Announcements

- Midterm Exam Two is scheduled on April 6 in class.
- On March 25 I will help you prepare Midterm Exam Two.
- On March 9 I will teach "Introduction to Pipelining".

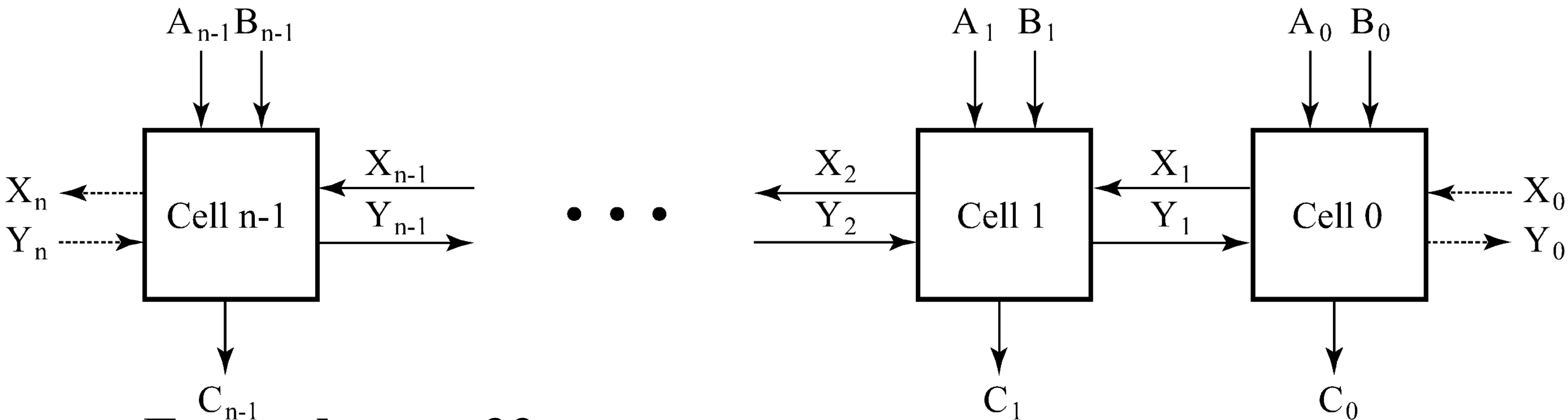
Chapter 3: Part 3 Arithmetic Functions

- Iterative combinational circuits
- Binary adders
  - Half and full adders
  - Ripple carry and carry lookahead adders
- Binary subtraction
- Binary adder-subtractors
  - Signed binary numbers
  - Signed binary addition and subtraction
  - Overflow
- Binary multiplication

Iterative Combinational Circuits

- **Arithmetic functions**
  - **Operate on binary vectors**
  - **Use the same subfunction in each bit position**
- **Can design functional block for subfunction and repeat to obtain functional block for overall function**
- ***Cell* - subfunction block**
- ***Iterative array* - a array of interconnected cells**

Block Diagram of a 1D Iterative Array



- **Example: n = 32**
  - **Number of inputs = ?**
  - **Truth table rows = ?**
  - **Equations with up to ? input variables**
  - **Equations with huge number of terms**
  - **Design impractical!**
- **Iterative array takes advantage of the regularity to make design feasible**

Functional Blocks: Addition

- **Binary addition used frequently**
- **Addition Development:**
  - ***Half-Adder* (HA), a 2-input bit-wise addition functional block,**
  - ***Full-Adder* (FA), a 3-input bit-wise addition functional block,**
  - ***Ripple Carry Adder*, an iterative array to perform binary addition, and**
  - ***Carry-Look-Ahead Adder* (CLA), a hierarchical structure to improve performance.**

Functional Block: Half-Adder

- A 2-input, 1-bit width binary adder that performs the following computations:

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- A half adder adds two bits to produce a two-bit sum
- The sum is expressed as a sum bit , S and a carry bit, C
- The half adder can be specified as a truth table for S and C ⇒

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Logic Simplification: Half-Adder

