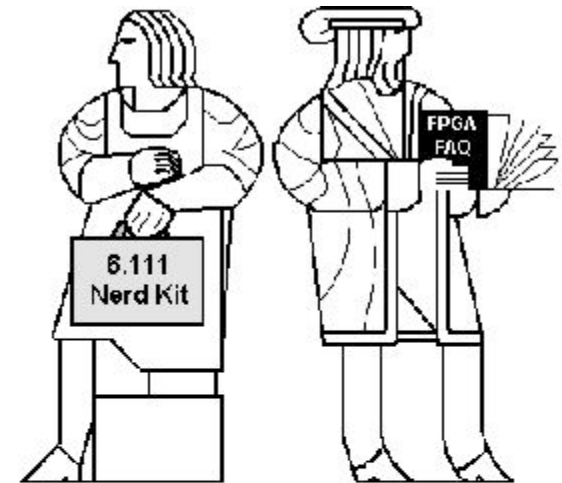


6.111 Lecture 12

Today: Arithmetic: Addition & Subtraction

1. Binary representation
2. Addition and subtraction
3. Speed: Ripple-Carry
4. Carry-bypass adder
5. Carry-lookahead adder



Acknowledgements:

- R. Katz, “*Contemporary Logic Design*”, Addison Wesley Publishing Company, Reading, MA, 1993. (Chapter 5)
- J. Rabaey, A. Chandrakasan, B. Nikolic, “*Digital Integrated Circuits: A Design Perspective*” Prentice Hall, 2003.
- Kevin Atkinson, Alice Wang, Rex Min

1. Binary Representation of Numbers

How to represent negative numbers?

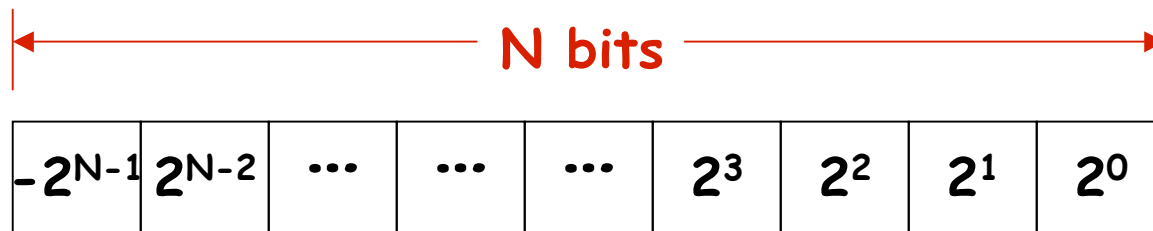
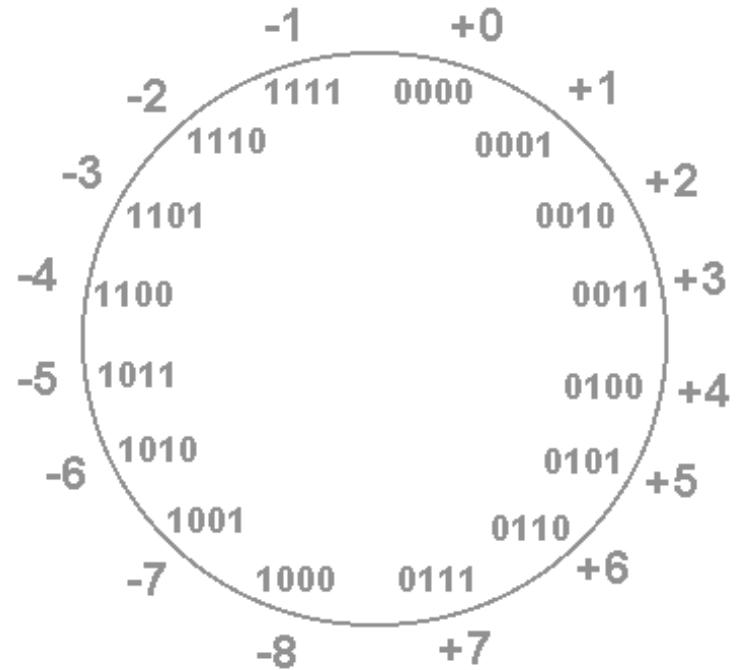
- Three common schemes:
 - sign-magnitude, ones complement, twos complement
- Sign-magnitude: MSB = 0 for positive, 1 for negative
 - Range: $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$
 - **Two representations** for zero: 0000... & 1000...
 - **Simple multiplication but complicated addition/subtraction**
- Ones complement: if N is positive then its negative is \bar{N}
 - Example: 0111 = 7, 1000 = -7
 - Range: $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$
 - **Two representations** for zero: 0000... & 1111...
 - Subtraction is addition followed by ones complement

Negative Numbers: Twos Complement

Twos complement = bitwise complement + 1

$0111 \rightarrow 1000 + 1 = 1001 = -7$
 $1001 \rightarrow 0110 + 1 = 0111 = +7$

- Asymmetric range
- Only one representation for zero
- Simple addition and subtraction
- Most common representation



“sign bit”

Range: -2^{N-1} to $2^{N-1} - 1$

“decimal” point

Twos Complement: Examples & Properties

- 4-bit examples:

4	0100	-4	1100	4	0100	-4	1100
<u>+ 3</u>	<u>0011</u>	+ (-3)	<u>1101</u>	<u>- 3</u>	<u>1101</u>	<u>+ 3</u>	<u>0011</u>
7	0111	-7	11001	1	10001	-1	1111

[Katz'93, chapter 5]

- 8-bit twos complement example:

$$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$$

- With twos complement representation for signed integers, the same binary addition procedure works for adding both signed and unsigned numbers.

- By moving the implicit location of "decimal" point, we can represent fractions too:

$$1101.0110 = -2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = -8 + 4 + 1 + 0.25 + 0.125 = -2.25$$

2. Binary Addition & Subtraction

Addition:

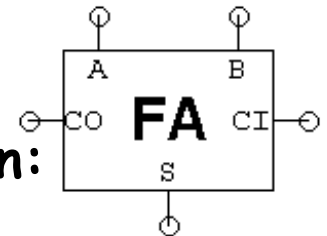
Here's an example of binary addition as one might do it by "hand":

Adding two N-bit numbers produces an (N+1)-bit result

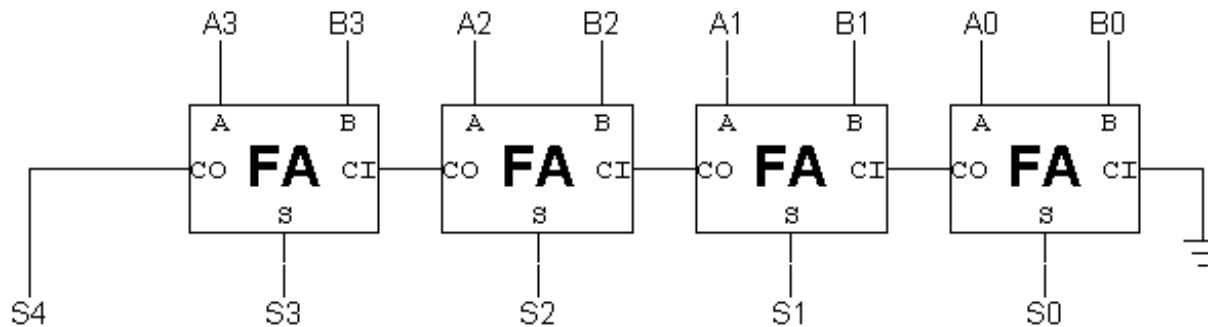
$$\begin{array}{r} 1101 \\ + 0101 \\ \hline 10010 \end{array}$$

Carries from previous column

We've already built the circuit that implements one column:



So we can quickly build a circuit to add two 4-bit numbers...



