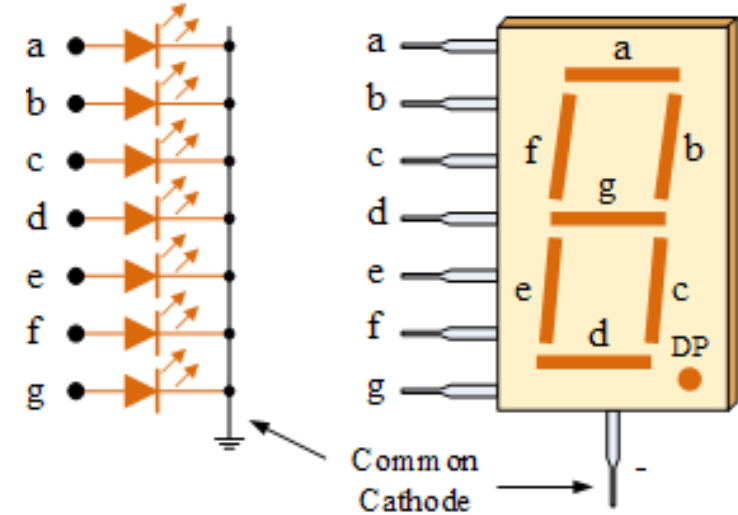


# Lecture 13 – Combinational logic circuits 2

Dr. Aftab M. Hussain,  
Assistant Professor, PATRIOT Lab, CVEST

# 7-segment decoder

- Let us choose common cathode LED display to make the function
- Thus, we can make the truth-table
- Obviously, the BCD system only goes from 0 to 9, while we have 16 rows for 4 inputs
- In the other rows, we fill all the outputs as don't care, because we are sure that these are not going to be input anyway (trust the engineer before you)
- Thus, we have six don't care conditions



| A | B | C | D | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

# 7-segment decoder

- Thus, the K-map for output  $a$  will look like this
- We have two clusters of 8:  $y$  and  $w$
- Two clusters of four:  $xz$  and  $x'z'$
- Thus, the logic function for  $a$  can be:

$$F_a = y + w + xz + x'z'$$

- Or we can have PoS as:
- One cluster of two:  $xy'z'$
- One min-term:  $w'x'y'z$
- Thus,

$$F_a = (x' + y + z)(w + x + y + z')$$

- This can be done for all the other outputs

|     |    | $y$      |     |    |                           |            |
|-----|----|----------|-----|----|---------------------------|------------|
|     |    | $yz$     | 00  | 01 | $\overbrace{11 \quad 10}$ |            |
| $w$ | 00 | $m_0$    | 1   | 0  | $m_3$ 1                   | $m_2$ 1    |
|     | 01 | $m_4$    | 0   | 1  | $m_7$ 1                   | $m_6$ 1    |
|     | 11 | $m_{12}$ | X   | X  | $m_{15}$ X                | $m_{14}$ X |
|     | 10 | $m_8$    | 1   | 1  | $m_{11}$ X                | $m_{10}$ X |
|     |    |          | $z$ |    |                           |            |