- **The K-Map for S, C is:**
- **This is a pretty trivial map! By inspection:**

S:

	$\overline{Y}$	Y
$\overline{X}$	0	1 (1)
X	1 (2)	3

C:

	$\overline{Y}$	Y
$\overline{X}$	0	1
X	2	1 (3)

$$S = X \cdot \overline{Y} + \overline{X} \cdot Y = X \oplus Y$$

$$S = (X + Y) \cdot (\overline{X} + \overline{Y})$$

- **and**

$$C = X \cdot Y$$

$$C = \overline{(\overline{(X \cdot Y)})}$$

- **These equations lead to several implementations.**

Five Implementations: Half-Adder

- We can derive following sets of equations for a half-adder:

(a) $S = X \cdot \overline{Y} + \overline{X} \cdot Y$
$\quad C = X \cdot Y$

(b) $S = (X + Y) \cdot (\overline{X} + \overline{Y})$
$\quad C = X \cdot Y$

(c) $S = \overline{(C + \overline{X} \cdot \overline{Y})}$
$\quad C = X \cdot Y$

(d) $S = (X + Y) \cdot \overline{C}$
$\quad \overline{C} = (\overline{X} + \overline{Y})$

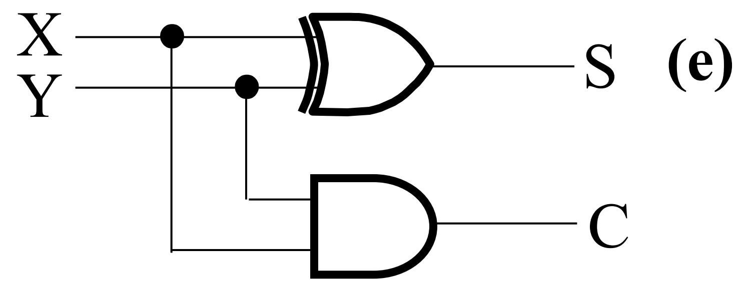
(e) $S = X \oplus Y$
$\quad C = X \cdot Y$

- (a), (b), and (e) are SOP, POS, and XOR implementations for S.
- In (c), the C function is used as a term in the AND-NOR implementation of S, and in (d), the $\overline{C}$ function is used in a POS term for S.

Implementations: Half-Adder

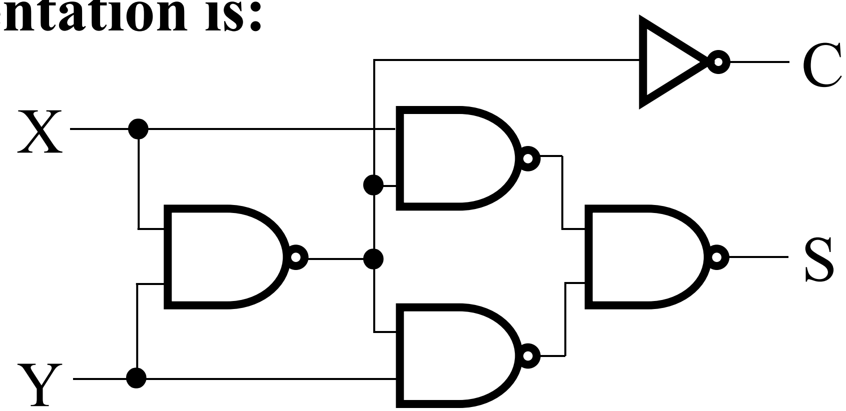
- **The most common half adder implementation is:**

$$S = X \oplus Y$$
$$C = X \cdot Y$$



- **A NAND only implementation is:**

$$S = (X + Y) \cdot C$$
$$C = \overline{(\overline{(X \cdot Y)})}$$



Functional Block: Full-Adder

- A full adder is similar to a half adder, but includes a carry-in bit from lower stages. Like the half-adder, it computes a sum bit, S and a carry bit, C.
  - For a carry-in (Z) of 0, it is the same as the half-adder:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

  - For a carry- in (Z) of 1:

