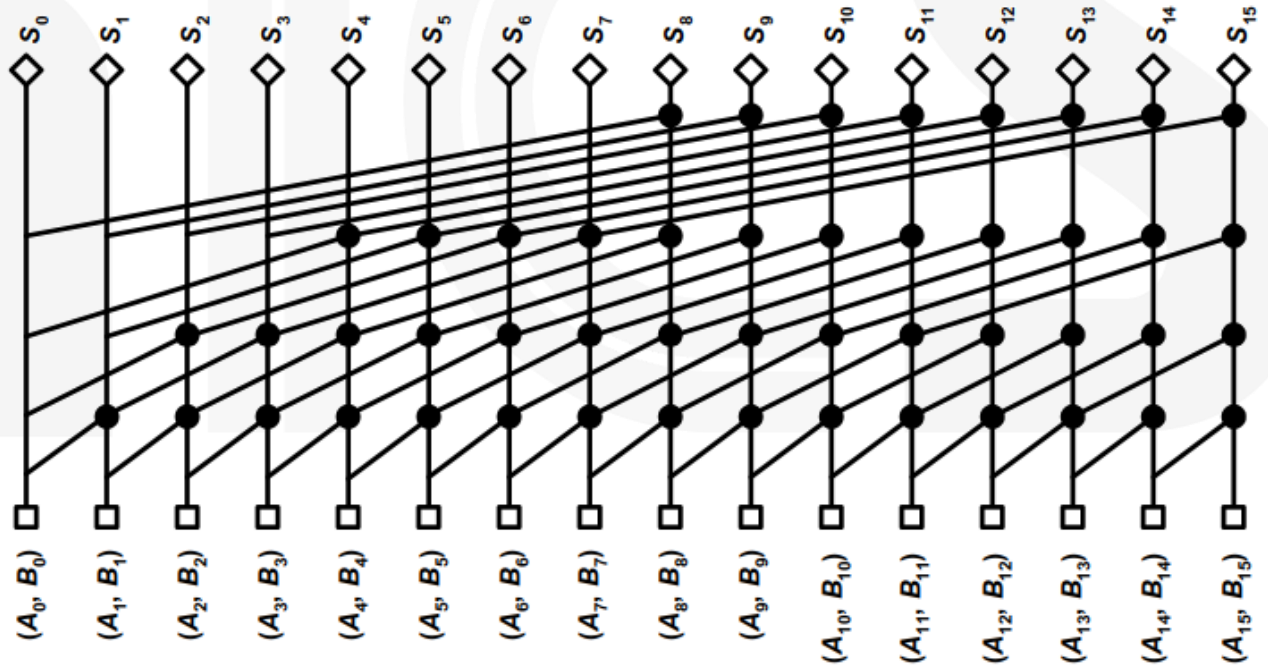


Tree Adders (Logarithmic CLA)

- **Many ways to construct these CLA or tree adders, based on:**
  - Radix: How many bits combined in each gate
  - Tree Depth: How many stages of logic to the final carry (>=$\log_{radix} N$)
  - Fanout: Maximal logic branching in tree

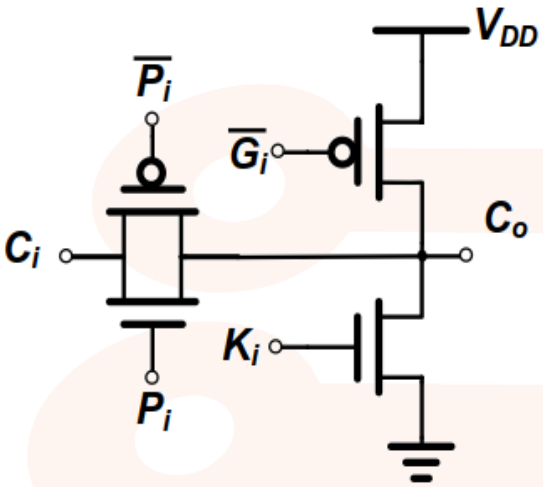


Brent-Kung Tree

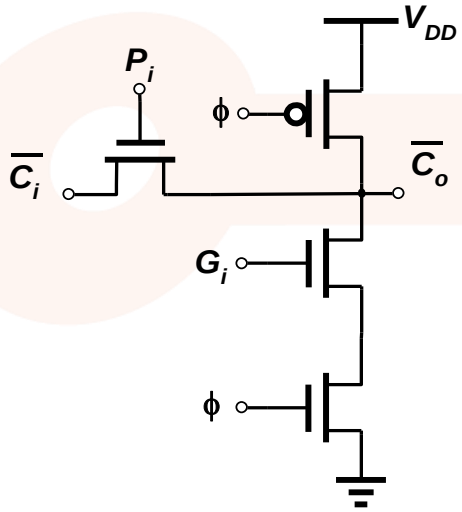


16-bit radix-2 Kogge-Stone tree

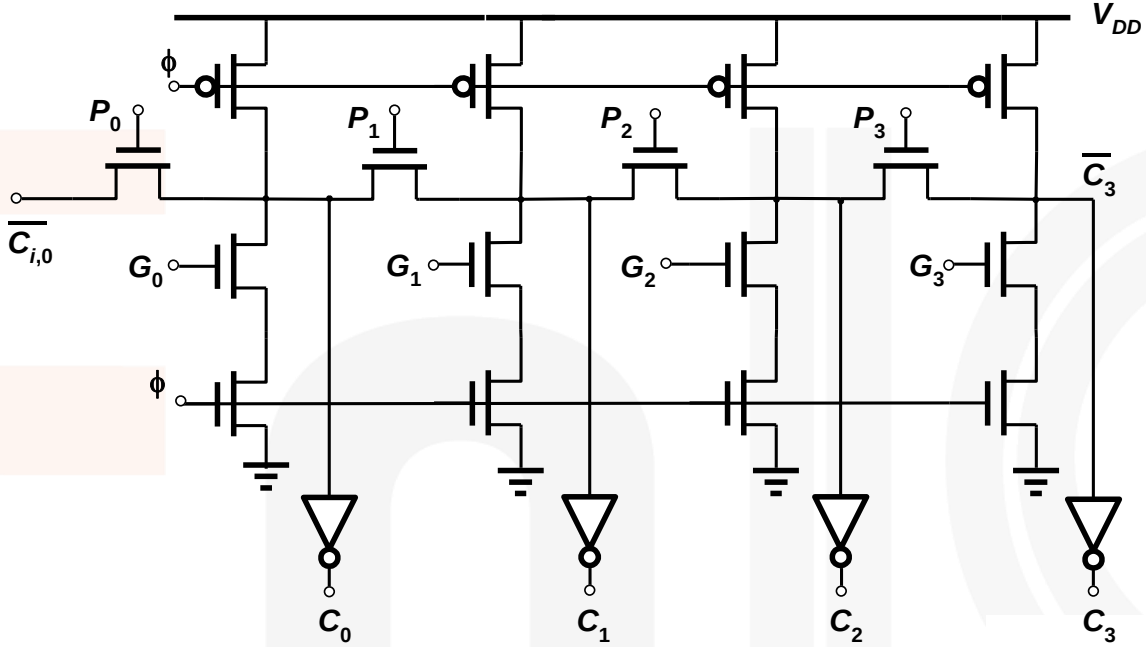
Manchester Carry-Chain Adder



Static Circuits



Dynamic Circuit

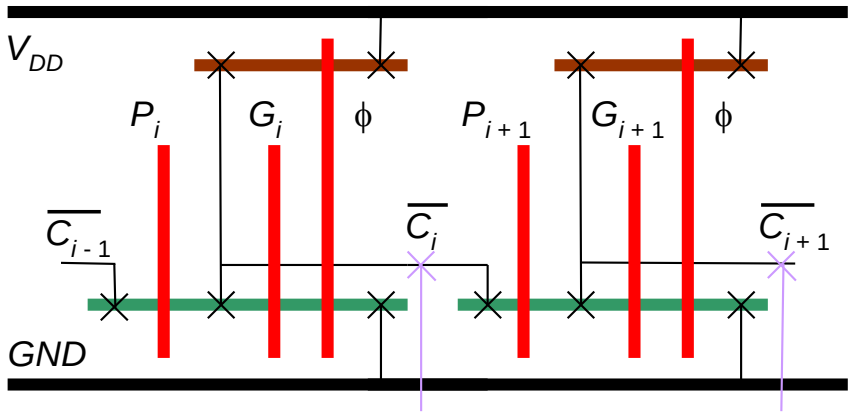


$$t_P = 0.69 \sum_{i=1}^{N} C_i \cdot \left(\sum_{j=1}^{i} R_j \right)$$

$$= 0.69 \frac{N(N+1)}{2} RC$$

where $R_j = R, \quad C_i = C$

Propagate/Generate Row



Inverter/Sum Row

The Computer Hall of Fame

Source: http://www.gondolin.org.uk

- **The home computer that 80s kids learned how to play games on and program with:**

The Commodore 64

- Introduced in Dec. 1982 for $595. Continued selling until 1992!
- 8-bit, 1 MHz, 64KB RAM, 16KB ROM
- Ran BASIC as it's interface.
- The highest selling single computer model of all time.
- It has been compared to the Ford Model T for its role in bringing a new technology to middle-class households via creative and affordable mass-production.
- Considered the computer that provided the foundation for the development of open-source software (freeware)

Source: wikipedia

IF PERSONAL COMPUTERS ARE FOR EVERYBODY, HOW COME THEY'RE PRICED FOR NOBODY?

A personal computer is supposed to be a computer for persons. Not just wealthy persons. Or whiz-kid persons. Or privileged persons.