$x$

# 7-segment decoder

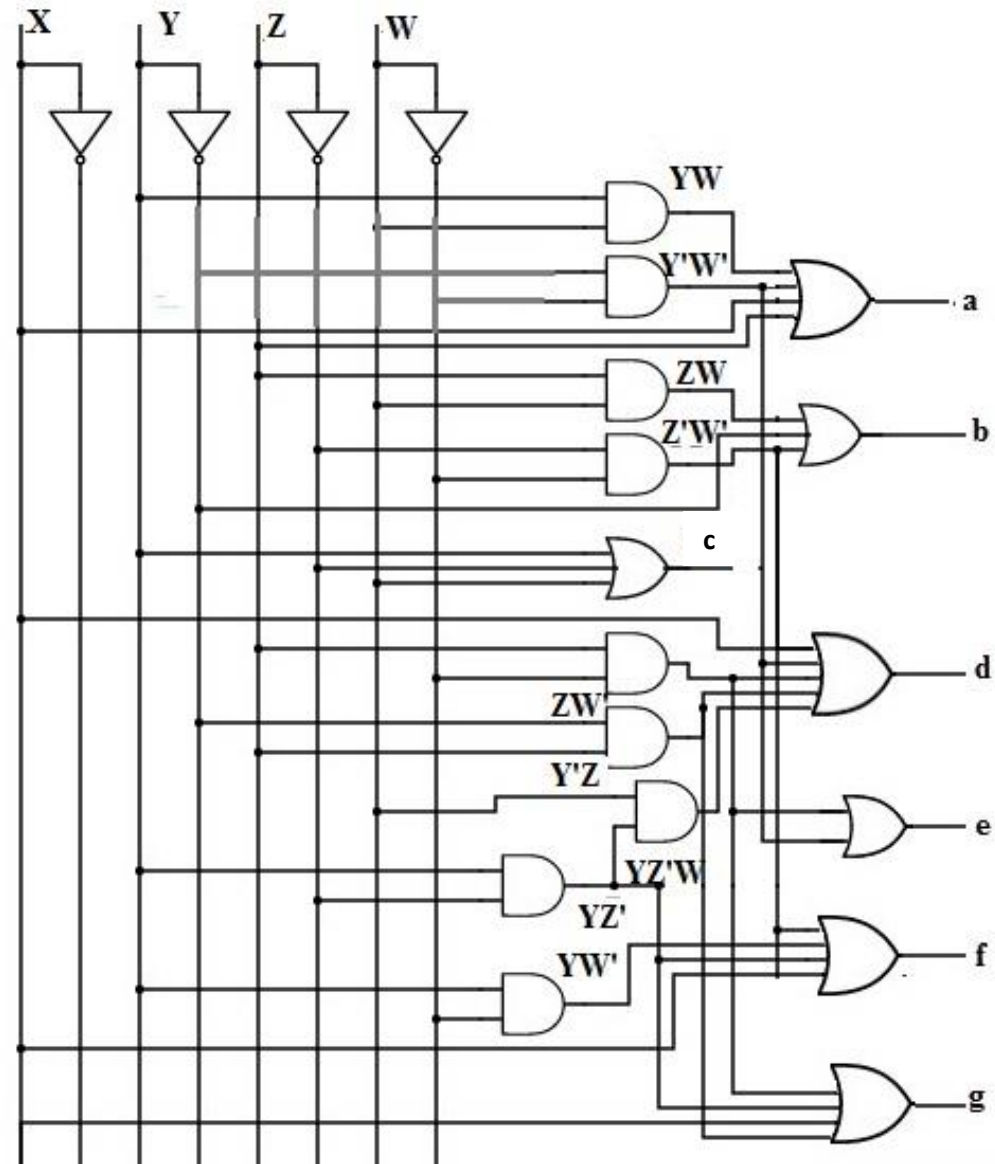
- Thus, the K-map for output  $c$  will look like this
- We have only one zero, so we go the PoS route
- The max term cluster of two is represented by  $yz'x'$
- In PoS form:

$$c = y' + z + x$$

| $wx \backslash yz$ |    | $y$           |               |               |               |
|--------------------|----|---------------|---------------|---------------|---------------|
|                    |    | 00            | 01            | 11            | 10            |
| $w$                | 00 | $m_0$<br>1    | $m_1$<br>1    | $m_3$<br>1    | $m_2$<br>0    |
|                    | 01 | $m_4$<br>1    | $m_5$<br>1    | $m_7$<br>1    | $m_6$<br>1    |
|                    | 11 | $m_{12}$<br>X | $m_{13}$<br>X | $m_{15}$<br>X | $m_{14}$<br>X |
|                    | 10 | $m_8$<br>1    | $m_9$<br>1    | $m_{11}$<br>X | $m_{10}$<br>X |
|                    |    | $z$           |               |               |               |

$x$

# 7-segment decoder

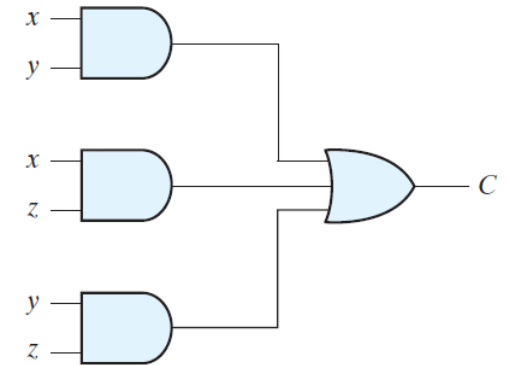
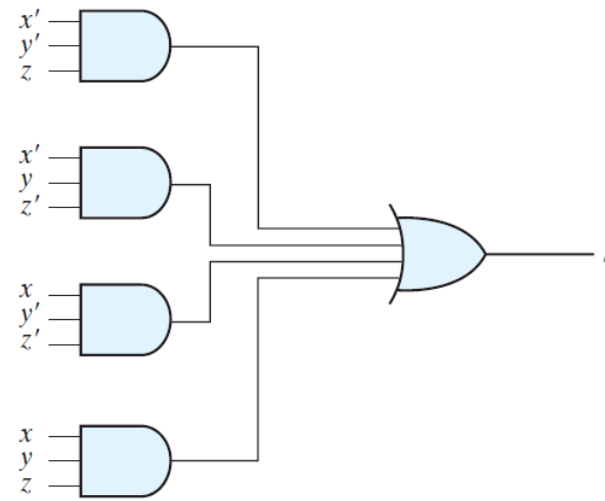
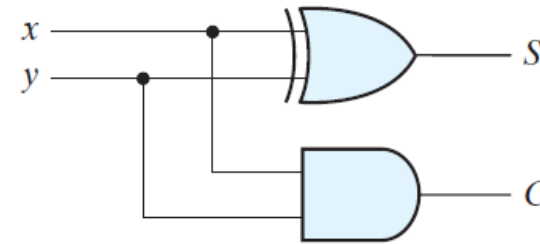