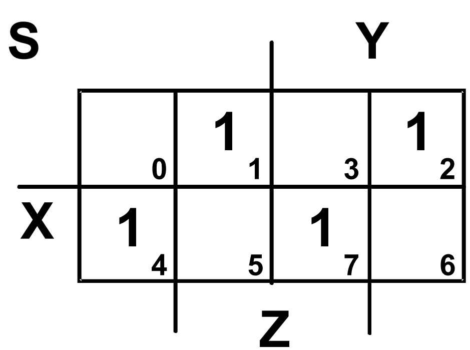
Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

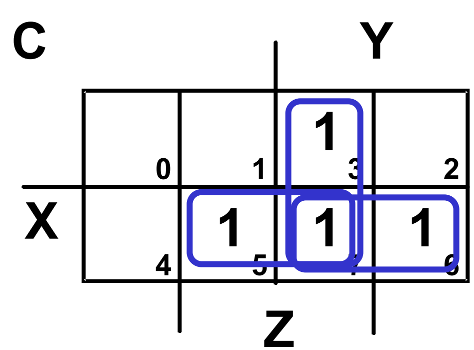
Logic Optimization: Full-Adder

- **Full-Adder Truth Table:**

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- **Full-Adder K-Map:**





Equations: Full-Adder

- **From the K-Map, we get:**

$$S = X\,\overline{Y}\,\overline{Z} + \overline{X}\,Y\,\overline{Z} + \overline{X}\,\overline{Y}\,Z + X\,Y\,Z$$
$$C = X\,Y + X\,Z + Y\,Z$$

- **The S function is the three-bit XOR function (Odd Function):**

$$S = X \oplus Y \oplus Z$$

- **The Carry bit C is 1 if both X and Y are 1 or if the sum is 1 and a carry-in (Z) occurs. Thus C can be re-written as:**

$$C = X\,Y + (X \oplus Y)\,Z$$

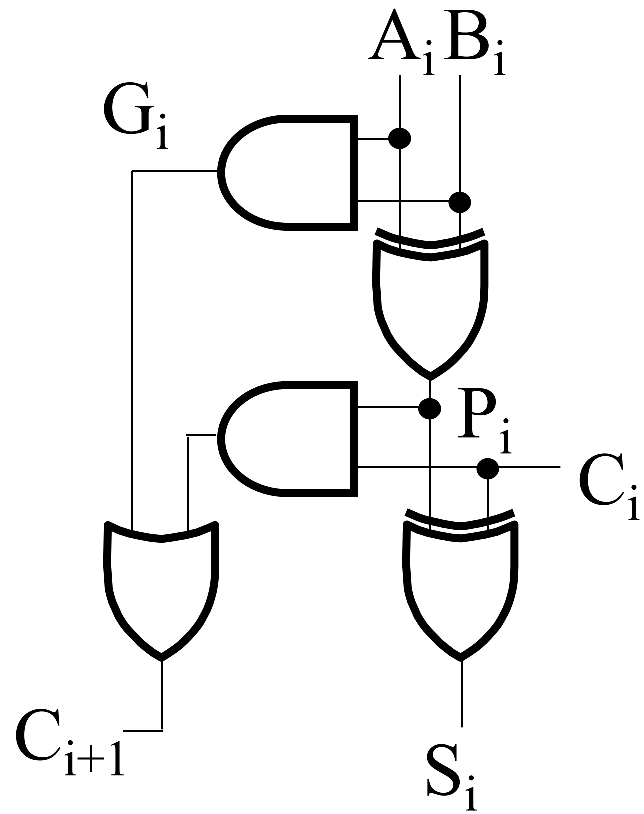
- **The term $X \cdot Y$ is *carry generate*.**
- **The term $X \oplus Y$ is *carry propagate*. (why?)**

Implementation: Full Adder

- **Full Adder Schematic**
- **Here X, Y, and Z, and C (from the previous pages) are A, B, C_i and C_{i+1}, respectively. Also,**