But person persons.

In other words, all the persons whom Apple, IBM, and Radio Shack seem to have forgotten about (including, most likely, you).

But that's okay. Because now you can get a high-powered home computer without taking out a second mortgage on your home.

It's the Commodore 64. We're not talking about a low-priced computer that can barely retain a phone number. We're talking about a memory of 64K. Which means it can perform tasks most other home computers can't. Including some of those that cost a lot more. (Take another look at the three computers above.)

By itself, the Commodore 64 is all the computer you'll ever need. Yet, if you do want to expand its capabilities some day, you can do so by adding a full complement of Commodore peripherals. Such as disk drives. Modems. And printers.

You can also play terrific games on the Commodore 64. Many of which will be far more challenging than those you could ever play on a game machine alone.

And as great as all this sounds, what's even greater-sounding is the price. It's hundreds of dollars less than that of our nearest competitor.

So while other companies are trying to take advantage of the computer revolution, it seems to us they're really taking advantage of something else:

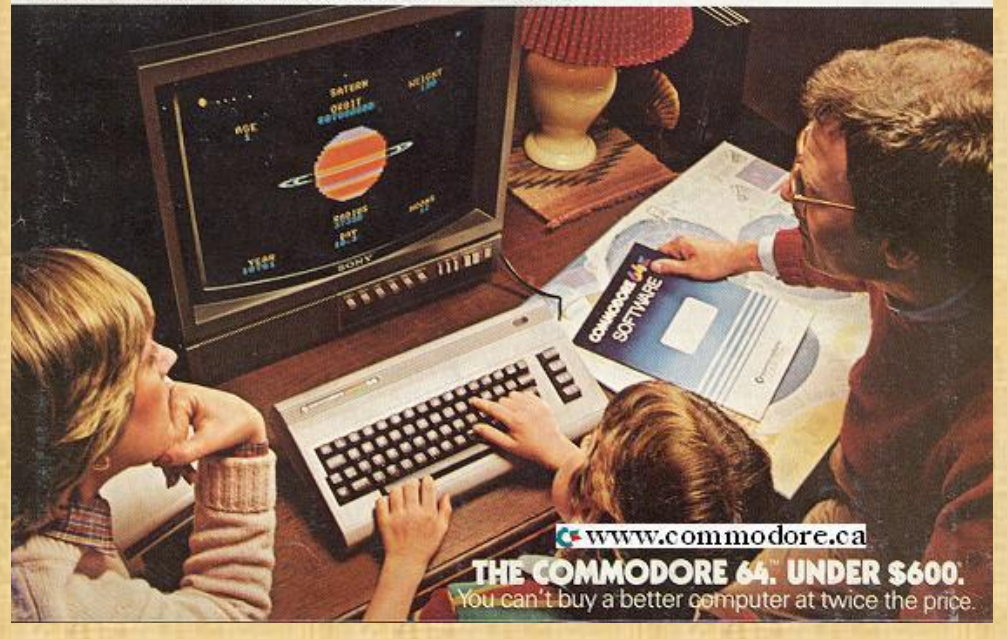
Their customers.

$1395*	$999*	$1355*
APPLE® IIe 64K	TRS-80® III 16K	IBM® PC 64K

*Manufacturers' suggested list prices as of March 20, 1983. Monitor included with TRS-80 III only. Commodore Business Machines, P.O. Box 500R, Conshohocken, PA 19428. Canada--3370 Pharmacy Avenue, Agincourt, Ont., Can. M1W 2K4.

commodore COMPUTER

Apple is a registered trademark of Apple Computer, Inc.
TRS-80 is a registered trademark of Tandy Corp. IBM is a registered trademark of International Business Machines Corp.



www.commodore.ca

THE COMMODORE 64. UNDER $600.

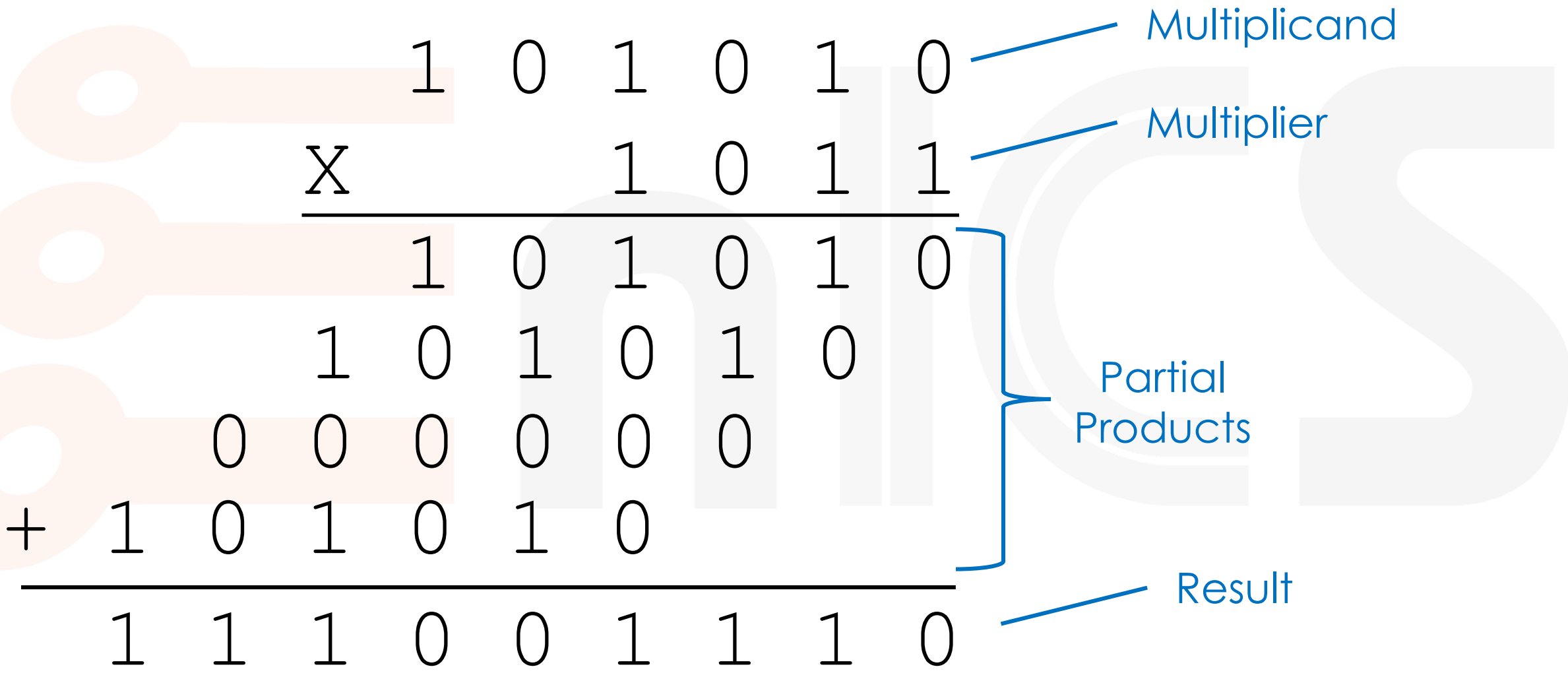
You can't buy a better computer at twice the price.

Basic Multiplication

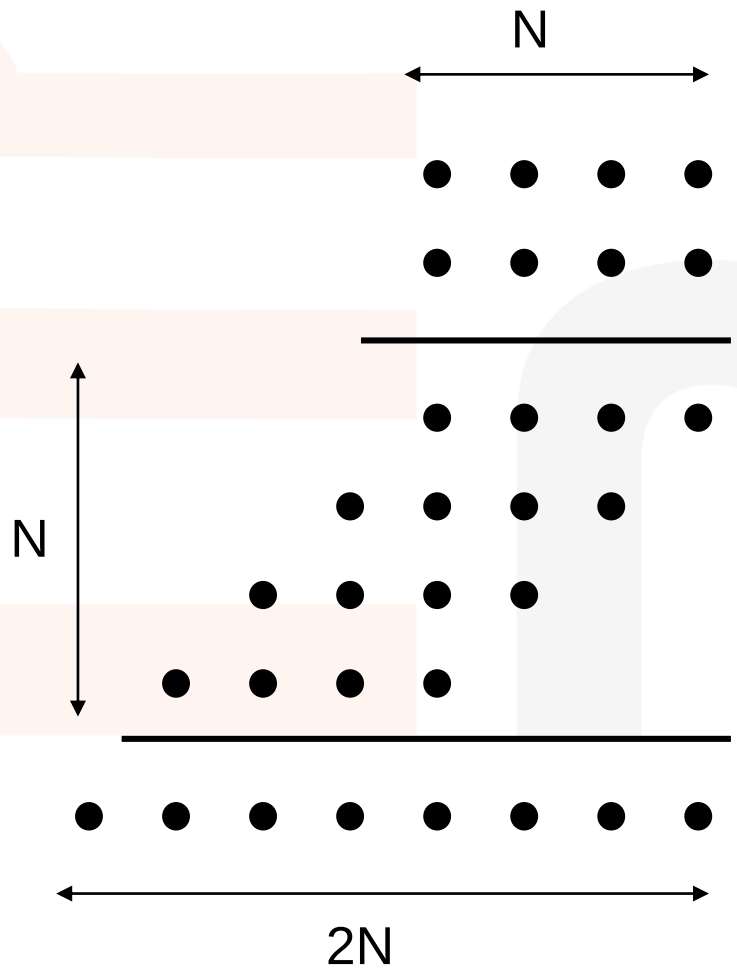
Grade School Multiplication

```
  1 2 3 4
X     1 2
-------
```