# *The Binary adder*

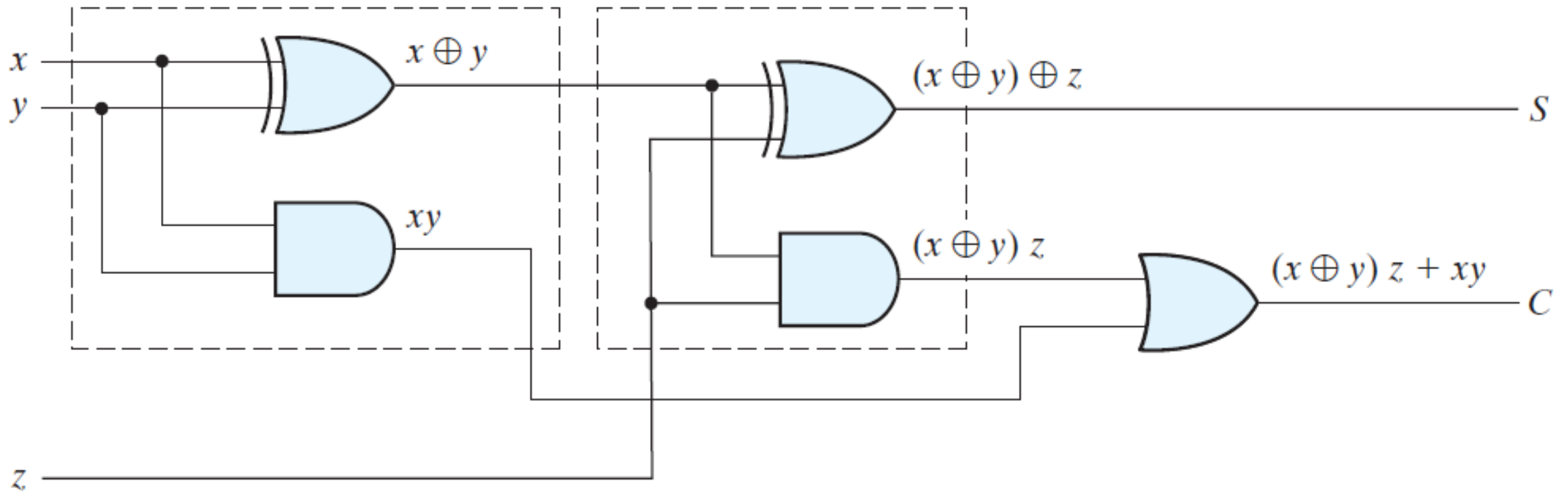
# Binary adder

- Digital computers perform a variety of information-processing tasks
- Among the functions encountered are the various arithmetic operations
- The most basic arithmetic operation is the addition of two binary digits
- A combinational circuit that performs the addition of two bits is called a *half adder*.
- One that performs the addition of three bits (two significant bits and a previous carry) is a *full adder*



# Binary adder

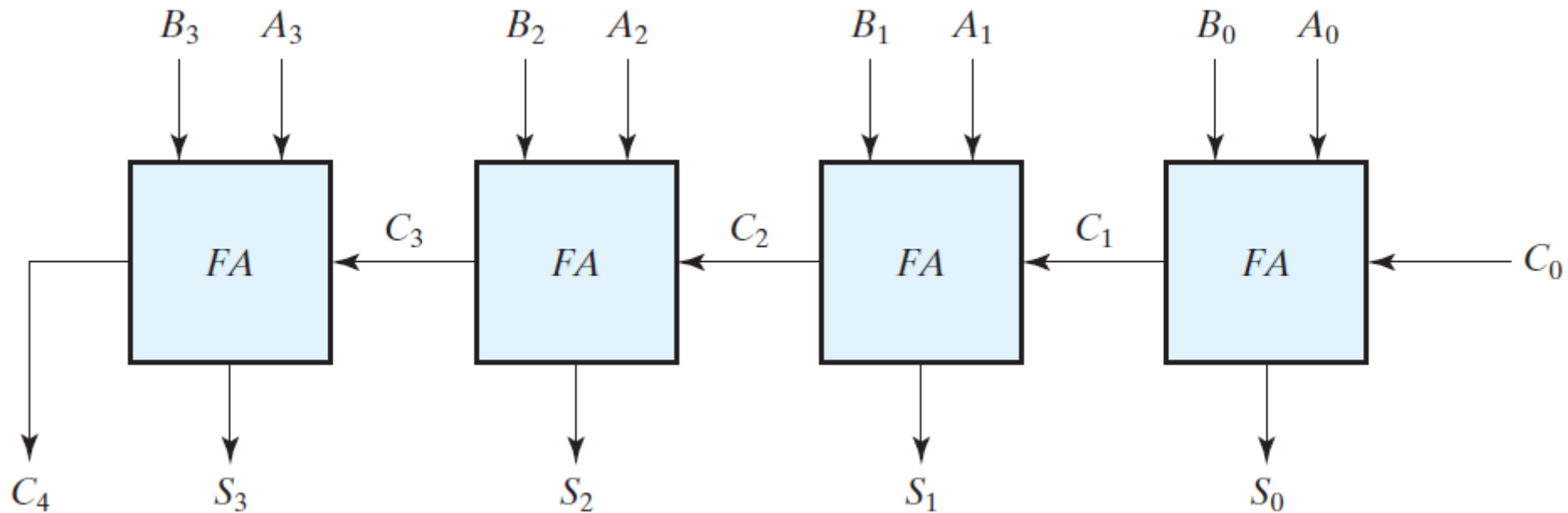
- We can use two half adders to create a full adder