  G = generate and

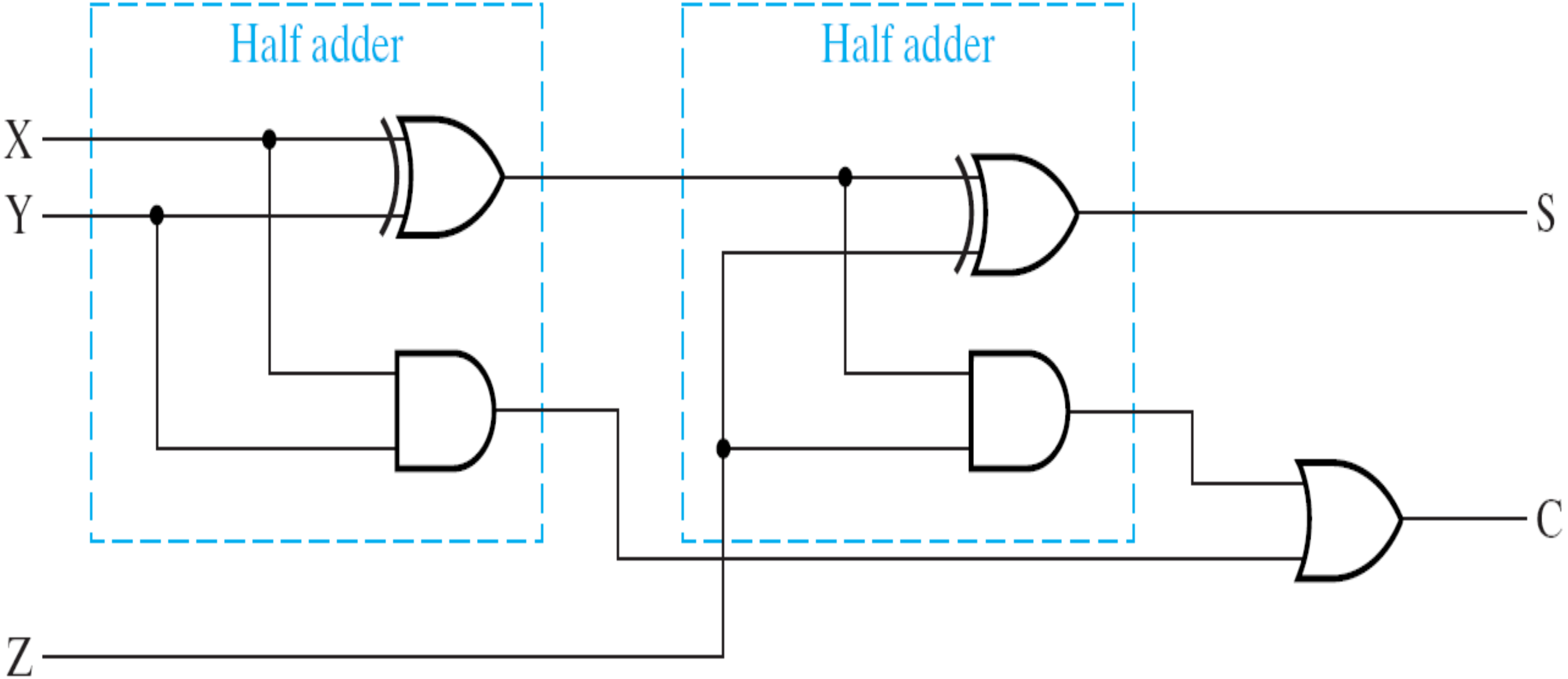
  P = propagate.
- **Note: This is really a combination of a 3-bit odd function (for S)) and Carry logic (for C_{i+1}):**



(G = Generate) OR (P =Propagate AND C_i = Carry In)