Multiplication using serial addition



Binary Multiplication



multiplicand

multiplier

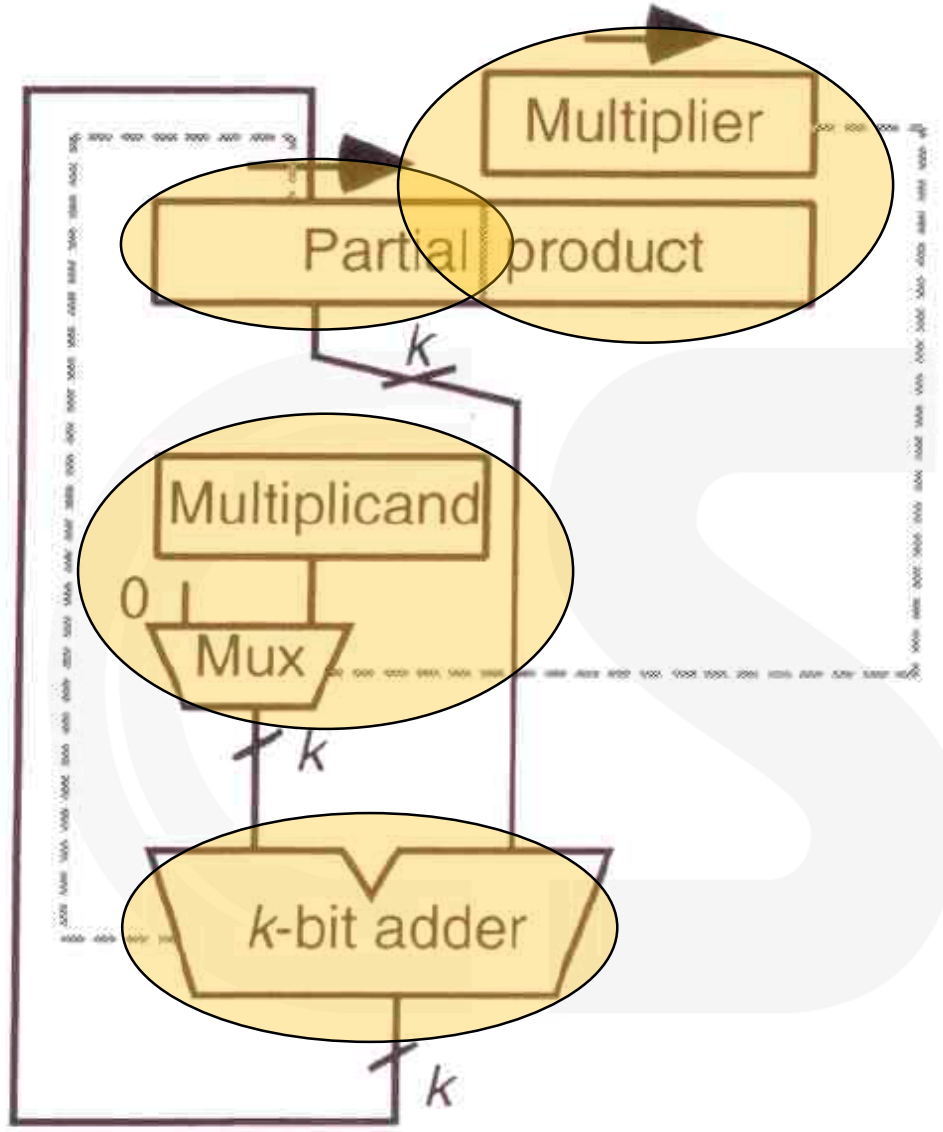
partial product array

can be formed in parallel

double precision product

Serial Shift and Add

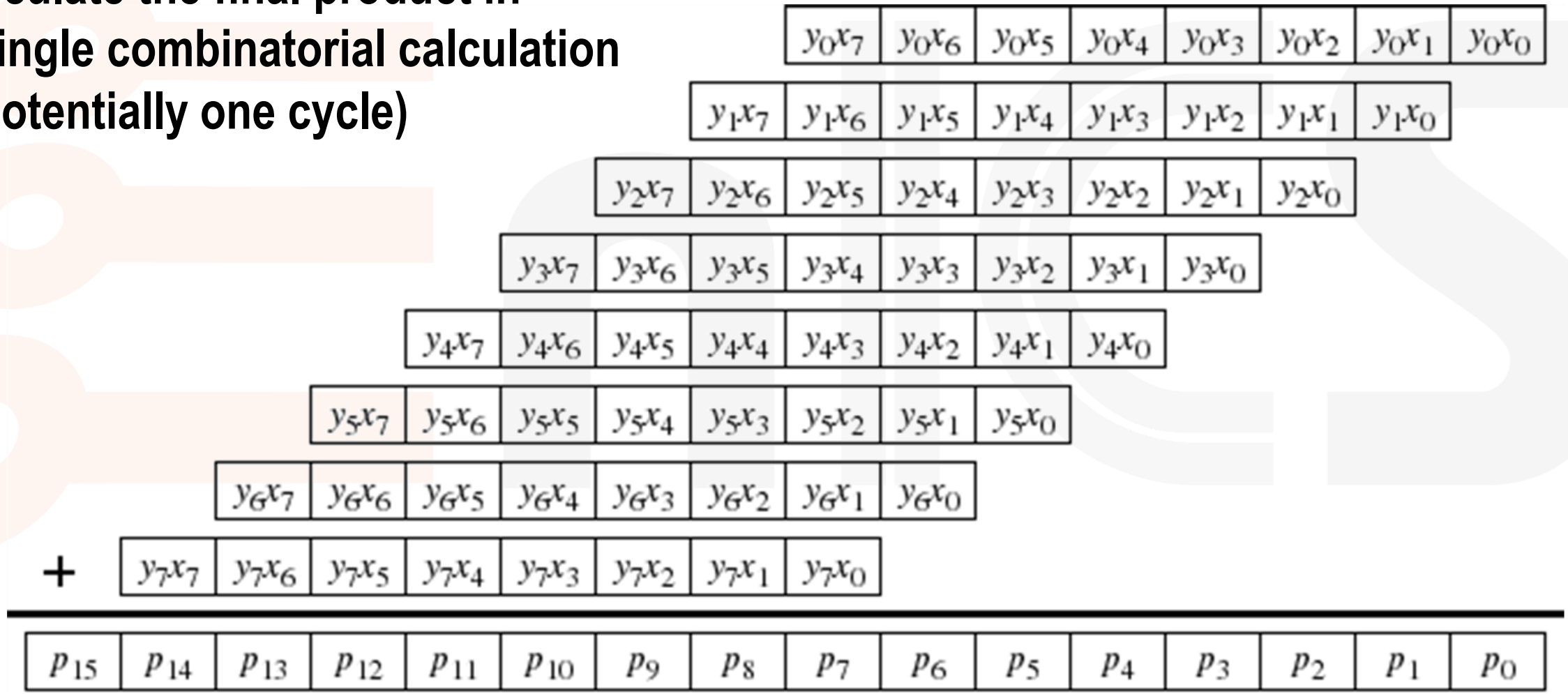
- **Concept:**
  - Multiplying by ‘1’ is copying the multiplicand
  - Multiplying by ‘0’ is a row of zeros
- **Select multiplicand or zeros according to multiplier bit**
- **Add to result**
- **Shift multiplier and accumulated result**



$$t_{serial} = O\left(N \cdot t_{\text{adder}}\right) = O\left(N^2\right)\Big|_{\text{for RCA}}$$