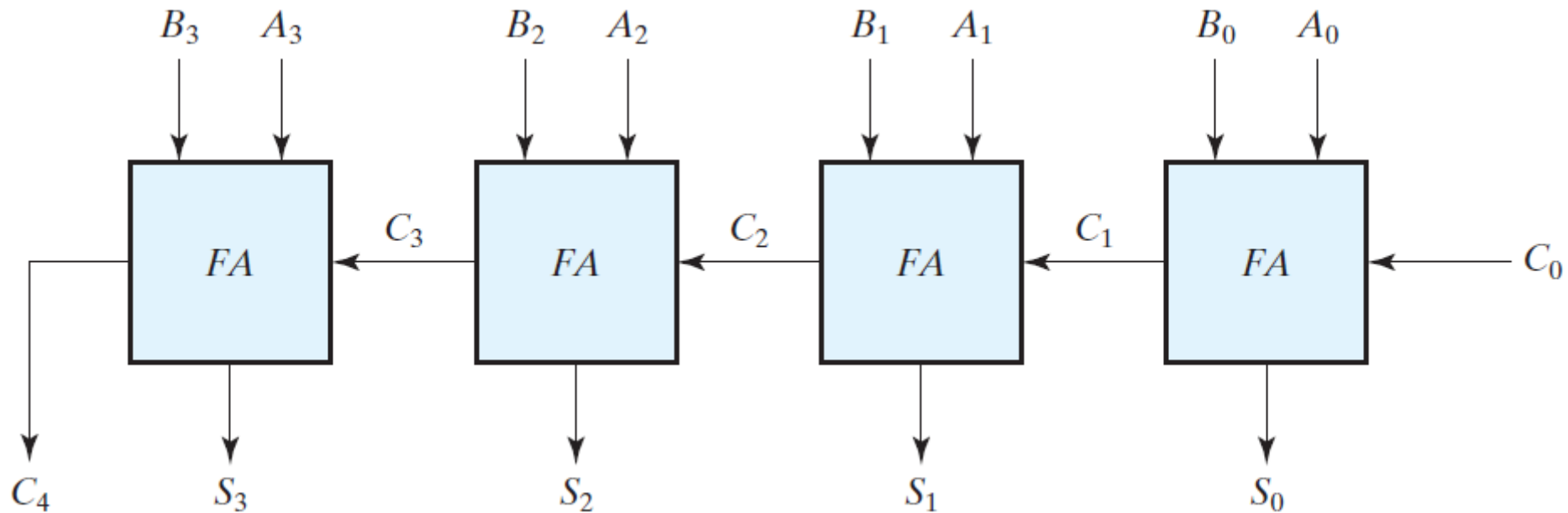
# n-bit binary adder

- Addition of  $n$ -bit numbers requires a chain of  $n$  full adders or a chain of one-half adder and  $n-1$  full adders
- Consider a four bit adder. The augend bits of  $A$  and the addend bits of  $B$  are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit
- The carries are connected in a chain through the full adders. The input carry to the adder is  $C_0$ , and it ripples through the full adders to the output carry  $C_4$ .
- The  $S$  outputs generate the required sum bits