"Ripple-carry adder"

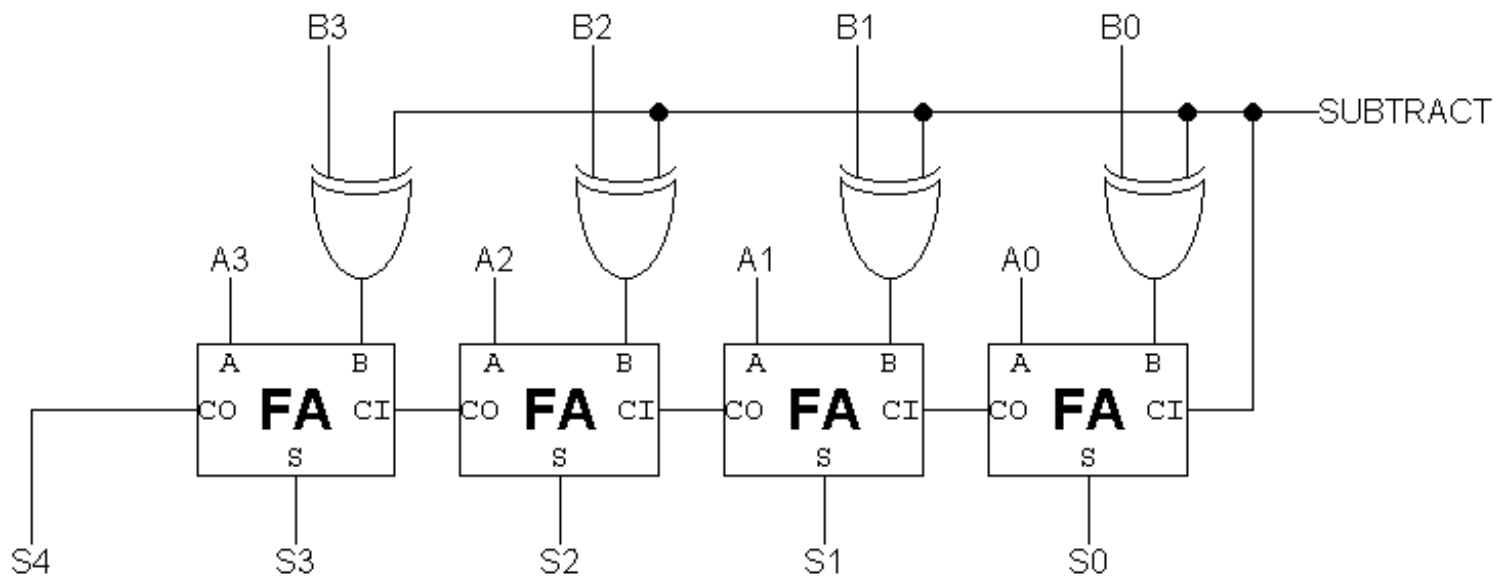
Subtraction: $A - B = A + (-B)$

Using 2's complement representation: $-B = \sim B + 1$

\sim = bit-wise complement



So let's build an arithmetic unit that does both addition and subtraction. Operation selected by *control input*:



Condition Codes

Besides the sum, one often wants four other bits of information from an arithmetic unit:

Z (zero): result is = 0

big NOR gate

N (negative): result is < 0

S_{N-1}

C (carry): indicates an add in the most significant position produced a carry, e.g., 1111 + 0001

from last FA

V (overflow): indicates that the answer has too many bits to be represented correctly by the result width, e.g., 0111 + 0111

$$V = A_{N-1}B_{N-1}\overline{S_{N-1}} + \overline{A_{N-1}}\overline{B_{N-1}}S_{N-1}$$

$$V = COUT_{N-1} \oplus CIN_{N-1}$$

To compare A and B, perform A-B and use condition codes:

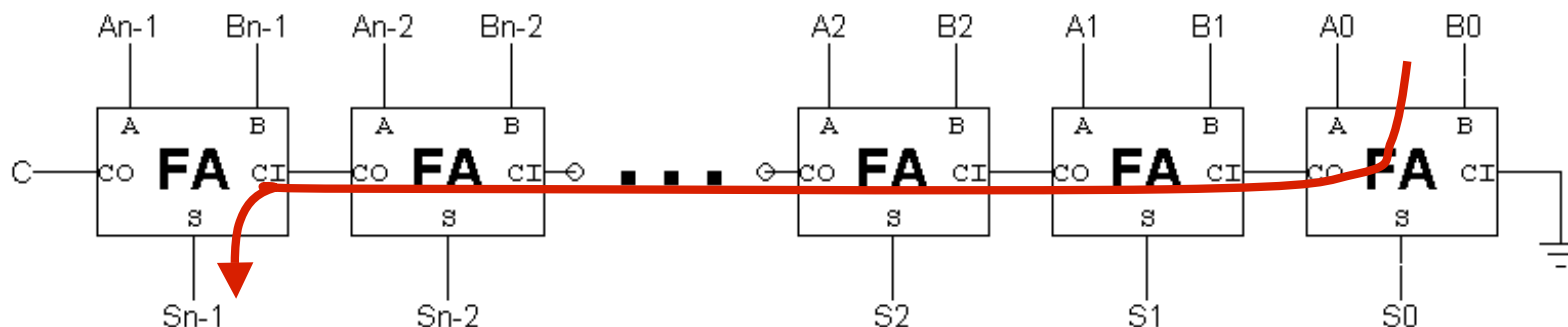
Signed comparison:

LT	$N \oplus V$
LE	$Z + (N \oplus V)$
EQ	Z
NE	$\sim Z$
GE	$\sim (N \oplus V)$
GT	$\sim (Z + (N \oplus V))$

Unsigned comparison:

LTU	C
LEU	$C + Z$
GEU	$\sim C$
GTU	$\sim (C + Z)$

3. Speed: t_{PD} of Ripple-carry Adder



Worse-case path: carry propagation from LSB to MSB, e.g., when adding 11...111 to 00...001.

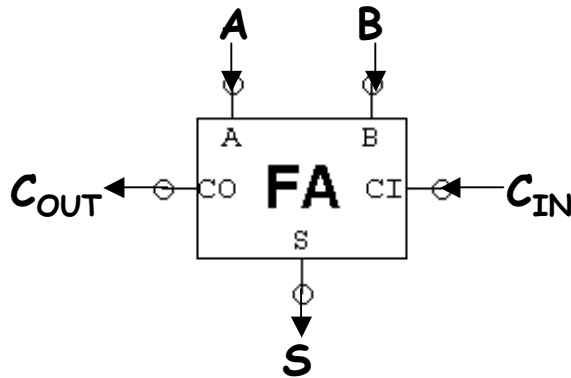
$$t_{PD} = (N-1) \underbrace{(t_{PD,OR} + t_{PD,AND})}_{CI \text{ to } CO} + \underbrace{t_{PD,XOR}}_{CI_{N-1} \text{ to } S_{N-1}} \approx \Theta(N)$$

$\Theta(N)$ is read "order N": means that the latency of our adder grows at worst in proportion to the number of bits in the operands.