Array Multiplier

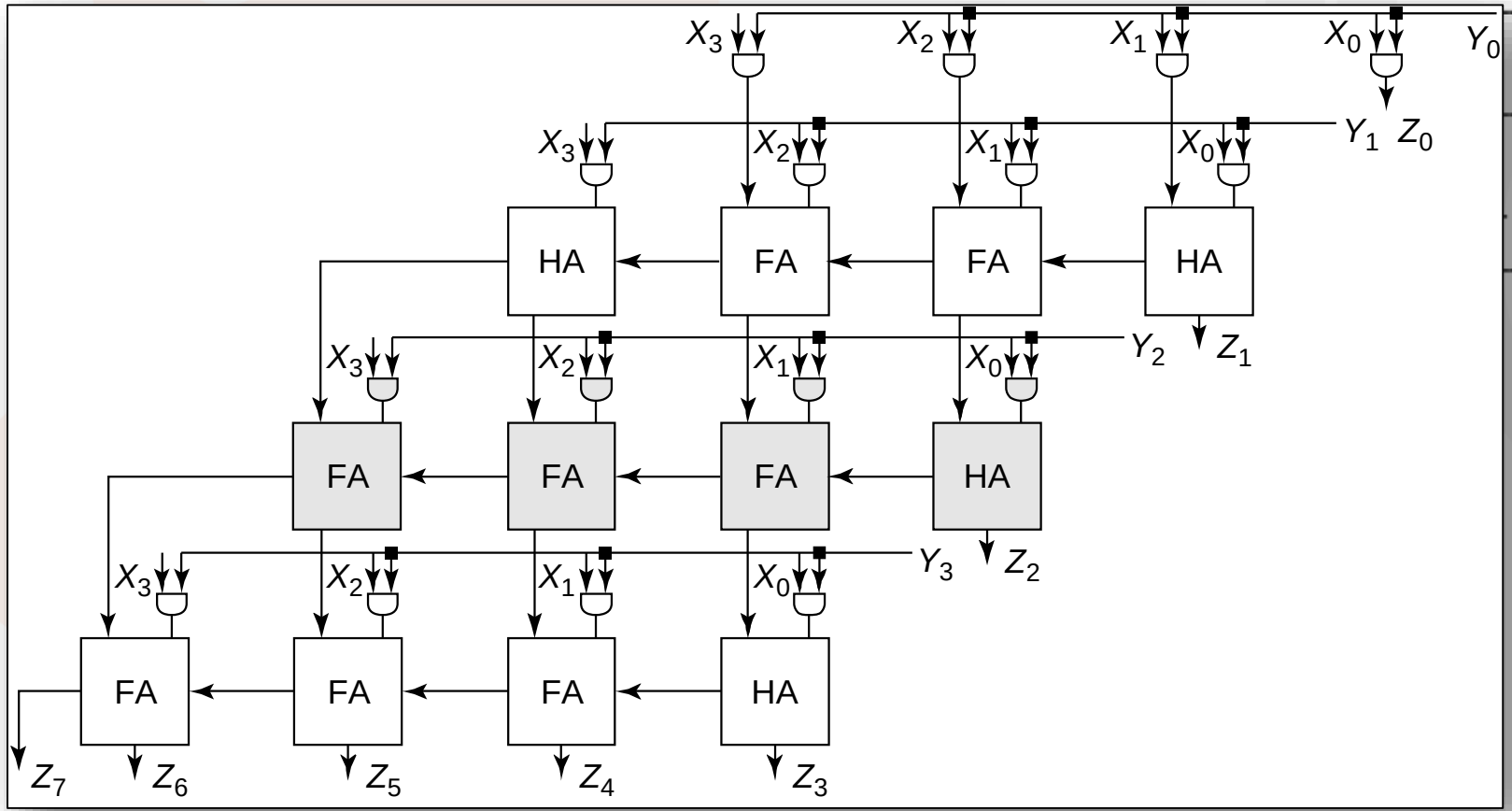
- **Calculate the final product in a single combinatorial calculation (=potentially one cycle)**



Array Multiplier Implementation

- **Stack 2-input Adders:**

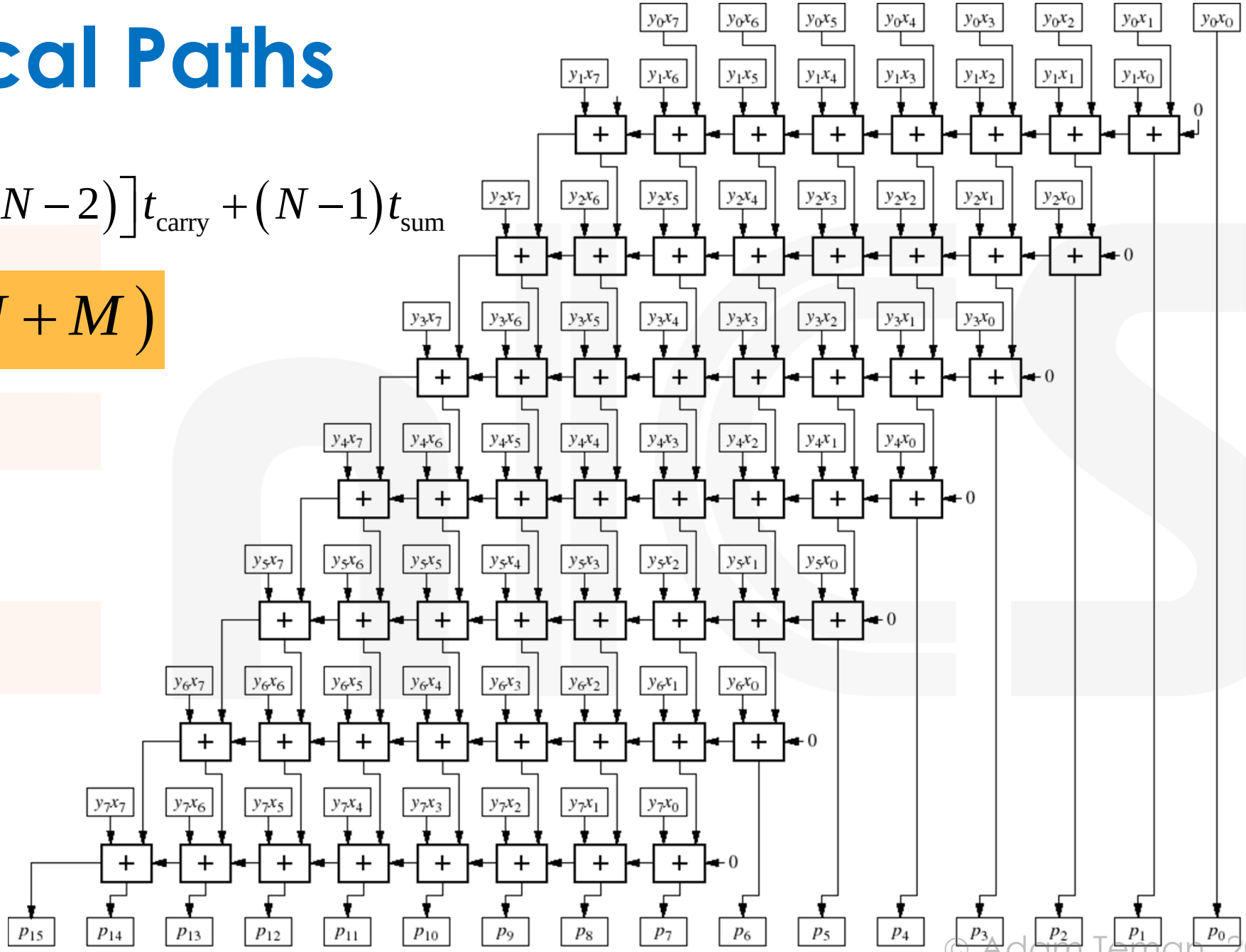
		A3	A2	A1	A0	Inputs
	x	B3	B2	B1	B0	
	C	B0 x A3	B0 x A2	B0 x A1	B0 x A0	
+	B1 x A3	B1 x A2	B1 x A1	B1 x A0		
			sum	sum		
			B2 x A0			Internal Signals
			sum			
			Y2	Y1	Y0	Outputs



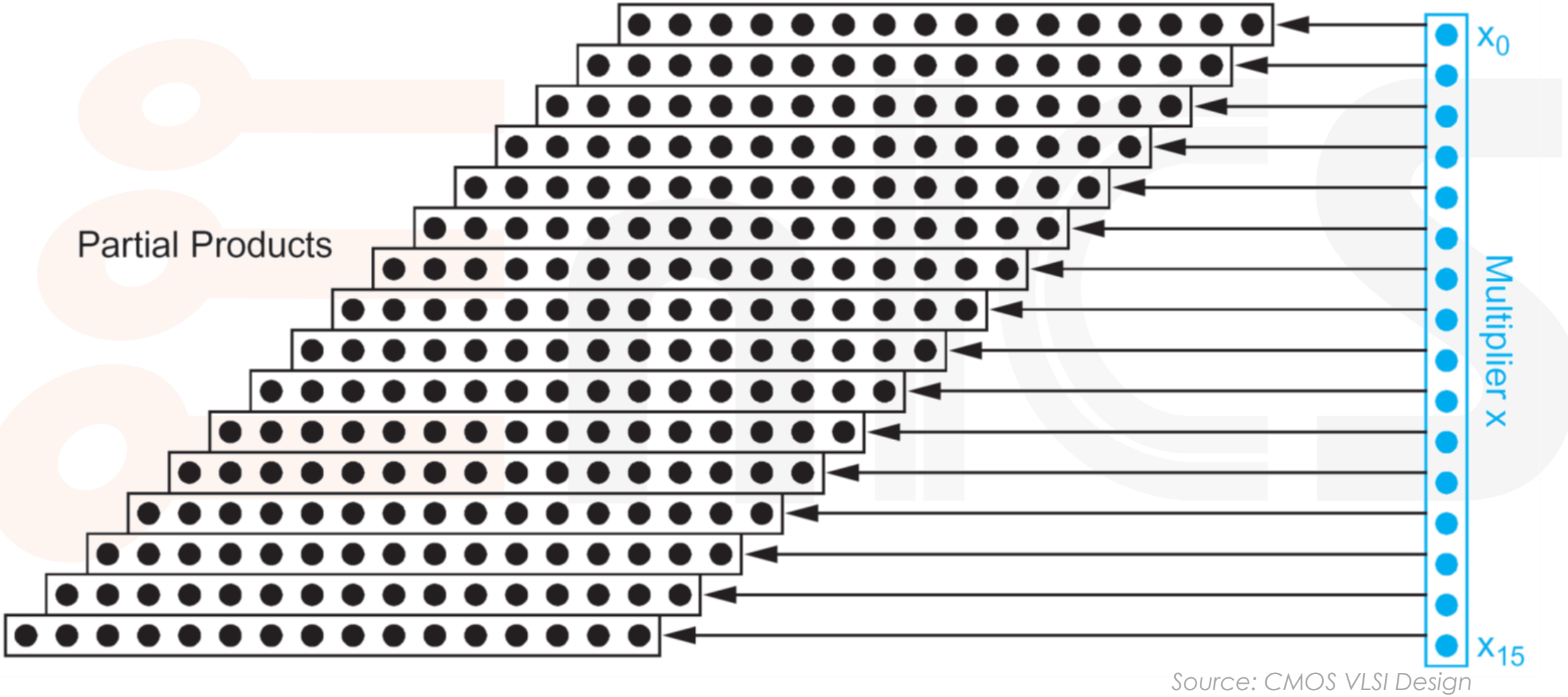
Many Critical Paths

$$t_{\text{mult}} \approx t_{\text{AND}} + \left[(M-1) + (N-2) \right] t_{\text{carry}} + (N-1) t_{\text{sum}}$$

$$\Rightarrow O(N+M)$$



Can we do it better?