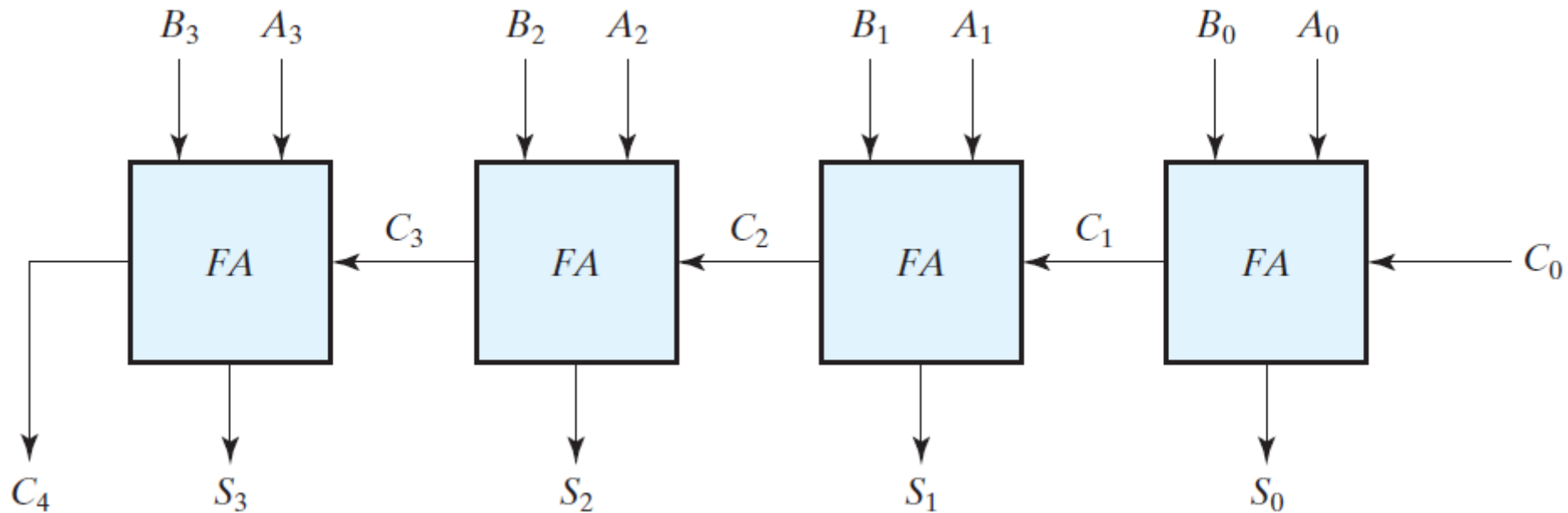
# n-bit binary adder

- Can we make this circuit through the normal route?
- Note that the classical method would require a truth table (and K-map) with  $2^9 = 512$  entries, since there are nine inputs to the circuit
- By using an iterative method of cascading a standard function, it is possible to obtain a simple and straightforward implementation



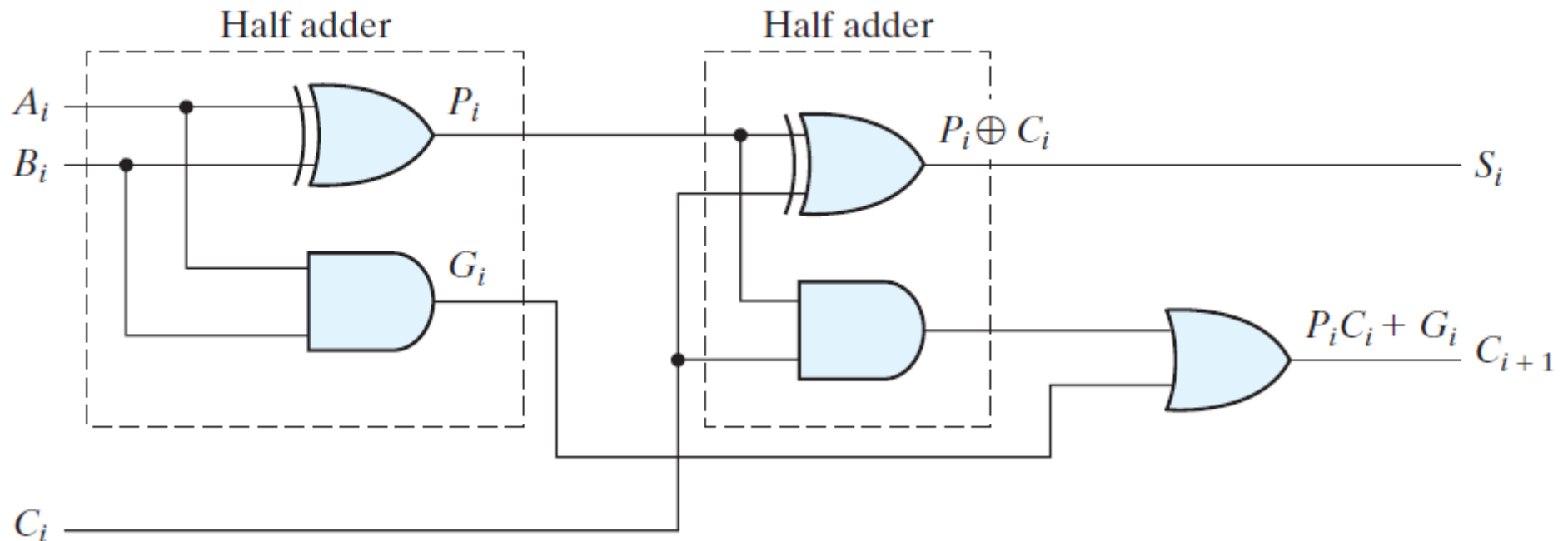
# Carry propagation problem

- The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time
- As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals
- The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders
- Since each bit of the sum output depends on the value of the input carry, the value of  $S_i$  at any given stage in the adder will be in its steady-state final value only after the input carry to that stage has been propagated