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

Faster carry logic

Let's see if we can improve the speed by rewriting the equations for C_{OUT} :



$$\begin{aligned}
 C_{OUT} &= AB + AC_{IN} + BC_{IN} \\
 &= AB + (A + B)C_{IN} \\
 &= G + P C_{IN}
 \end{aligned}$$

↑ generate ↑ propagate

where $G = AB$ and
 $P = A + B$

Generate (G) = AB

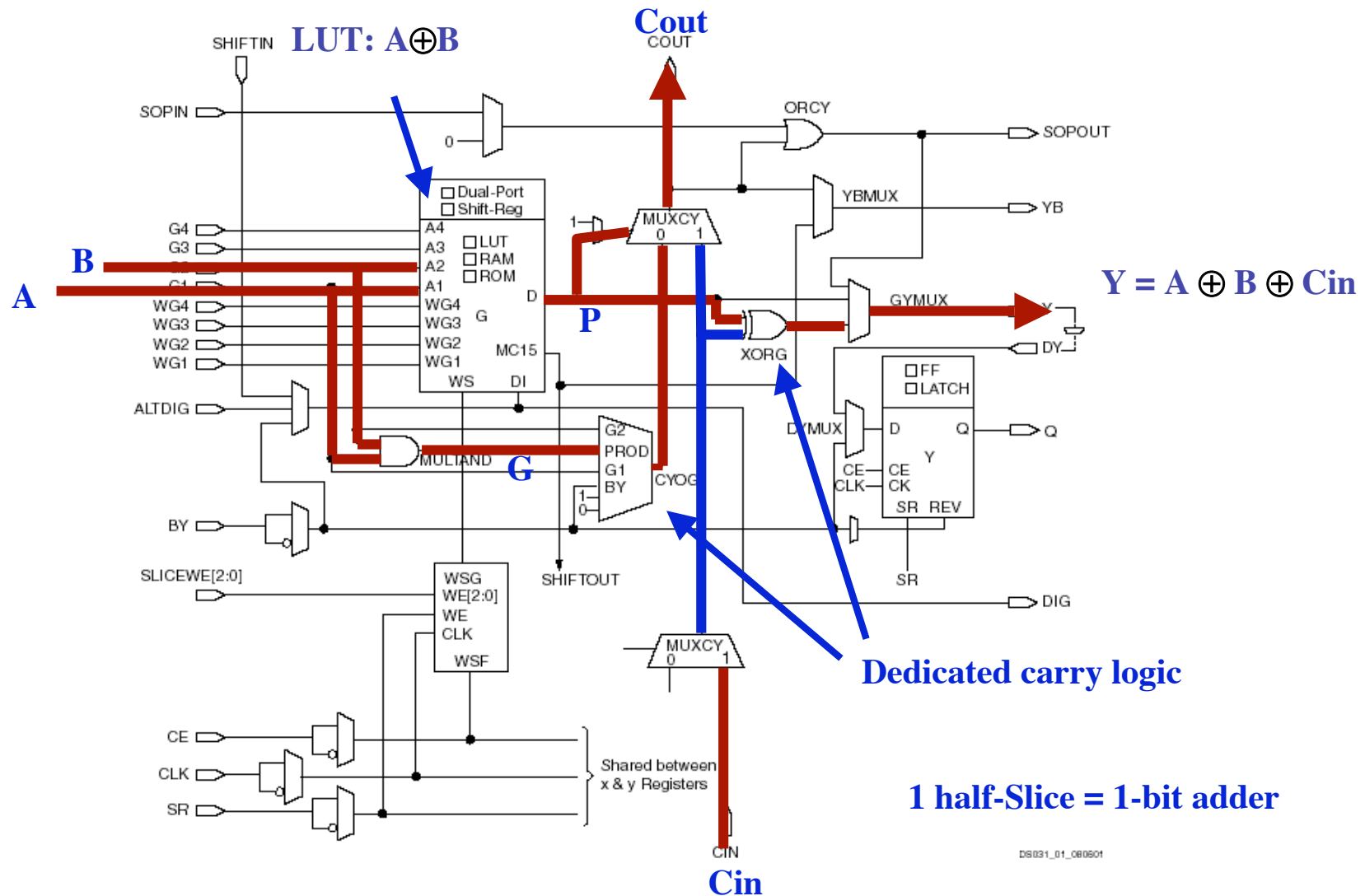
Propagate (P) = $A \oplus B$

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

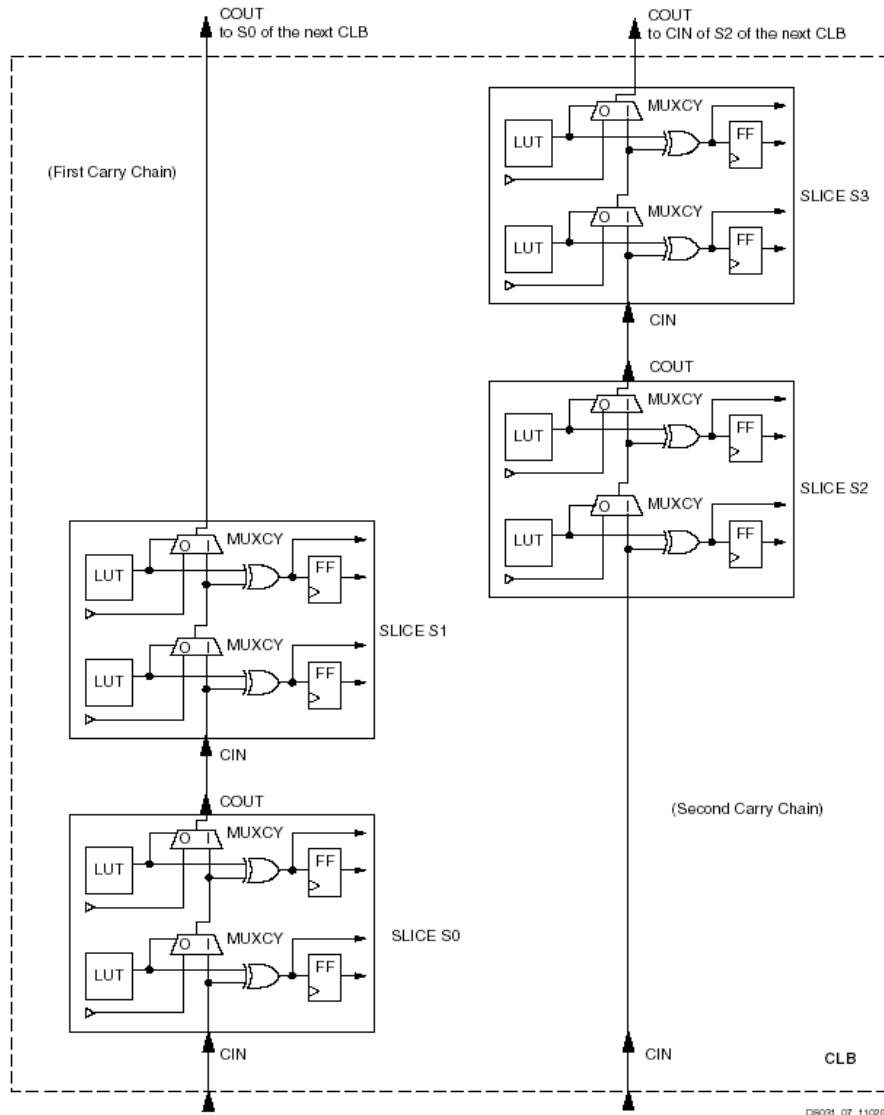
$$S(G, P) = P \oplus C_i$$

Actually, P is usually defined as $P = A \oplus B$ which won't change C_{OUT} but will allow us to express S as a simple function :
 $S = P \oplus C_{IN}$