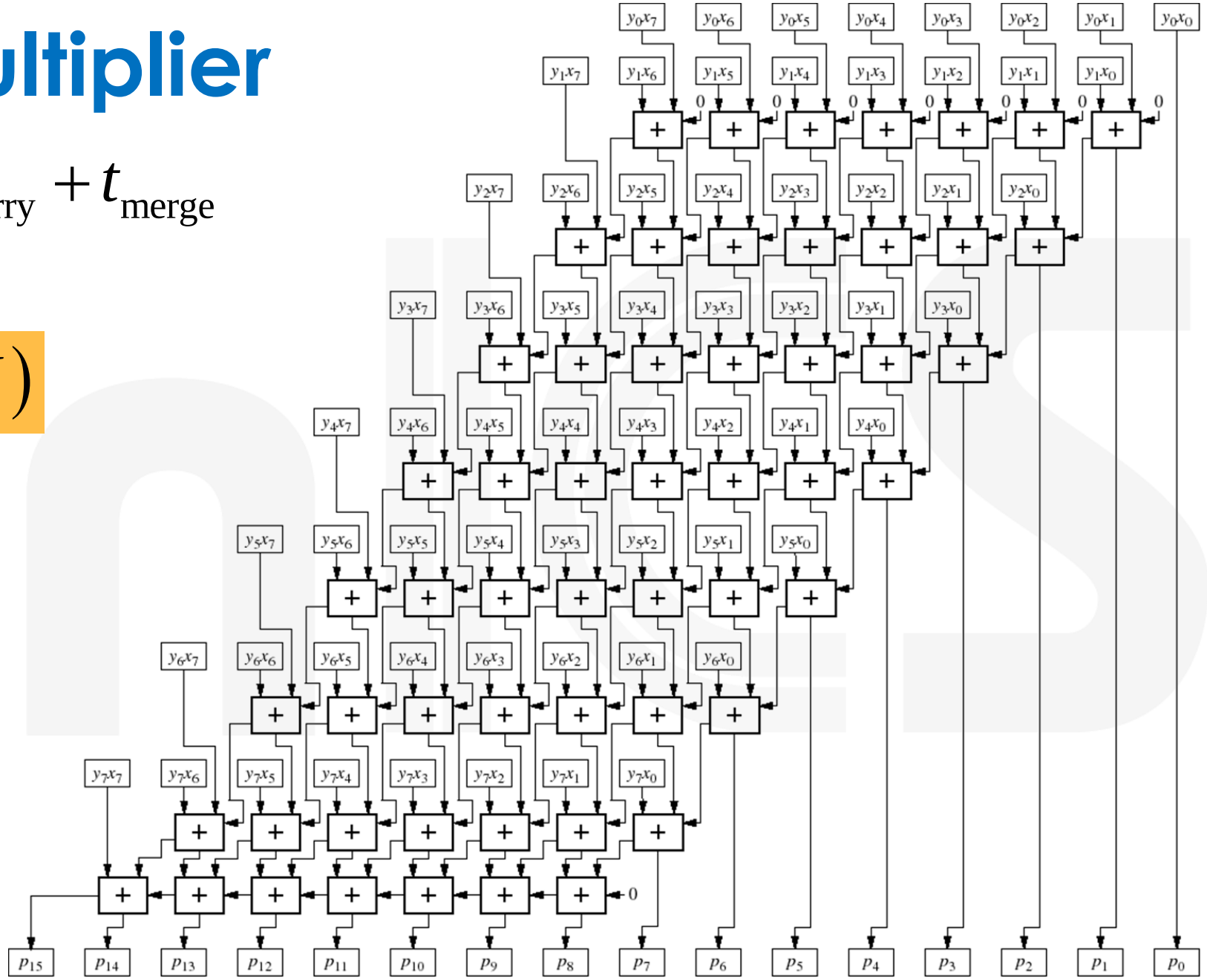


Source: CMOS VLSI Design

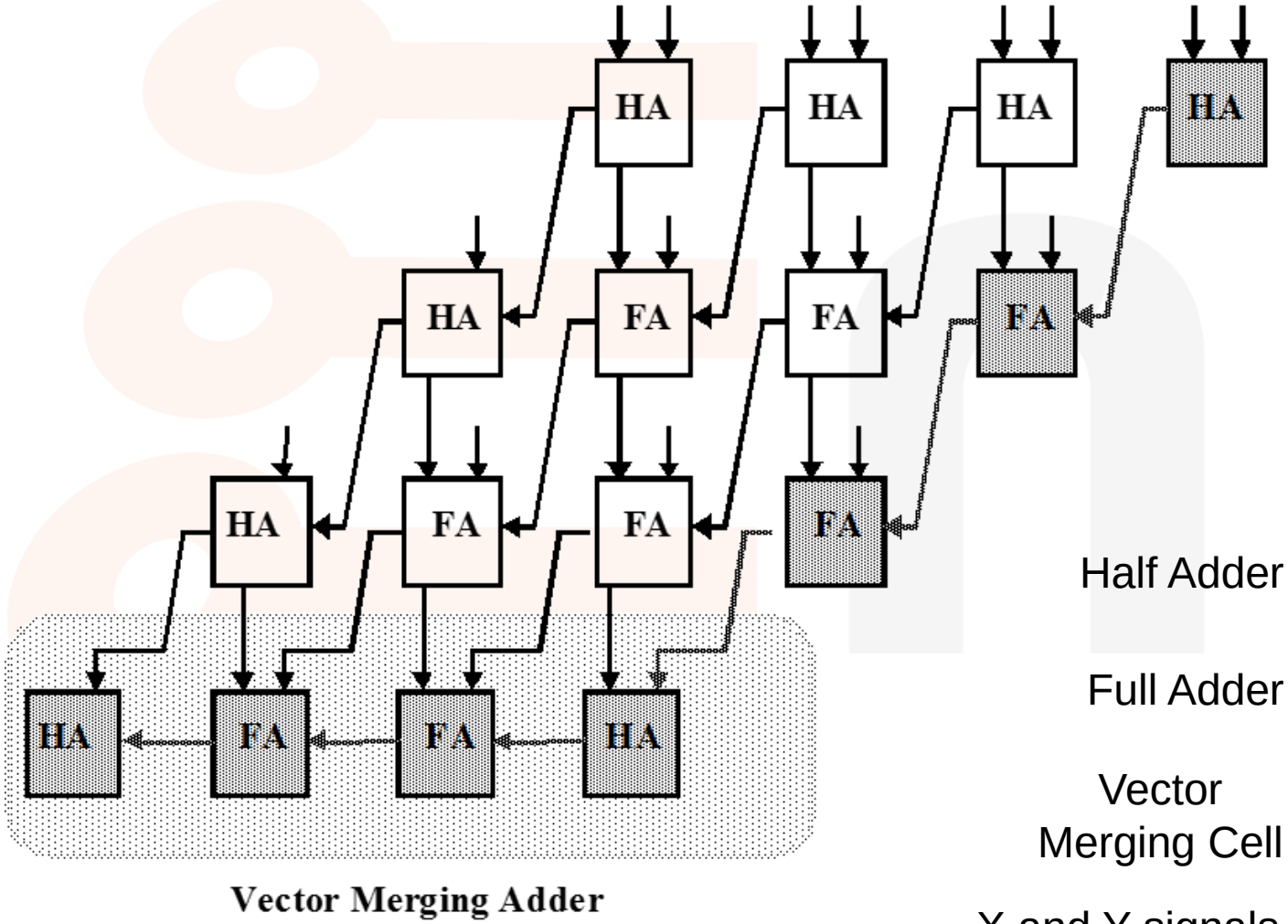
Carry-Save Multiplier

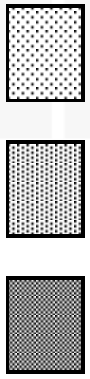
$$t_{\text{mult}} = t_{\text{AND}} + (N-1)t_{\text{carry}} + t_{\text{merge}}$$