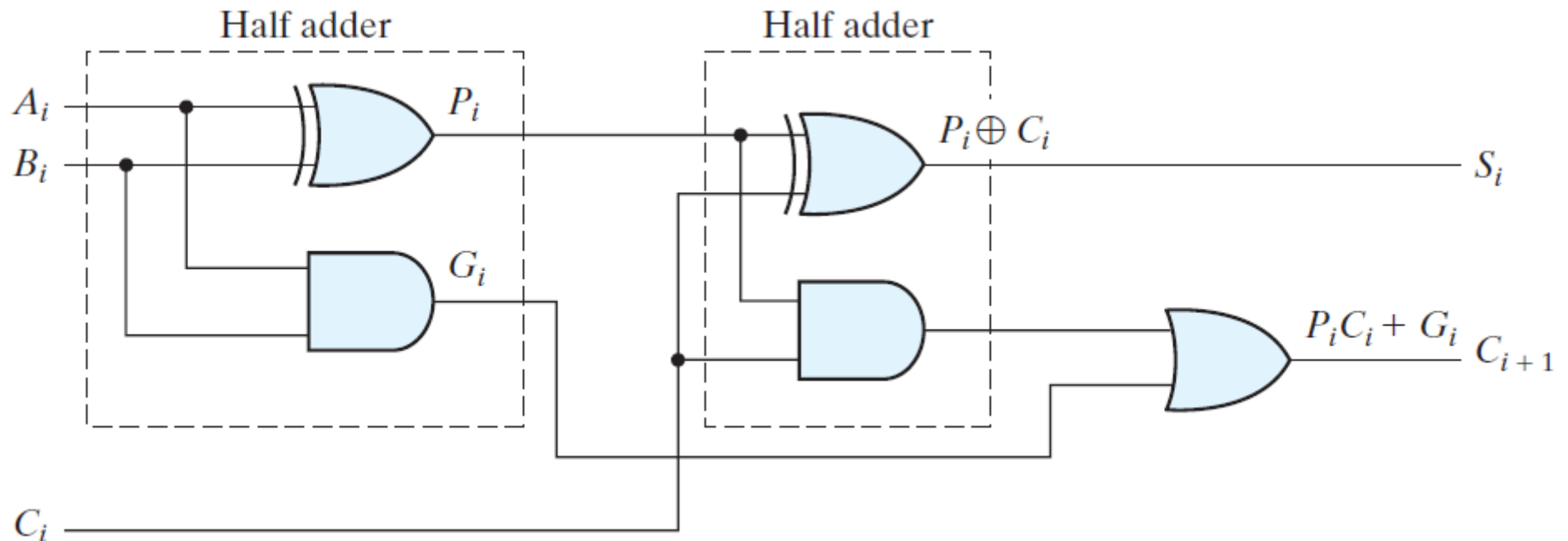
# Carry propagation problem

- The number of gate levels for the carry propagation can be found from the circuit of the full adder
- The signals at  $P_i$  and  $G_i$  settle to their steady-state values after they propagate through their respective gates
- These two signals are common to all half adders and depend on only the input augend and addend bits
- The signal from the input carry  $C_i$  to the output carry  $C_{i+1}$  propagates through an AND gate and an OR gate, which constitute two gate levels
- If there are four full adders in the adder, the output carry  $C_4$  would have  $2 * 4 = 8$  gate levels from  $C_0$  to  $C_4$
- For an  $n$ -bit adder, there are  $2n$  gate levels for the carry to propagate from input to output



# Carry propagation problem

- There are several techniques for reducing the carry propagation time in a parallel adder
- An obvious solution to this problem is to actually make the  $2^n$  truth-table, K-map and get a two level implementation (either SoP or PoS)
- The most widely used technique employs the principle of *carry lookahead logic*