Virtex II Adder Implementation

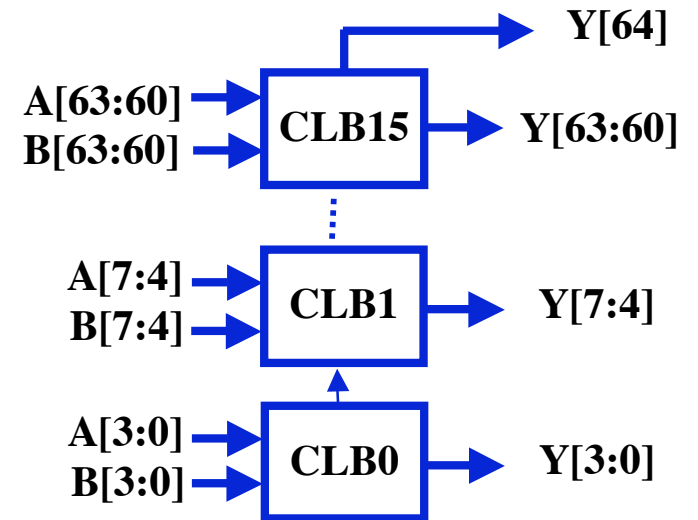
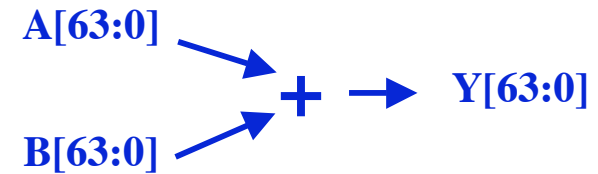


Virtex II Carry Chain



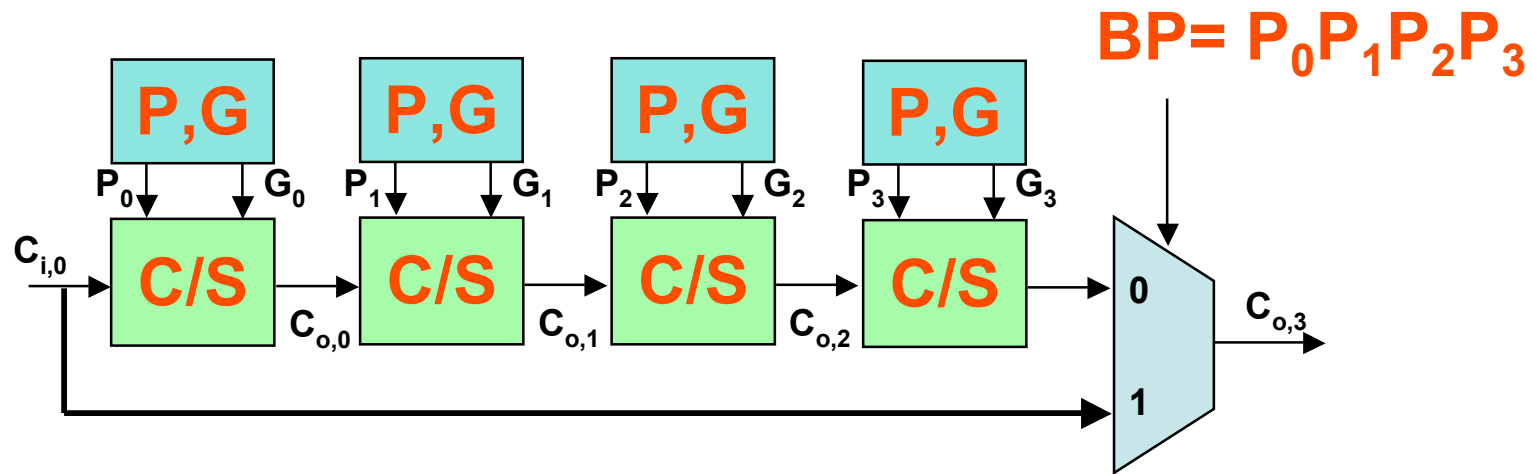
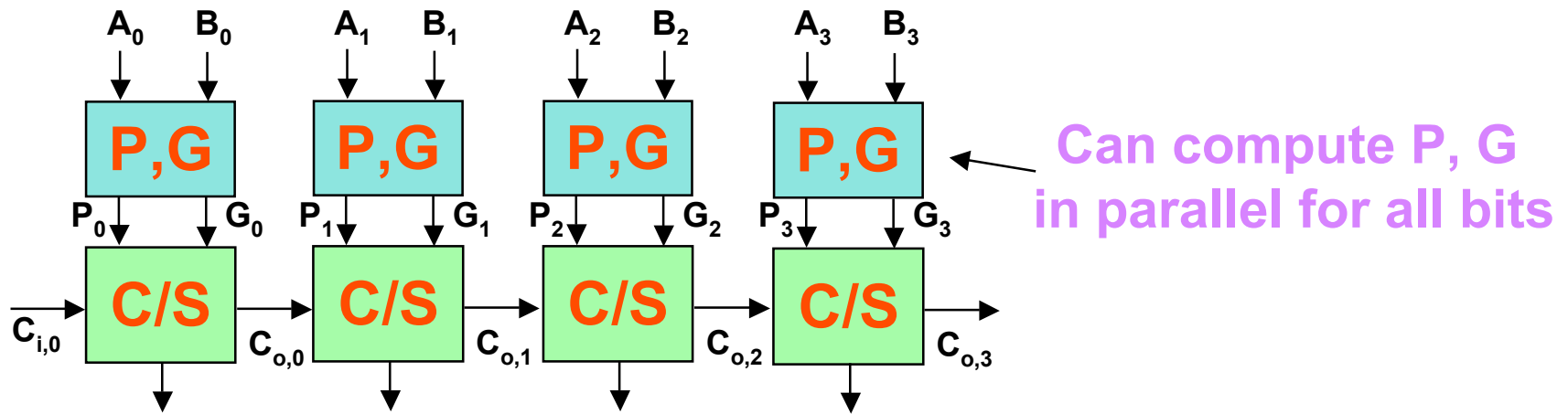
1 CLB = 4 Slices = 2, 4-bit adders

64-bit Adder: 16 CLBs



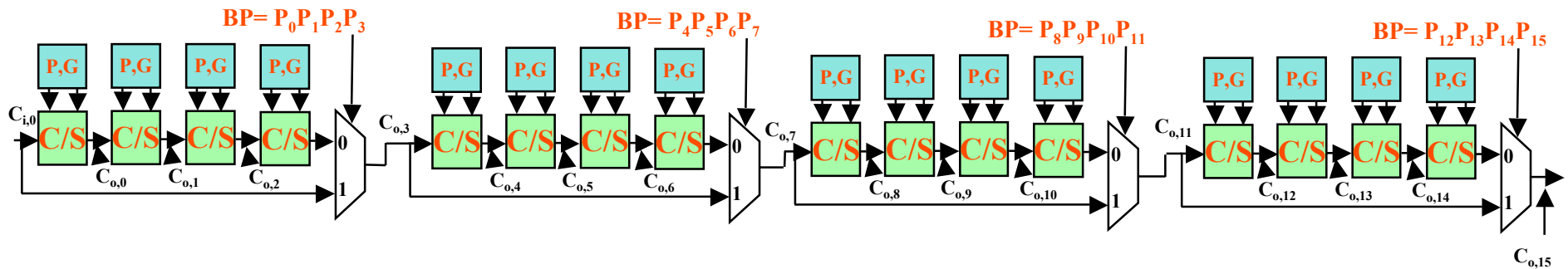
CLBs must be in same column

4. Carry Bypass Adder



Key Idea: if $(P_0 P_1 P_2 P_3)$ then $C_{o,3} = C_{i,0}$

16-bit Carry Bypass Adder



What is the worst case propagation delay for the 16-bit adder?