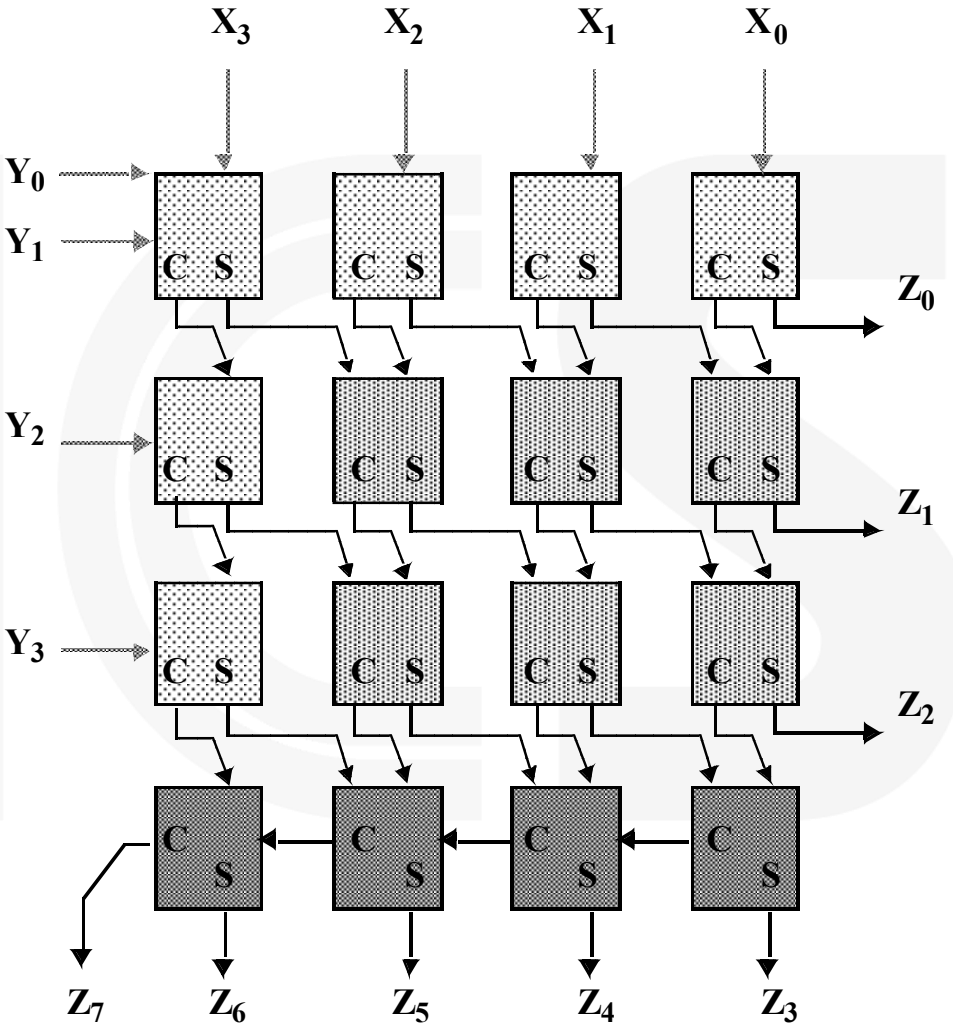
$$\Rightarrow O(N + \log_2 N)$$



Multiplier Floorplan







X and Y signals are broadcast through the complete array

Faster Multipliers

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Booth Recoding

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

- **Multiplying by '0' is redundant.**
- **Can we reduce the number of partial products?**
- **Based on the observation that**
  - We can turn sequences of 1's into sequences of 0's. For example: 0111=1000-0001
- **So we can introduce a '-1' bit and recode the multiplier:**
  - For example, the number 56

$$\begin{array}{rr} 1000 & (8) \\ -0001 & (1) \\ \hline 0111 & (7) \end{array} \qquad \begin{array}{rr} 01000000 & (64) \\ -00001000 & (8) \\ \hline 00111000 & (56) \end{array}$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	1	0	0	0
0	+1	0	0	−1	0	0	0

$$\begin{aligned} 2^5 + 2^4 + 2^3 &= 32 + 16 + 8 \\ &= 56 \end{aligned}$$

$$\begin{aligned} 2^6 - 2^3 &= 64 - 8 \\ &= 56 \end{aligned}$$

Radix-2 Booth Recoding

- **Parse multiplier from left to right**
  - For each change from 0 to 1, encode a '1'
  - For each change from 1 to 0, encode a '-1'
  - For bit 0, assume bit i=-1 is a 0
- **Example: 0011 0111 0011 = 0x373**

$$0\ \ 1\ \ 0\ \ -1 \mid 1\ \ 0\ \ 0\ \ -1 \mid 0\ \ 1\ \ 0\ \ -1$$

$$\begin{array}{cccc|cccc|cccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{r} 0\text{x}\ 484 \\ -0\text{x}\ 111 \\ \hline 0\text{x}\ 373 \end{array}$$

Modified (Radix-4) Booth Recoding

- **Radix-2 Booth Recoding doesn't work for parallel hardware implementations:**
  - A worst case (010101010101010) doesn't reduce the number of partial products.
  - Variable length recoders (according to the length of '1' strings) cannot be implemented efficiently.
- **Instead, just assume a constant length recoder.**
  - First apply standard booth recoding.
  - Next encode each pair of bits:

1. Within a sequence:

$$\begin{array}{cc} 0 & 0 \\ \hline & 0 \end{array}$$

2. Begin of a 1's-sequence:

$$\begin{array}{cc} 0 & +1 \\ \hline & +1 \end{array} \qquad \begin{array}{cc} +1 & -1 \\ \hline & +1 \end{array} \qquad \begin{array}{cc} +1 & 0 \\ \hline & +2 \end{array}$$

3. End of a 1's-sequence:

$$\begin{array}{cc} 0 & -1 \\ \hline & -1 \end{array} \qquad \begin{array}{cc} -1 & +1 \\ \hline & -1 \end{array} \qquad \begin{array}{cc} -1 & 0 \\ \hline & -2 \end{array}$$

- **This can be summarized in a truth table:**

Partial Product Selection Table	
Multiplier Bits	**Recorded Bits**
000	0
001	+ Multiplicand
010	+ Multiplicand
011	+2 × Multiplicand
100	-2 × multiplicand
101	- Multiplicand
110	- Multiplicand
111	0

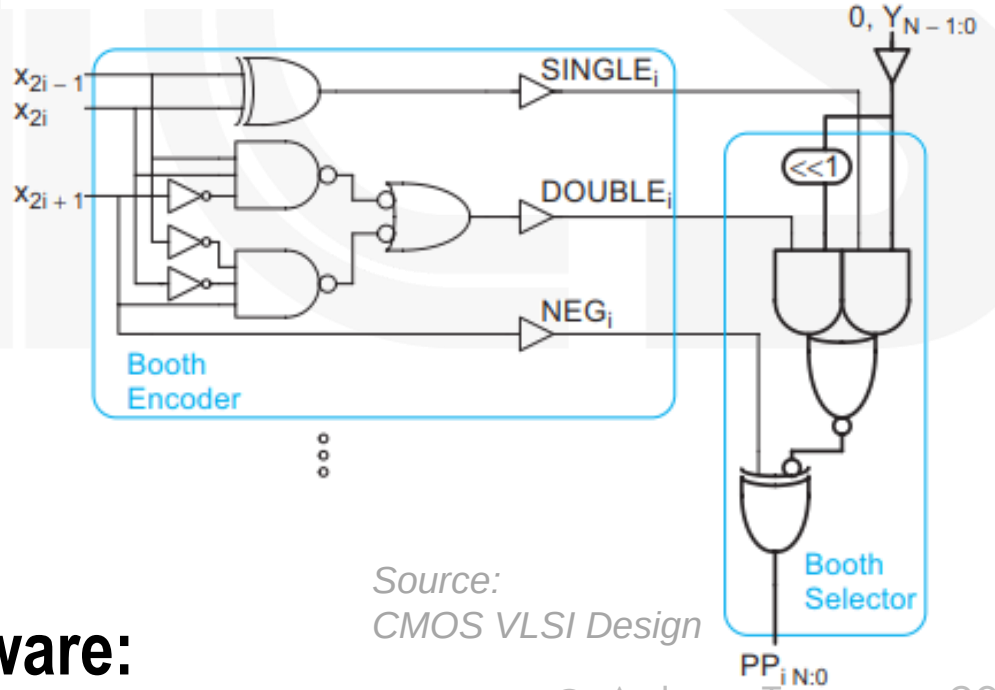
Modified (Radix-4) Booth Recoding

- **For example, let's take our previous example:**
  - 0011 0111 0011 = 01 0-1 10 0-1 01 0-1
  - This comes out: 1 -1 2 -1 1 -1.
- **We could have done this by using the table:**

Partial Product Selection Table	
Multiplier Bits	**Recorded Bits**
000	0
001	+ Multiplicand
010	+ Multiplicand
011	+2 × Multiplicand
100	-2 × multiplicand
101	- Multiplicand
110	- Multiplicand
111	0

0 0 1 1 0 1 1 1 0 0 1 1

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_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