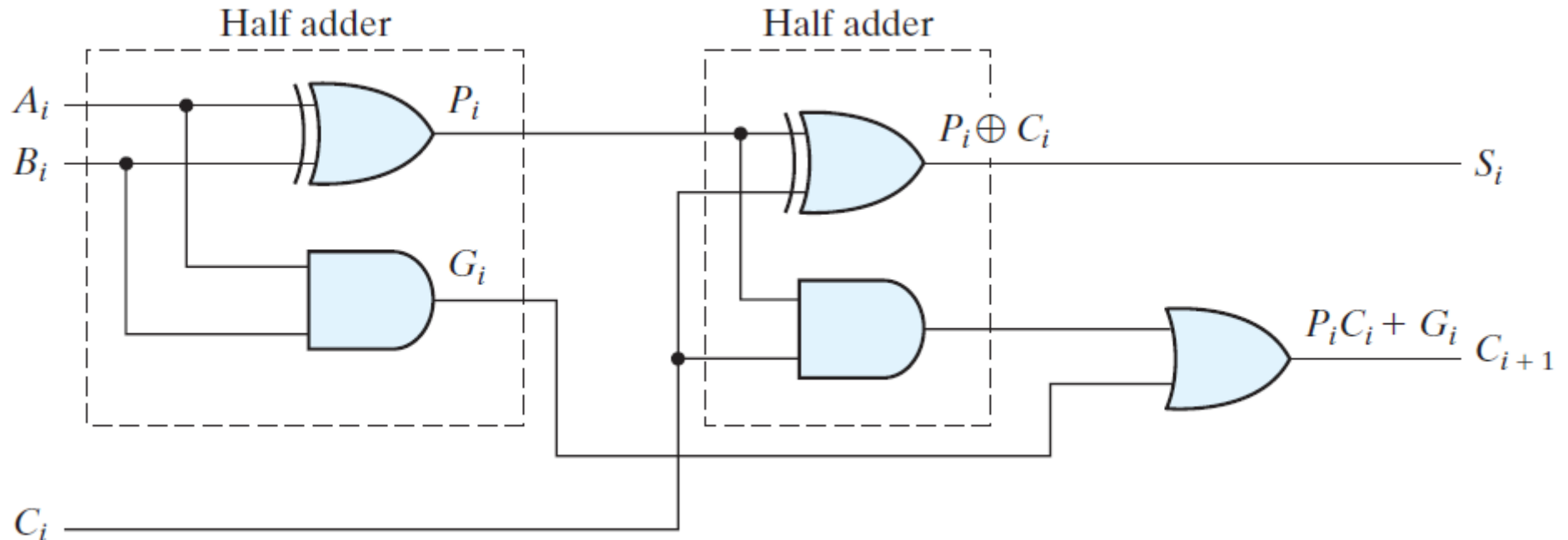


# Carry propagation problem

- With the definition of P and G, we can write:

$$S_i = P_i + C_i \text{ and } C_{i+1} = G_i + P_i C_i$$

- $G_i$  is called a *carry generate*, and it produces a carry of 1 when both  $A_i$  and  $B_i$  are 1, regardless of the input carry  $C_i$
- $P_i$  is called a *carry propagate*, because it determines whether a carry into stage  $i$  will propagate into stage  $i + 1$  (i.e., whether an assertion of  $C_i$  will propagate to an assertion of  $C_{i+1}$ )



# Carry propagation problem

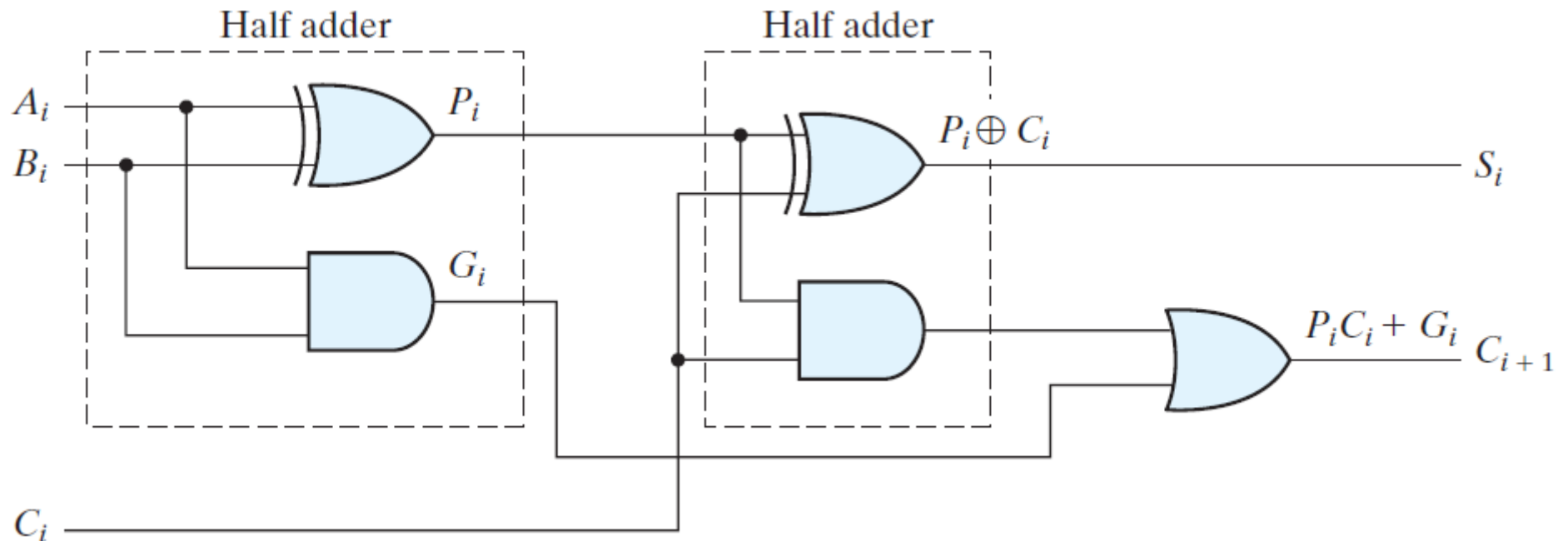
- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each  $C_i$  from the previous equations:

$$C_0 = \text{input carry}$$

$$C_1 = G_0 + P_0C_0$$

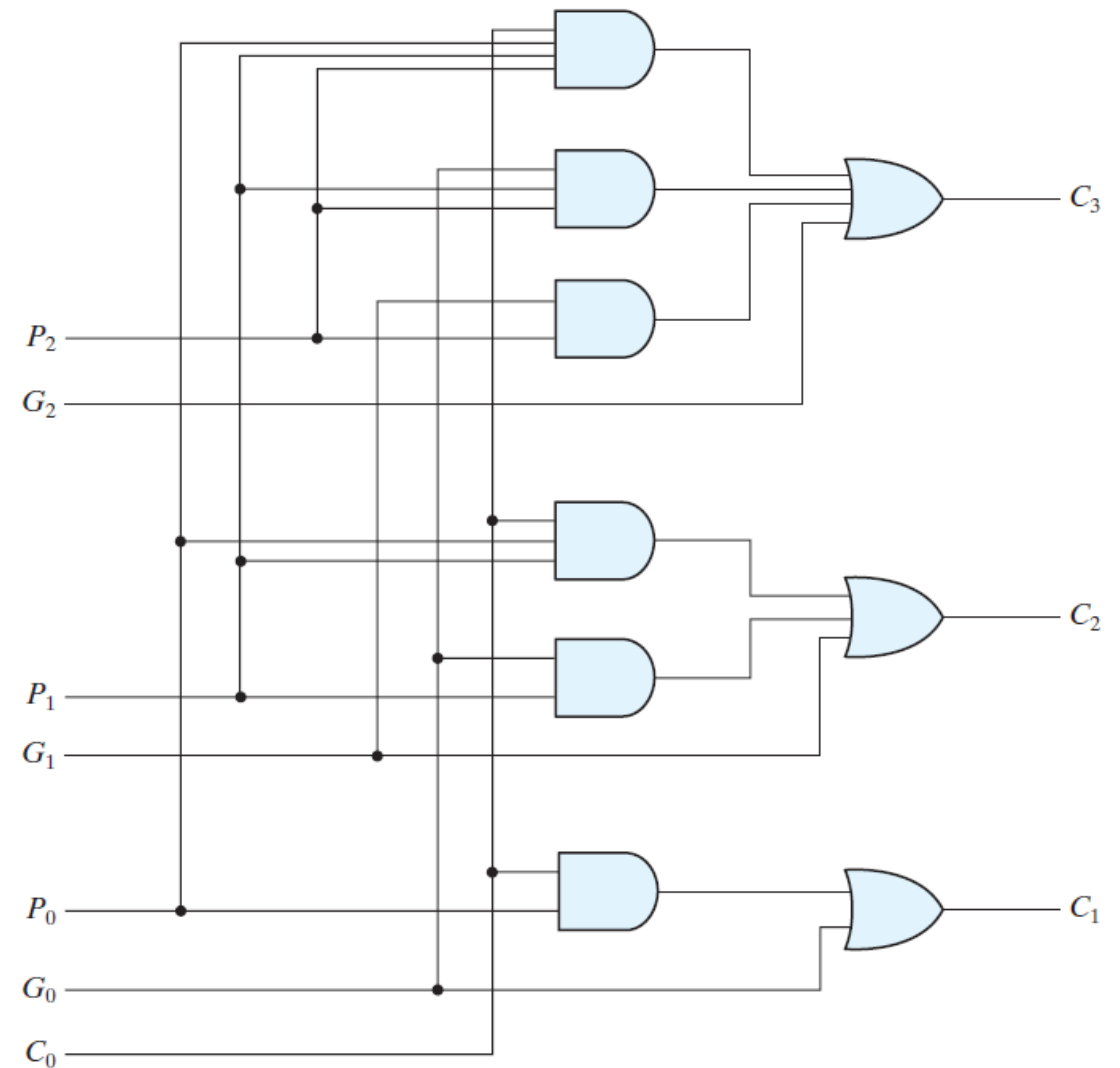
$$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$



# Carry propagation problem

- Since the Boolean function for each output carry is expressed in sum-of-products form only dependent on  $P$  and  $G$ , each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND)
- Note that this circuit can add in less time because  $C_3$  does not have to wait for  $C_2$  and  $C_1$  to propagate; in fact,  $C_3$  is propagated at the same time as  $C_1$  and  $C_2$
- This gain in speed of operation is achieved at the expense of additional complexity (hardware)





# Carry propagation problem

- We can make the four bit adder as shown
- Each sum output requires two XOR gates
- The output of the first XOR gate generates the  $P_i$  variable, and the AND gate generates the  $G_i$  variable
- The carries are propagated through the carry lookahead generator and applied as inputs to the second XOR gate
- All output carries are generated after a delay through only two levels of gates
- Thus, outputs  $S_1$  through  $S_3$  have equal propagation delay times