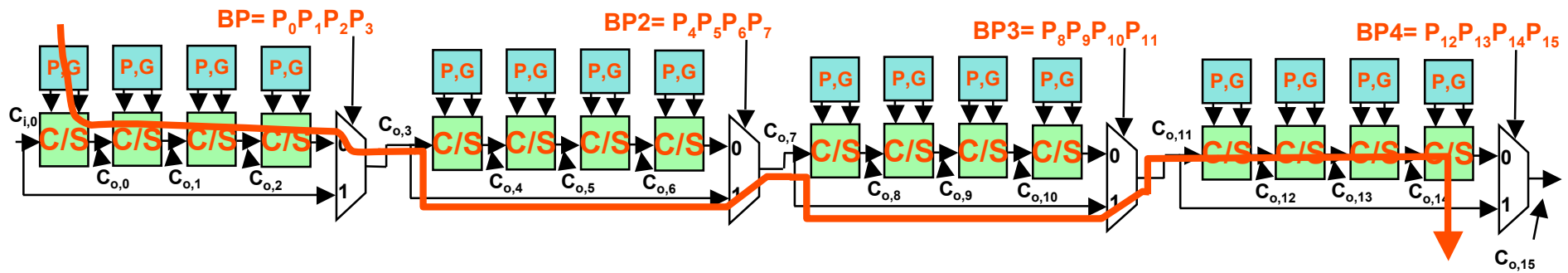
Assume the following for delay each gate:

P, G from A, B: 1 delay unit

P, G, C_i to C_o or Sum for a C/S: 1 delay unit

2:1 mux delay: 1 delay unit

Critical Path Analysis



For the second stage, is the critical path:

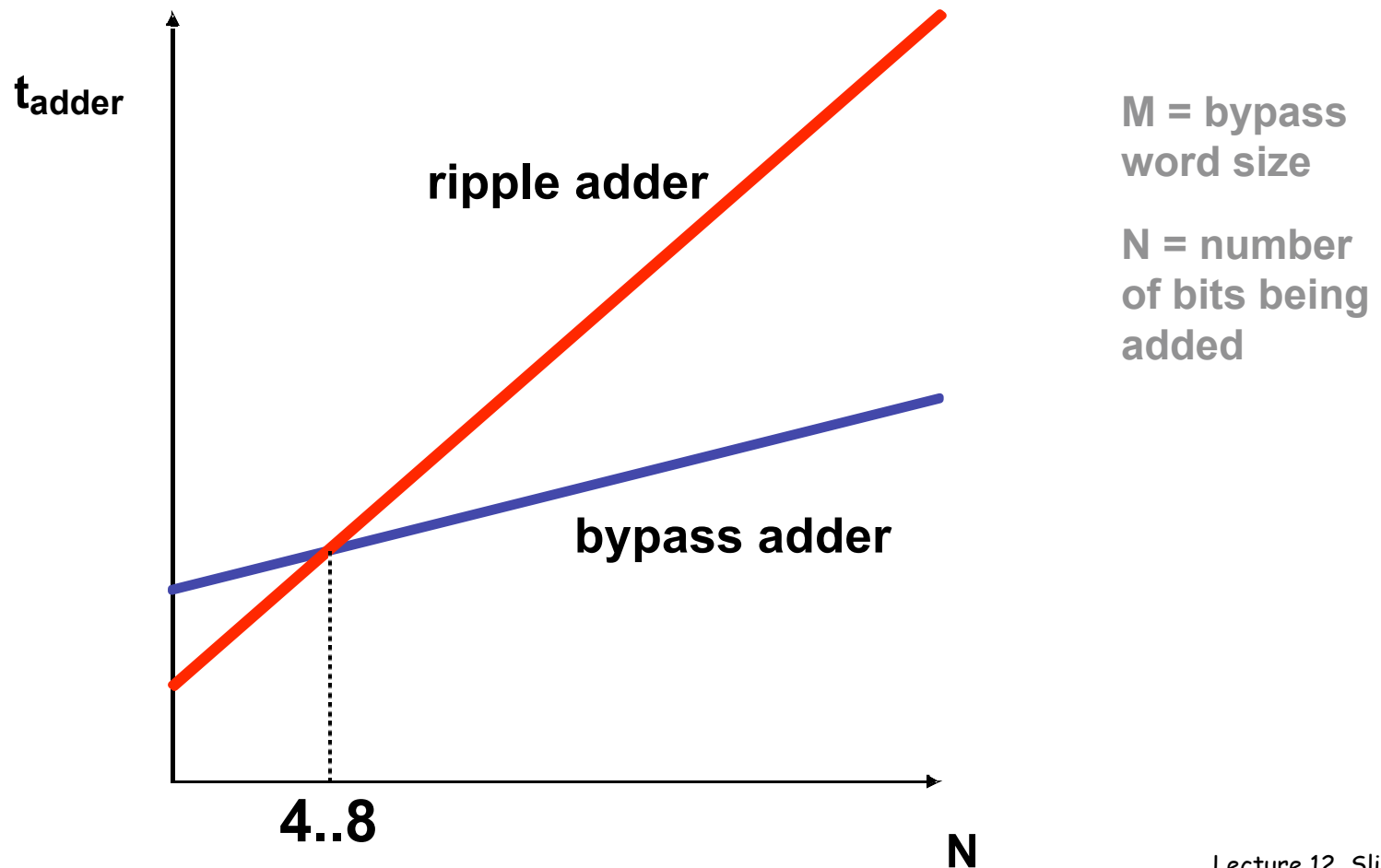
BP2 = 0 or BP2 = 1 ?

**Message: Timing Analysis is Very Tricky –
Must Carefully Consider Data Dependencies For
False Paths**

Carry Bypass vs Ripple Carry

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

Carry Bypass: $t_{\text{adder}} = 2(M-1) t_{\text{carry}} + t_{\text{sum}} + (N/M-1) t_{\text{bypass}}$



5. Carry Lookahead Adder (CLA)

- Recall that $C_{OUT} = G + P C_{IN}$ where $G = AB$ and $P = A \oplus B$
- For adding two N-bit numbers:

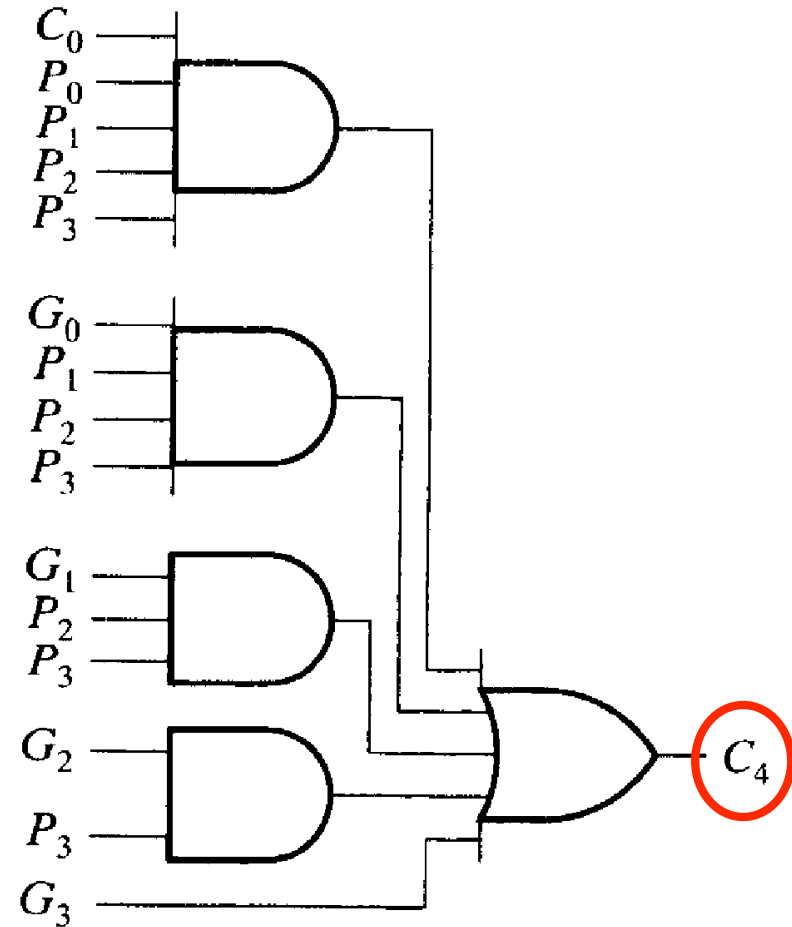
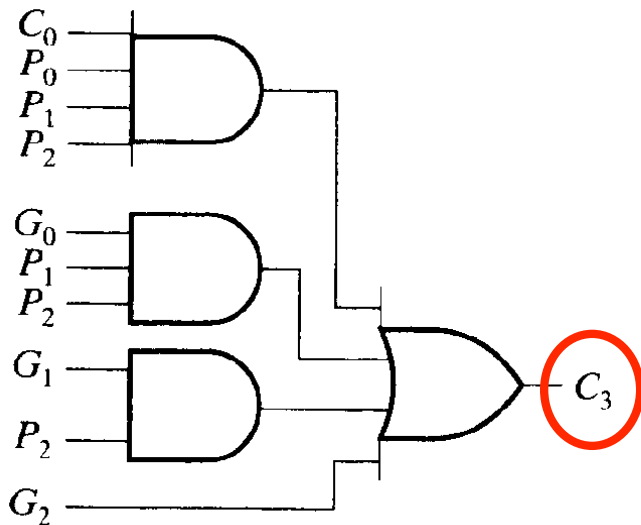
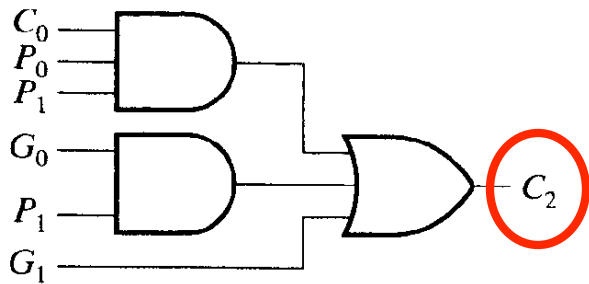
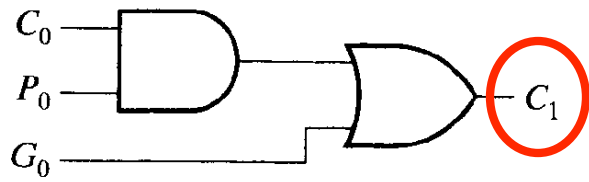
$$\begin{aligned}C_N &= G_{N-1} + P_{N-1}C_{N-1} \\ &= G_{N-1} + P_{N-1}G_{N-2} + P_{N-1}P_{N-2}C_{N-2} \\ &= G_{N-1} + P_{N-1}G_{N-2} + P_{N-1}P_{N-2}G_{N-3} + \dots + P_{N-1} \dots P_0 C_{IN}\end{aligned}$$

C_N in only 3 gate delays* :
1 for P/G generation, 1 for ANDs, 1 for final OR

*assuming gates with N inputs

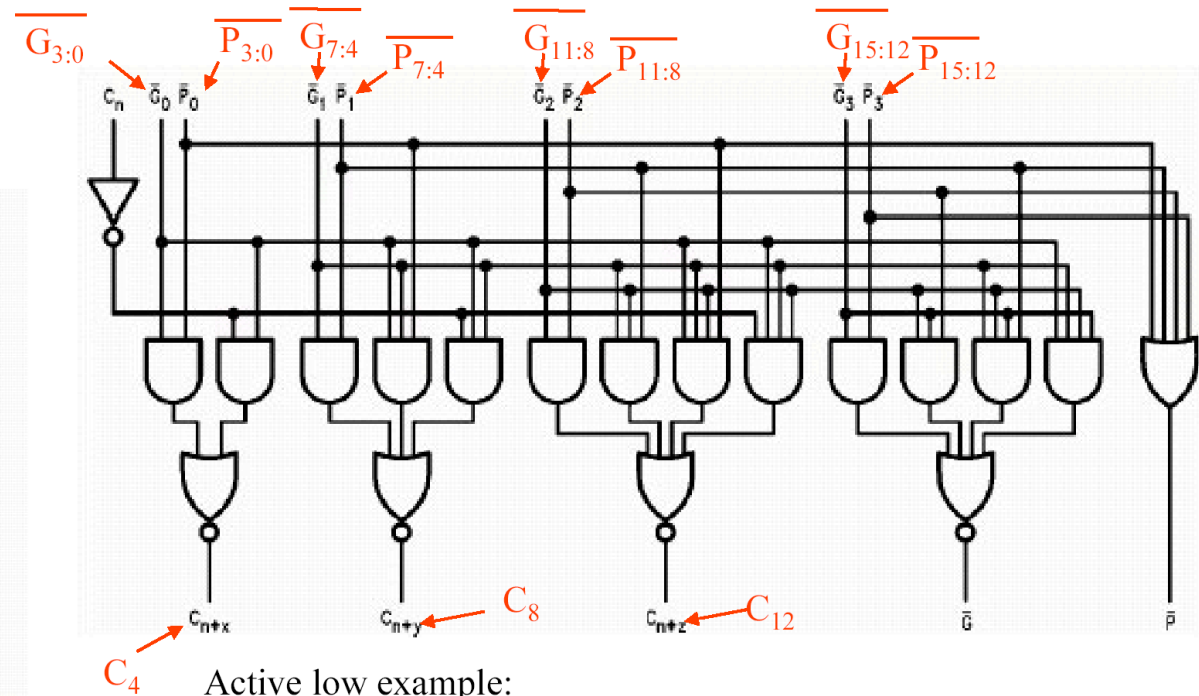
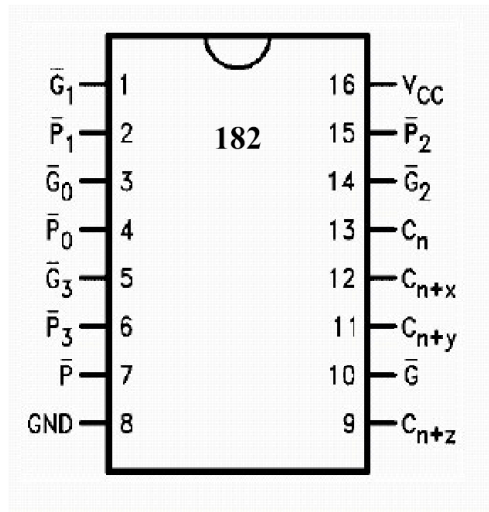
- Idea: pre-compute all carry bits as $f(Gs, Ps, C_{IN})$

Carry Lookahead Circuits



The 74182 Carry Lookahead Unit

74182 carry lookahead unit



Active low example:

$$\begin{aligned} C_{n+x} &= \overline{\overline{G_0} \cdot \overline{P_0}} + \overline{\overline{G_0} \cdot \overline{C_n}} \\ &= \overline{\overline{G_0} \cdot \overline{P_0} \cdot \overline{G_0} \cdot \overline{C_n}} \\ &= (G_0 + P_0) \cdot (G_0 + C_n) = G_0 + P_0 C_n \end{aligned}$$

$$\rightarrow C_4 = G_{3:0} + P_{3:0} C_n$$

$$C_{n+y} = C_8 = G_{7:4} + P_{7:4} G_{3:0} + P_{7:4} P_{3:0} C_{i,0} = G_{7:0} + P_{7:0} C_n$$

$$\begin{aligned} C_{n+z} = C_{12} &= G_{11:8} + P_{11:8} G_{7:4} + P_{11:8} P_{7:4} G_{3:0} + P_{11:8} P_{7:4} P_{3:0} C_n \\ &= G_{11:0} + P_{11:0} C_n \end{aligned}$$

- high speed carry lookahead generator
- used with 74181 to extend carry lookahead beyond 4 bits
- correctly handles the carry polarity of the 181