$C_{i+1} = G + P_i \cdot C_i$

Logic Diagram of Full Adder



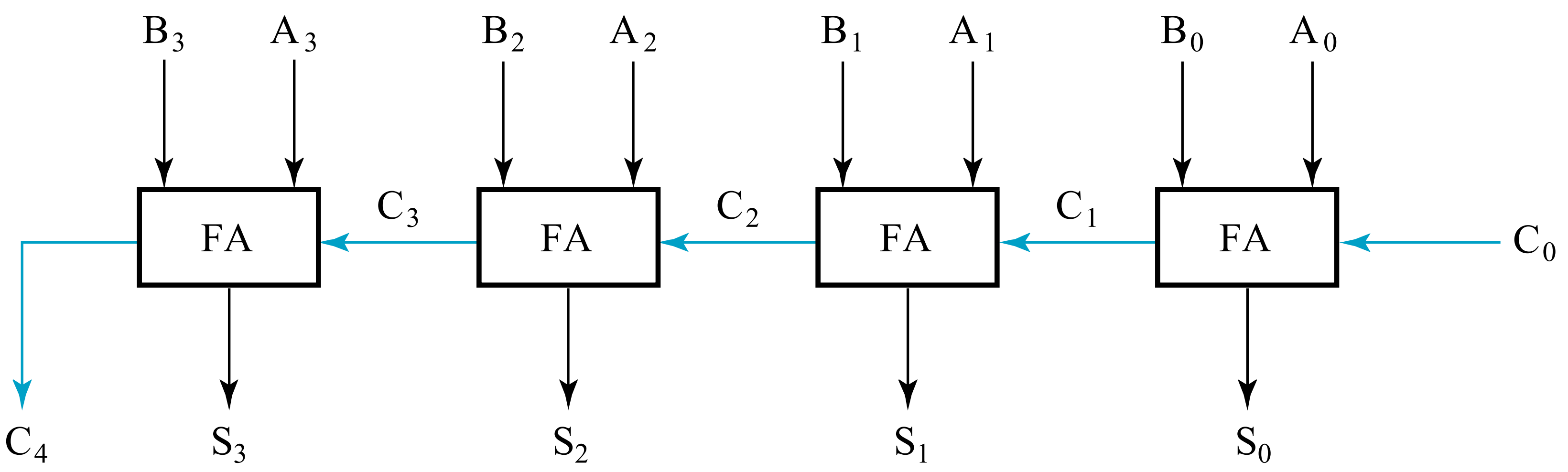
Binary Adders

- To add multiple operands, we "bundle" logical signals together into vectors and use functional blocks that operate on the vectors
- Example: 4-bit ripple carry adder: Adds input vectors A(3:0) and B(3:0) to get a sum vector S(3:0)
- Note: carry out of cell i becomes carry in of cell i + 1