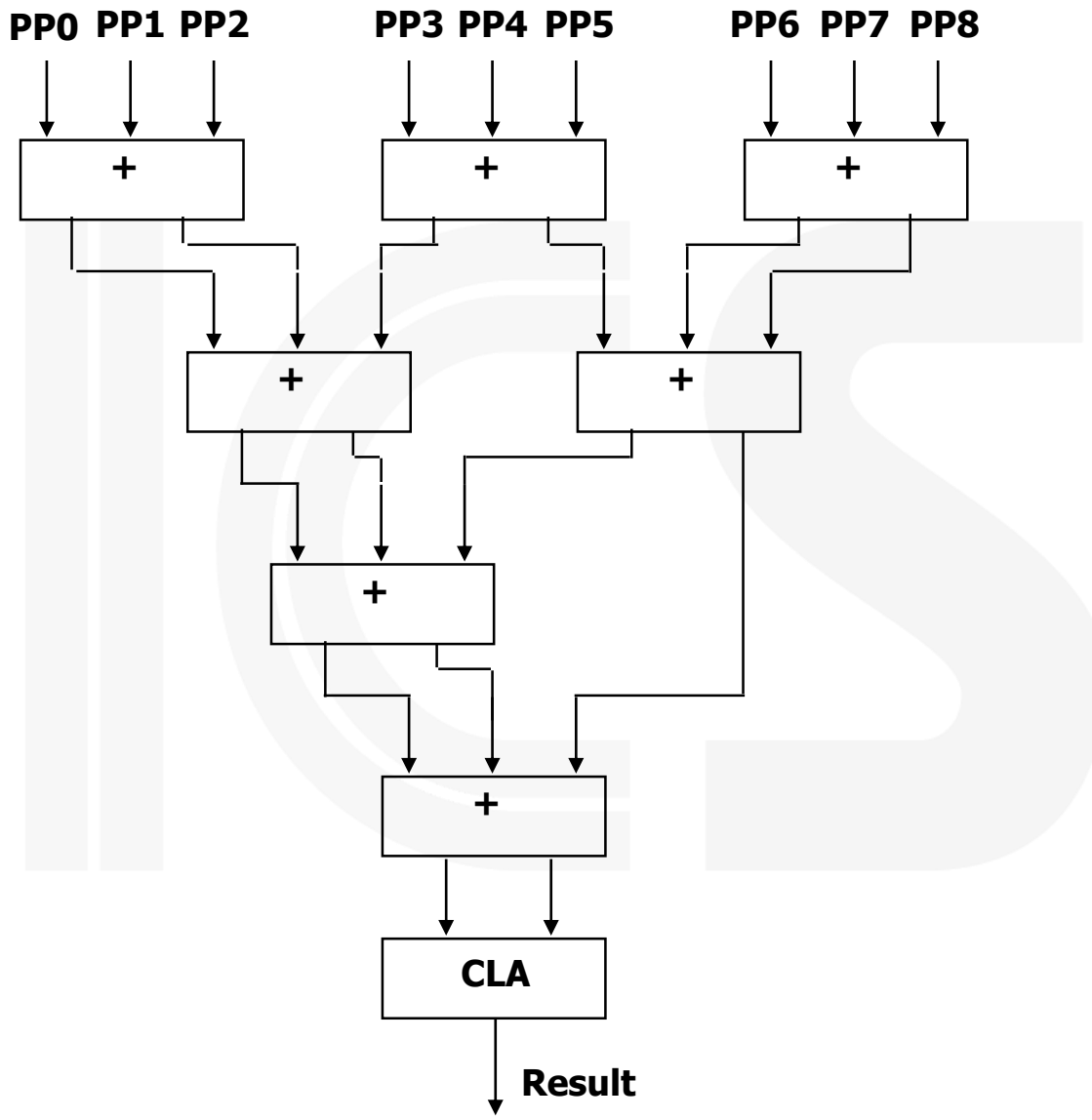


Source: CMOS VLSI Design

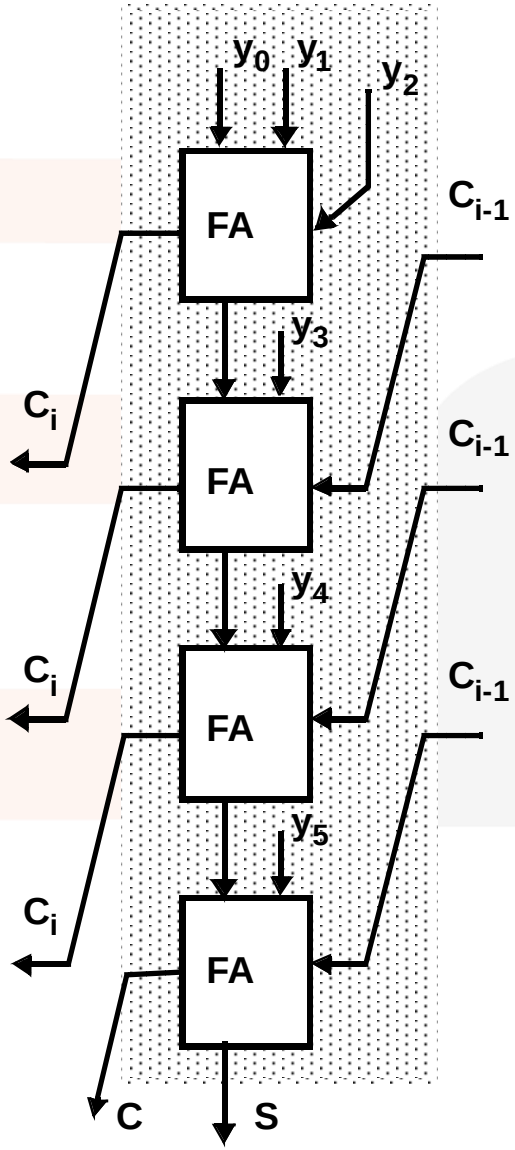
- **To implement this we need pretty simple hardware:**

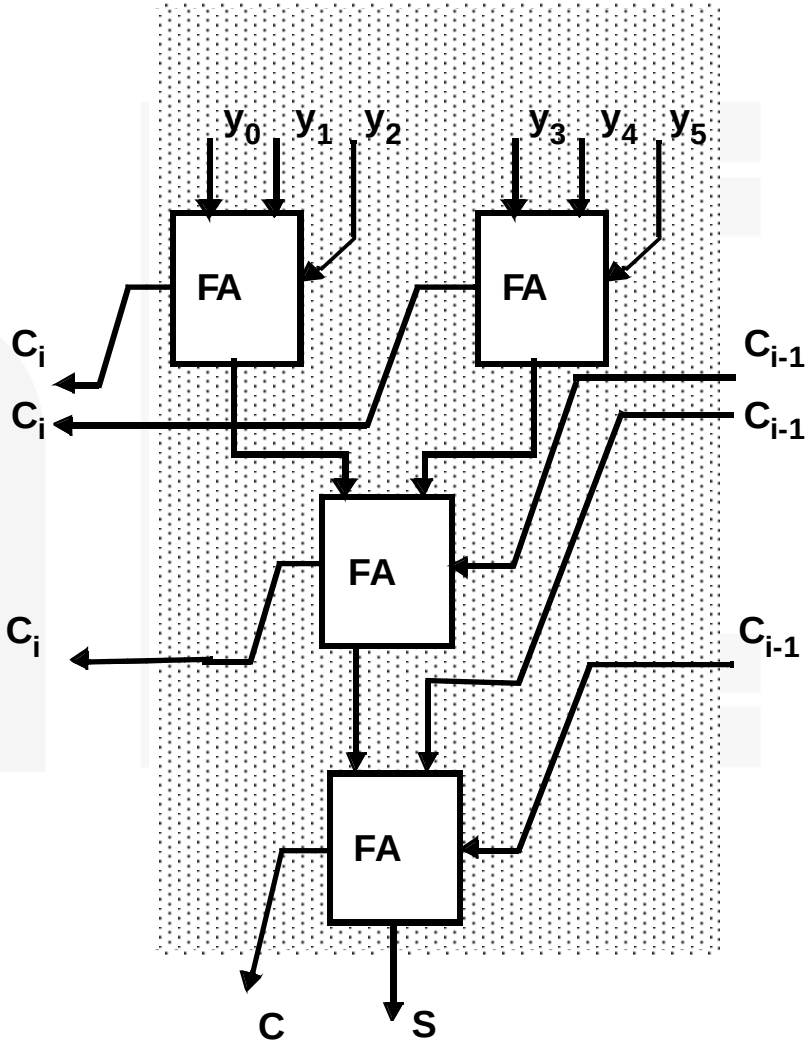
Tree Multipliers

- Can we further reduce the multiplier delay by employing logarithmic (tree) structures?