Block Generate and Propagate

G and P can be computed for groups of bits (instead of just for individual bits). This allows us to choose the maximum fan-in we want for our logic gates and then build a hierarchical carry chain using these equations:

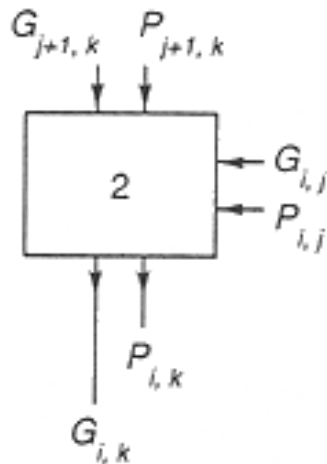
$$C_{J+1} = G_{IJ} + P_{IJ}C_I$$

$$G_{IK} = G_{J+1,K} + P_{J+1,K}G_{IJ}$$

$$P_{IK} = P_{IJ}P_{J+1,K}$$

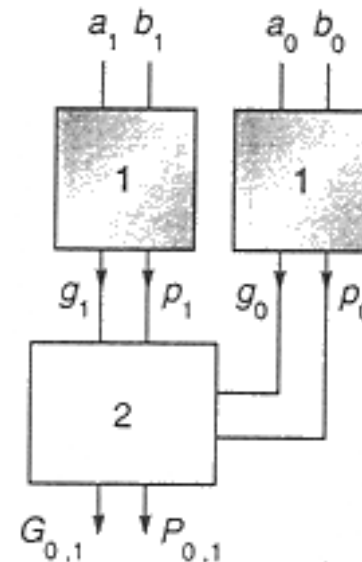
“generate a carry from bits I thru K if it is generated in the high-order ($J+1, K$) part of the block or if it is generated in the low-order (I, J) part of the block and then propagated thru the high part”