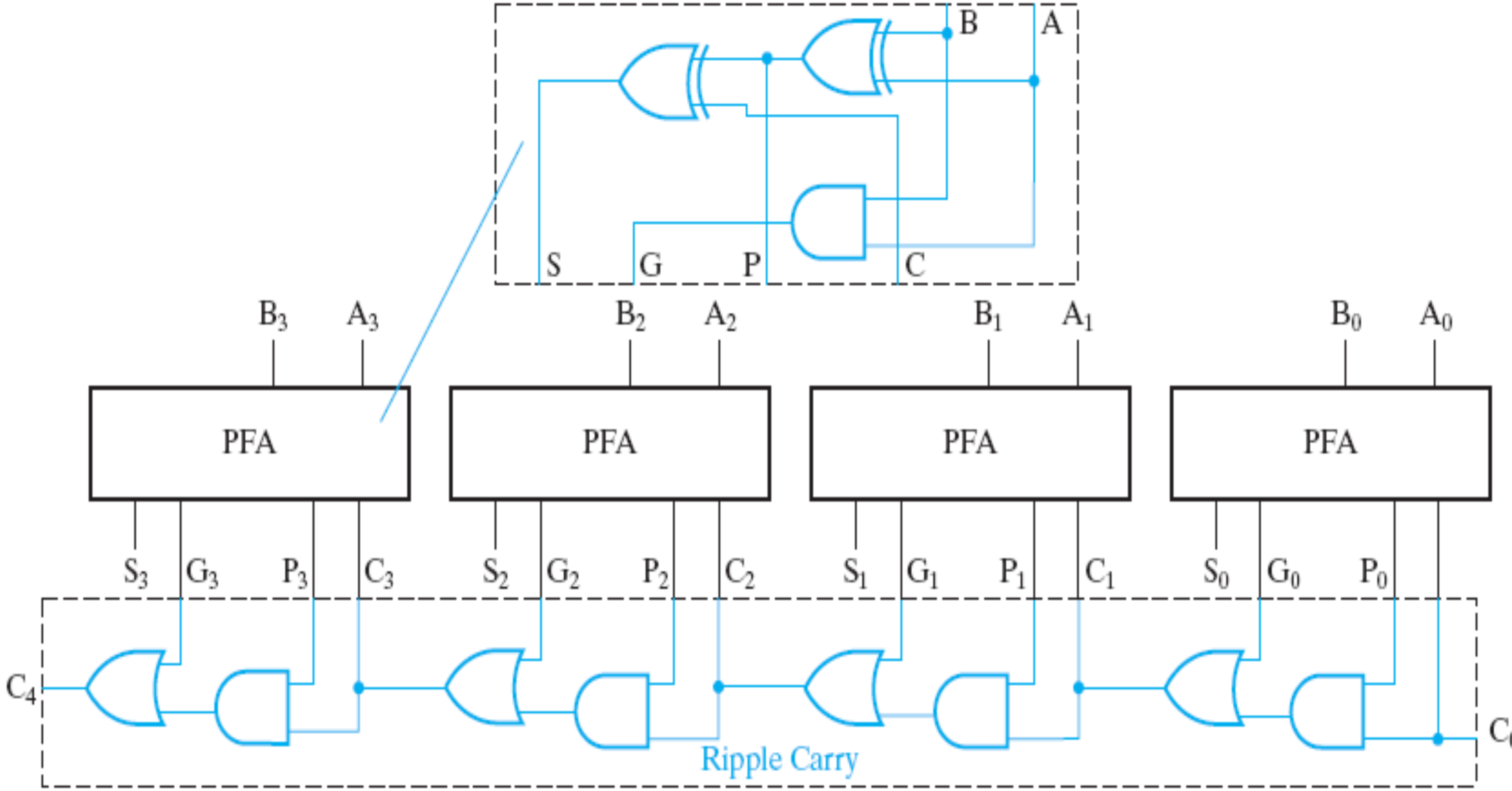
Description	Subscript 3 2 1 0	Name
Carry In	0 1 1 0	C_i
Augend	1 0 1 1	A_i
Addend	0 0 1 1	B_i
Sum	1 1 1 0	S_i
Carry out	0 0 1 1	C_{i+1}

4-bit Ripple-Carry Binary Adder

- **A four-bit Ripple Carry Adder made from four 1-bit Full Adders:**



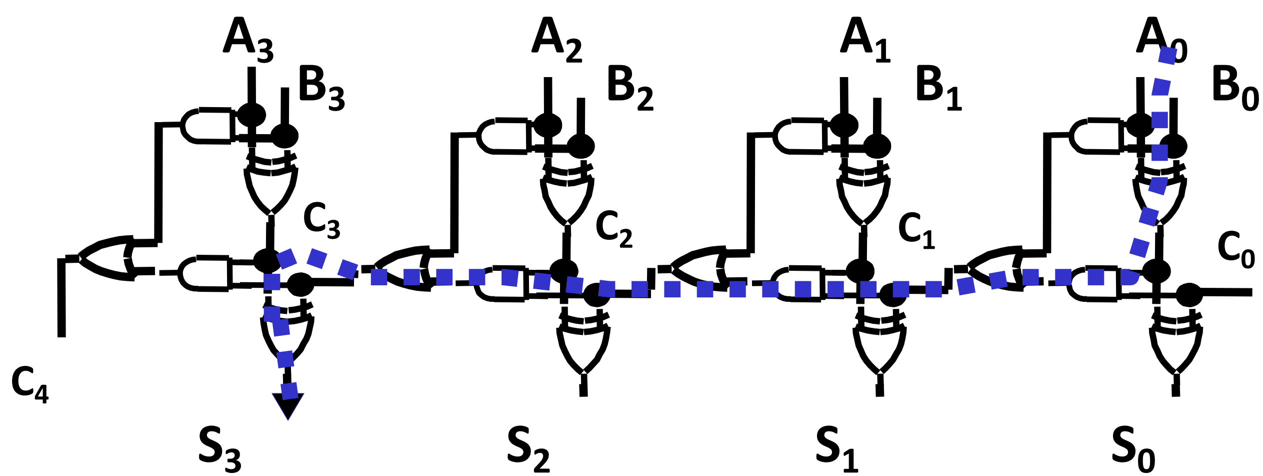
Ripple Carry



(a)