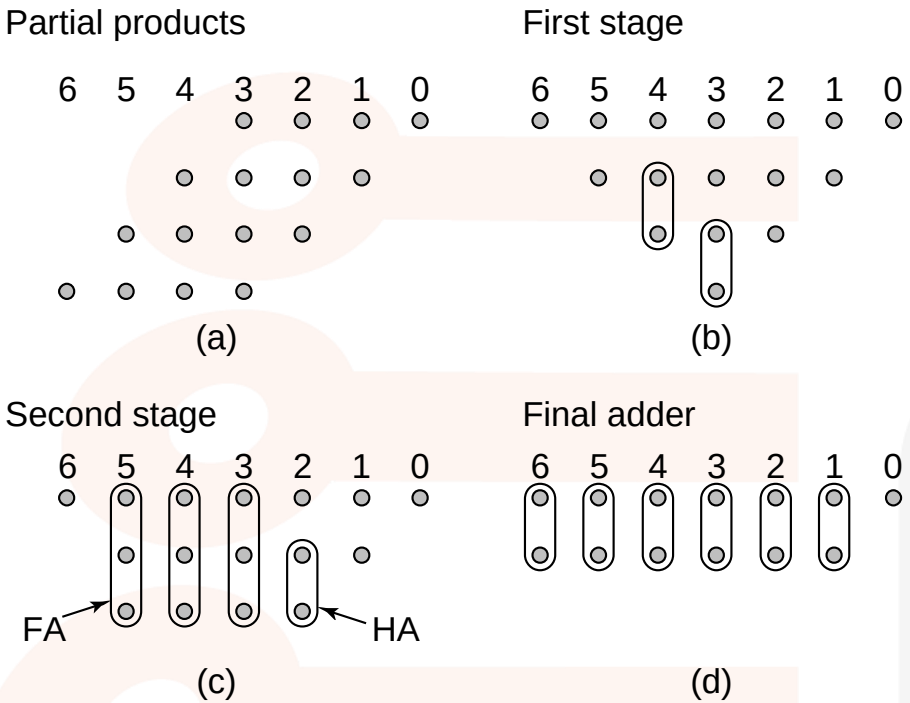
Wallace-Tree Multiplier

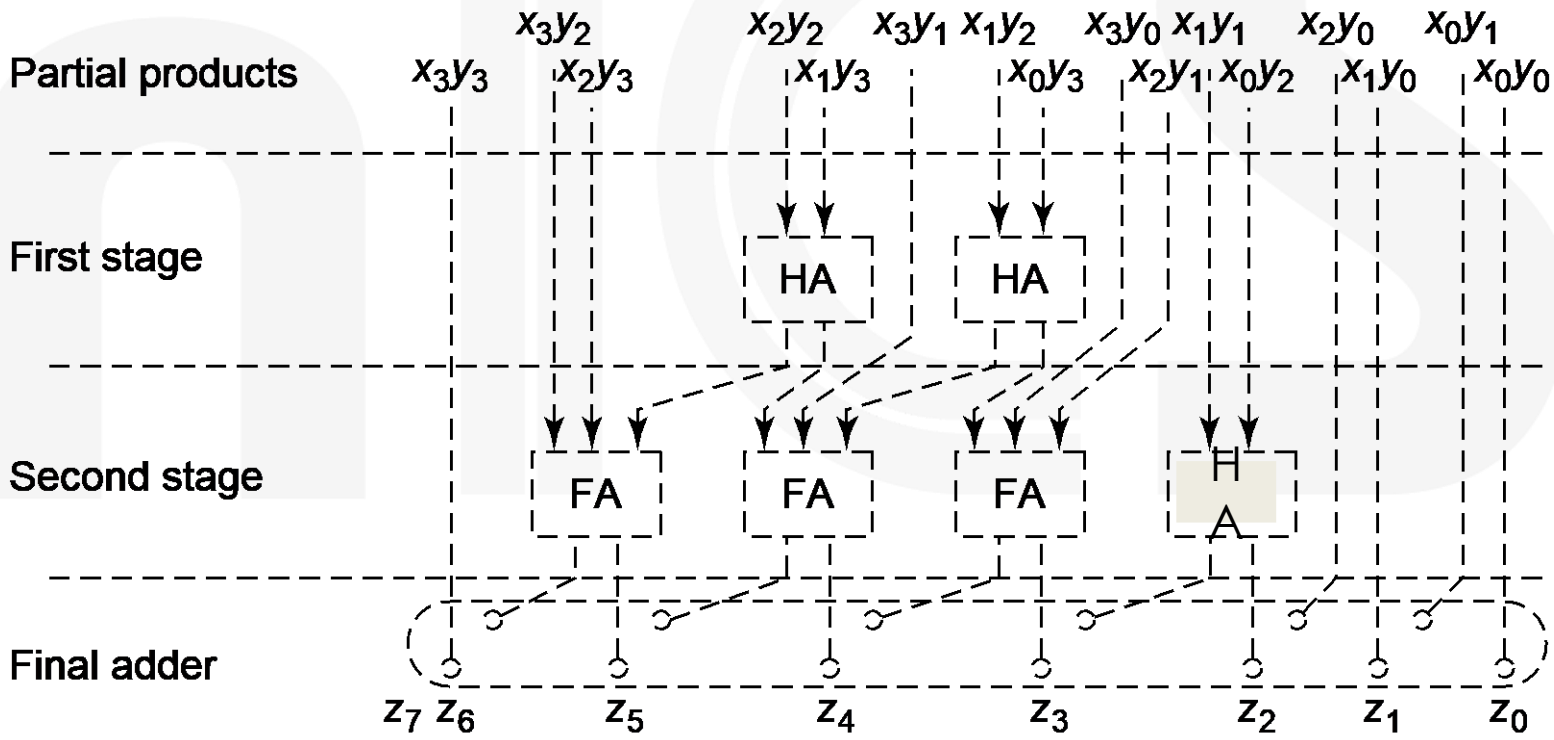




Wallace-Tree Multiplier

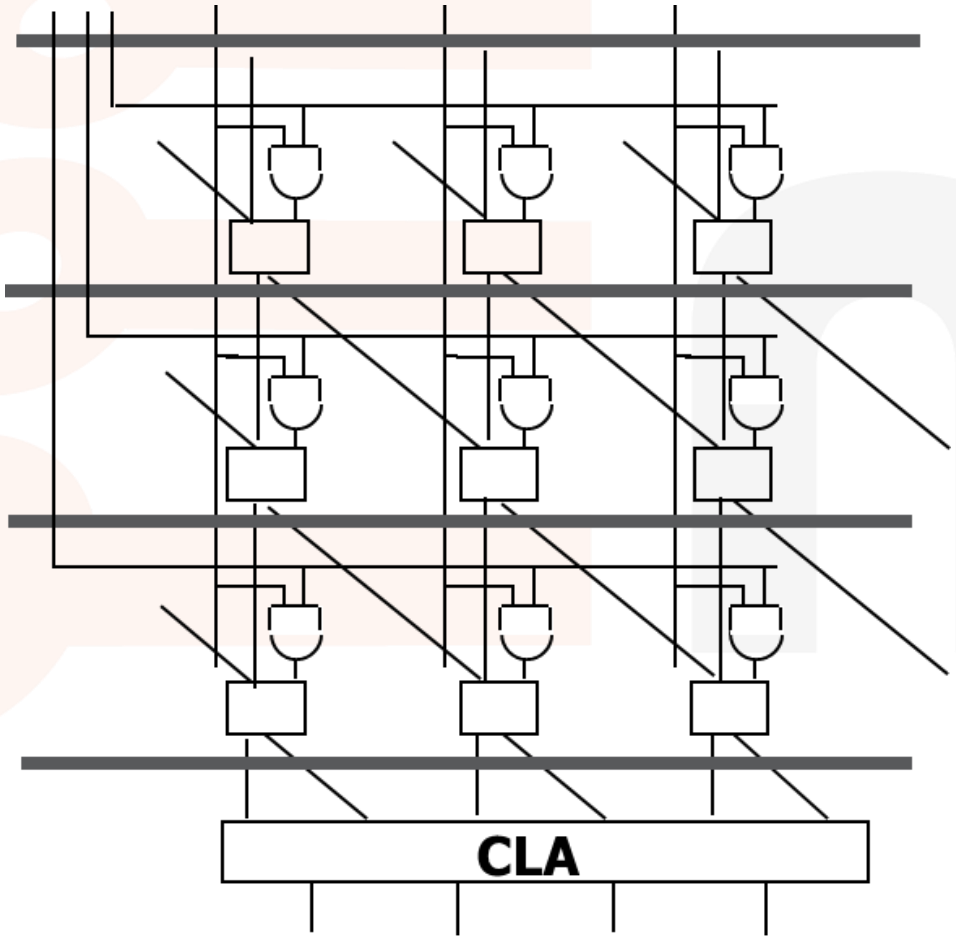
Wallace-Tree Multiplier

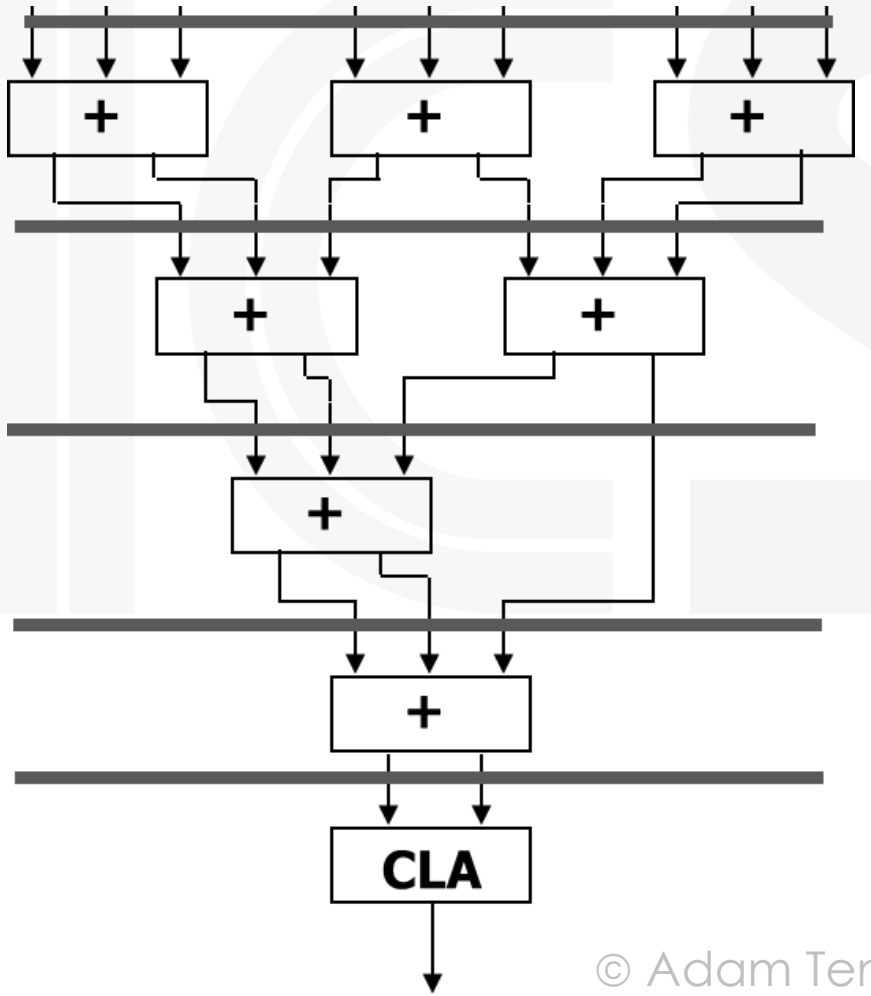




Pipelining Multipliers

- **Pipelining can be applied to most multiplier structures:**





Further Reading

- Rabaey, et al. “Digital Integrated Circuits” (2nd Edition)
- Elad Alon, Berkeley ee141 (online)
- Weste, Harris, “CMOS VLSI Design (4th Edition)”