where $I < J$ and $J+1 < K$



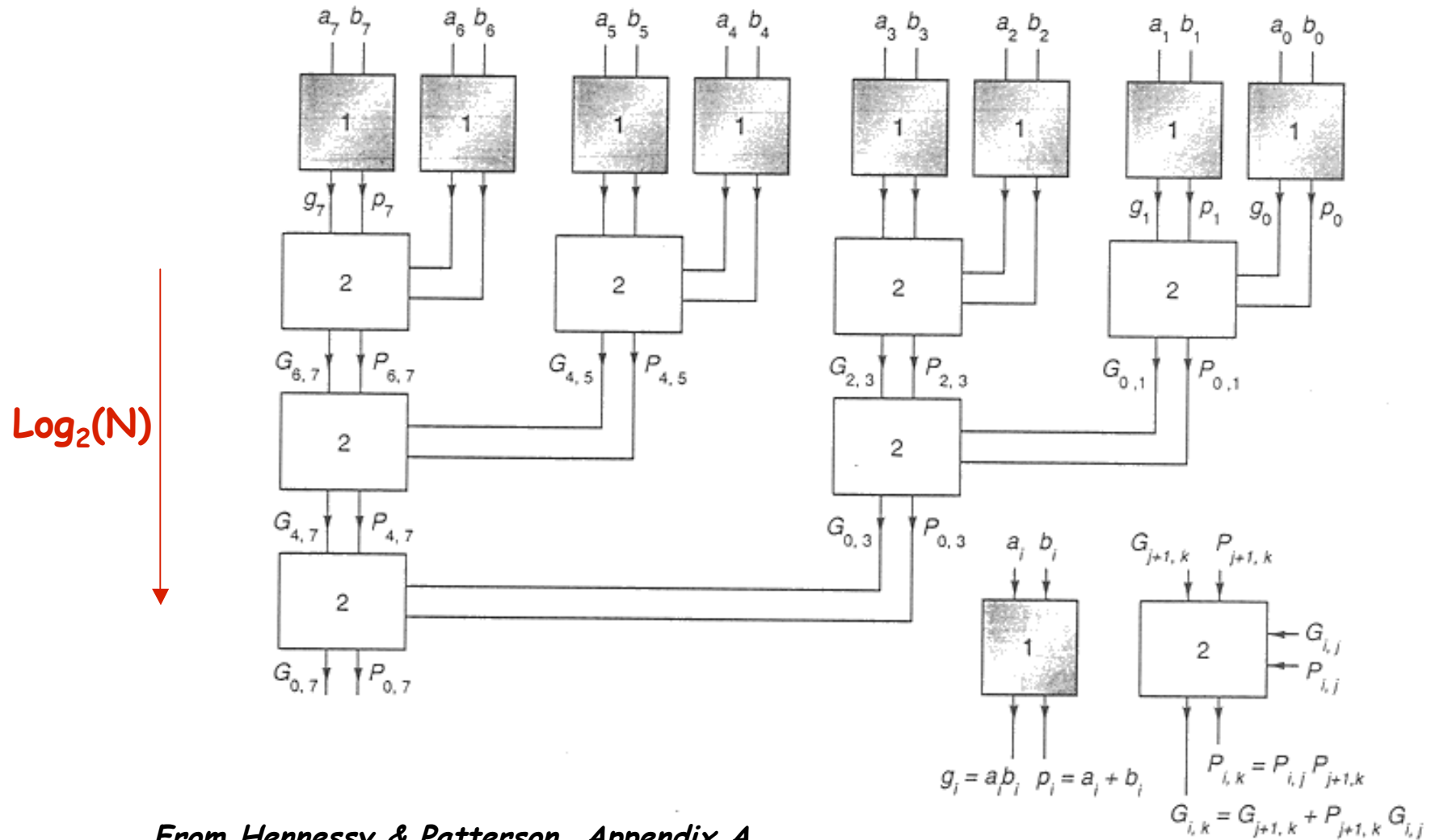
P/G generation

1st level of lookahead

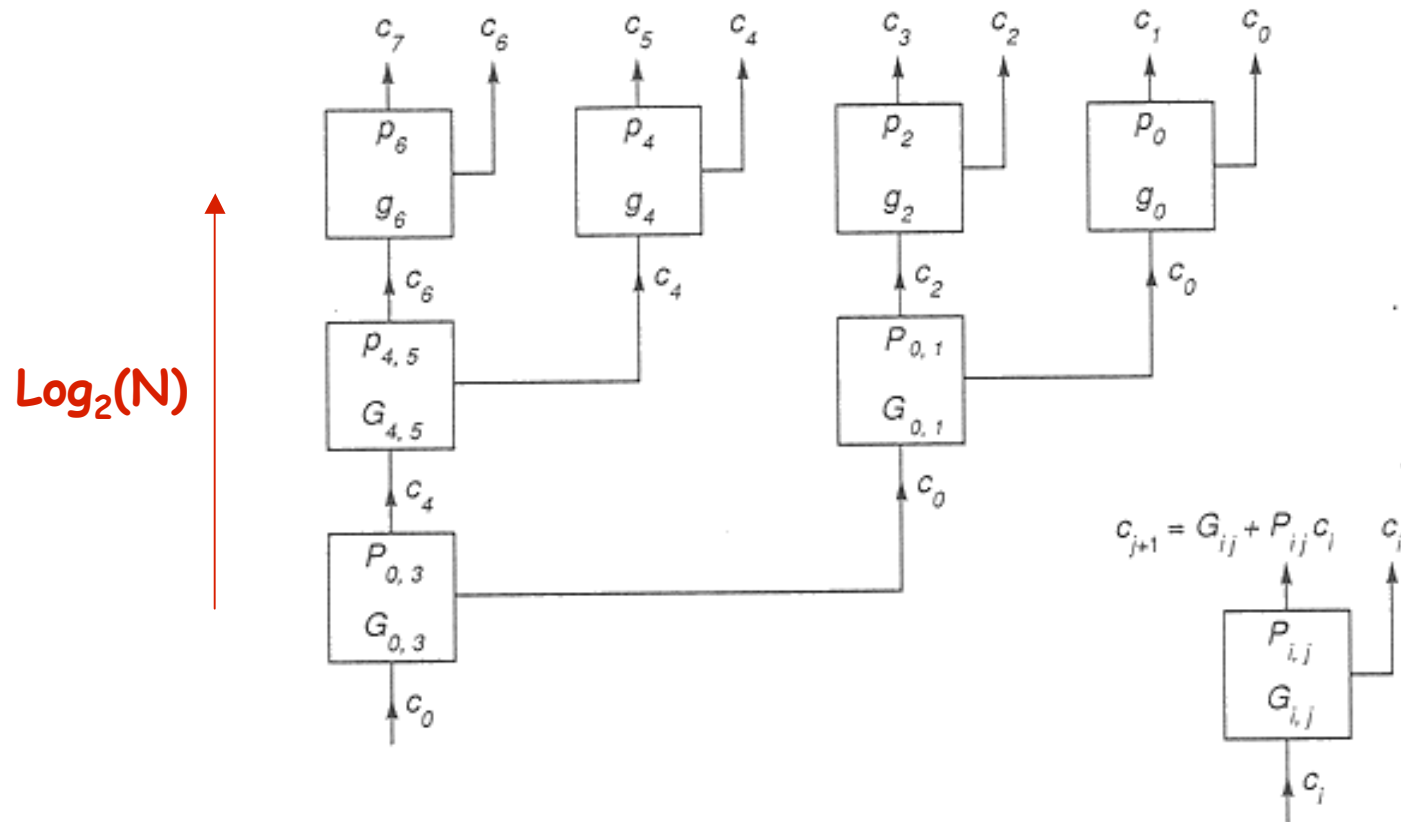


Hierarchical building block

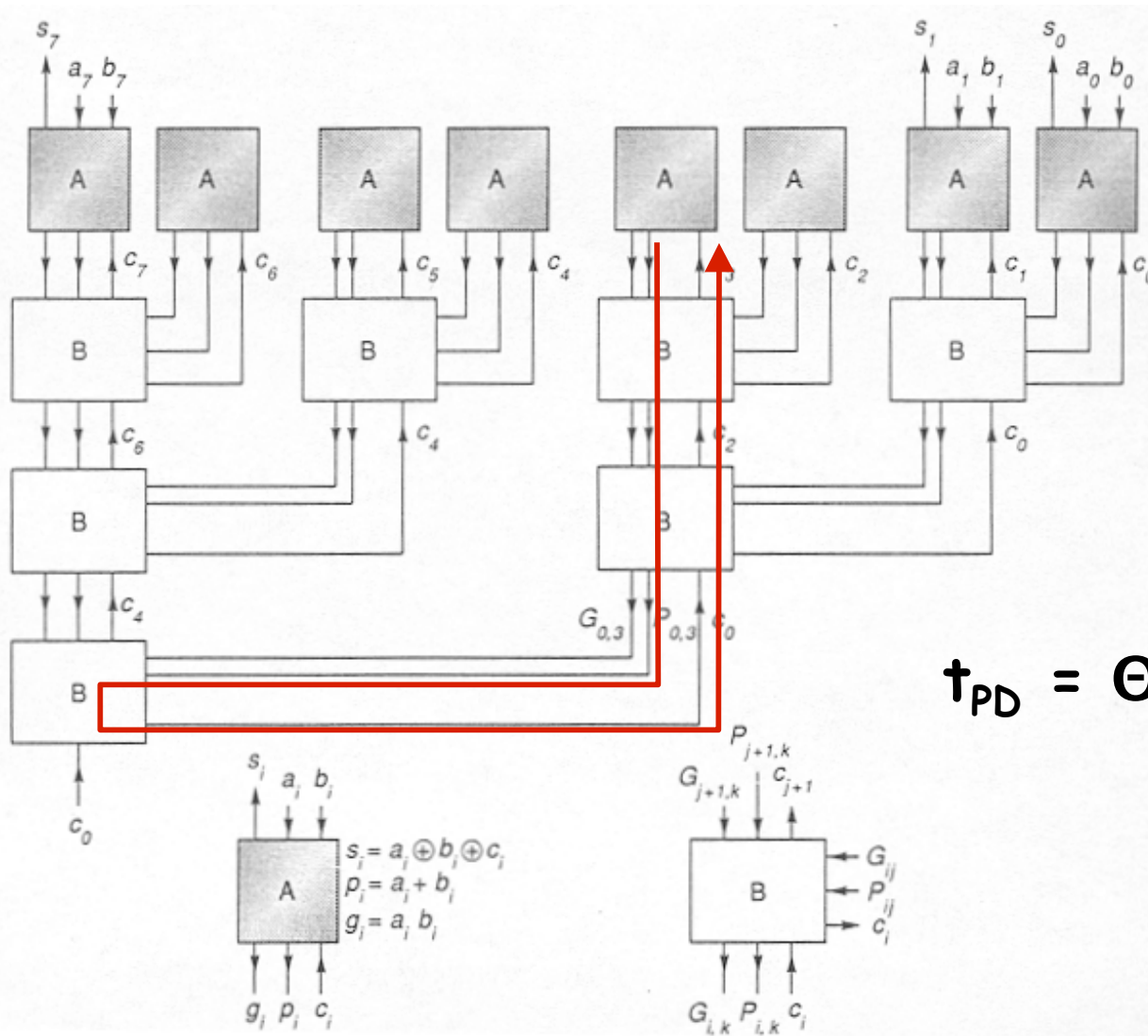
8-bit CLA (P/G generation)



8-bit CLA (carry generation)

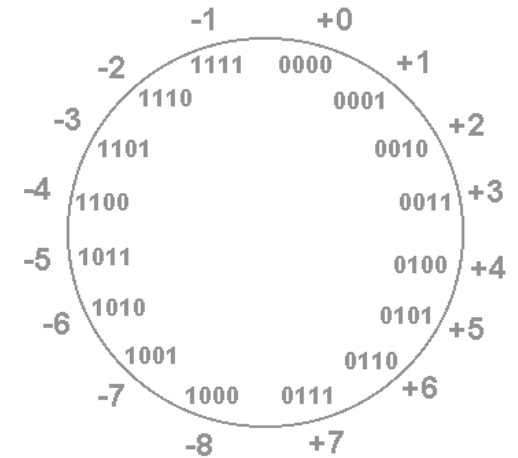


8-bit CLA (complete)



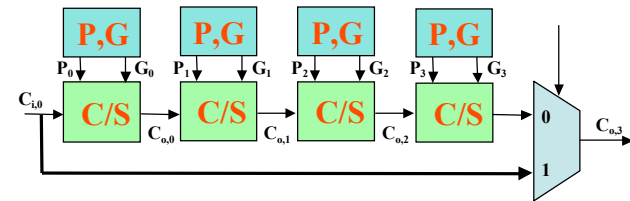
$$t_{PD} = \Theta(\log(N))$$

Summary



- **Negative numbers:**
 - Twos Complement $-B = \bar{B} + 1$
 - Addition & Subtraction use same adder

- **Ripple Carry Adder:**
 - $t_{\text{adder}} = (N-1) t_{\text{carry}} + t_{\text{sum}}$



- **Carry Bypass Adder:**
 - $t_{\text{adder}} \frac{1}{4} (M-1) t_{\text{carry}} + t_{\text{sum}} + (N/M-1) t_{\text{bypass}}$

- **Carry Lookahead Adder:**
 - $t_{\text{adder}} \frac{1}{4} 2 \log_2(N) t_{\text{pg}}$