Carry Propagation & Delay

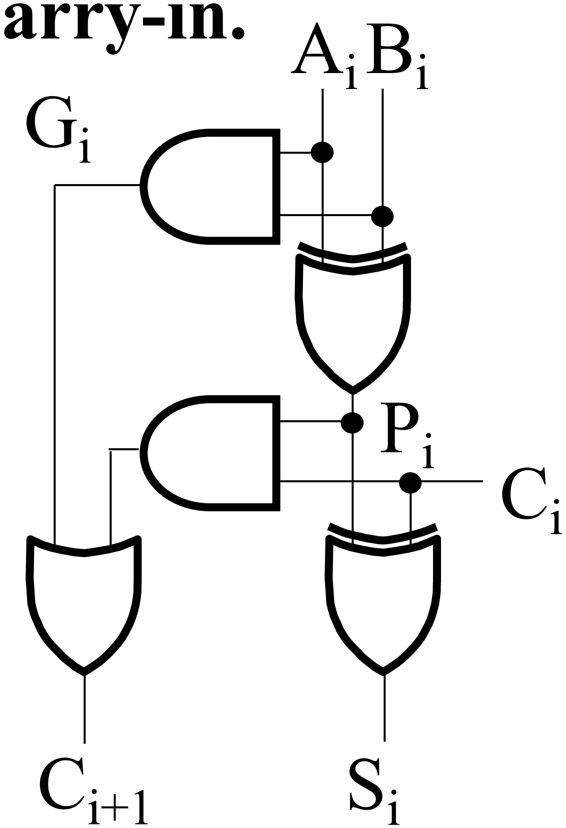
- One problem with the addition of binary numbers is the length of time to propagate the ripple carry from the least significant bit to the most significant bit.
- The gate-level propagation path for a 4-bit ripple carry adder of the last example:



- Note: The "long path" is from A_0 or B_0 though the circuit to S_3.

Carry Lookahead

- Given Stage i from a Full Adder, we know that there will be a carry generated when $A_i = B_i =$ "1", whether or not there is a carry-in.
- Alternately, there will be a carry propagated if the "half-sum" is "1" and a carry-in, C_i occurs.
- These two signal conditions are called *generate*, denoted as G_i, and *propagate*, denoted as P_i respectively and are identified in the circuit:



Carry Lookahead (continued)

- **In the ripple carry adder:**
  - **Gi, Pi, and Si are <u>local</u> to each cell of the adder**
  - **Ci is also local each cell**
- **In the carry lookahead adder, in order to reduce the length of the carry chain, Ci is changed to a more global function spanning multiple cells**
- **Defining the equations for the Full Adder in term of the P_i and G_i:**

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

$$S_i = P_i \oplus C_i \qquad\qquad C_{i+1} = G_i + P_i C_i$$

Carry Lookahead Development

- C_{i+1} can be removed from the cells and used to derive a set of carry equations spanning multiple cells.
- Beginning at the cell 0 with carry in C_0:

$$C_1 = G_0 + P_0\, C_0$$

$$C_2 = G_1 + P_1\, C_1 = G_1 + P_1(G_0 + P_0\, C_0)$$
$$= G_1 + P_1G_0 + P_1P_0\, C_0$$

$$C_3 = G_2 + P_2\, C_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0\, C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0\, C_0$$

$$C_4 = G_3 + P_3\, C_3 = G_3 + P_3G_2 + P_3P_2G_1$$
$$+ P_3P_2P_1G_0 + P_3P_2P_1P_0\, C_0$$

Carry Lookahead Adder