

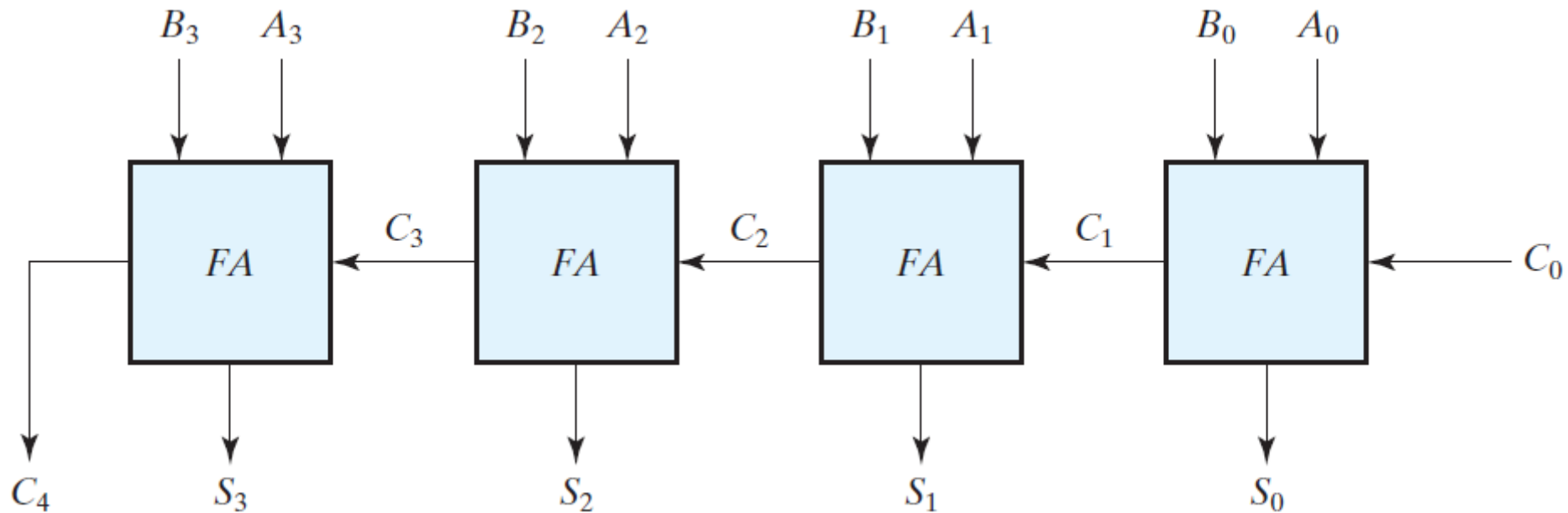
# Lecture 14 – Combinational circuits 3

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

## Chapter 4

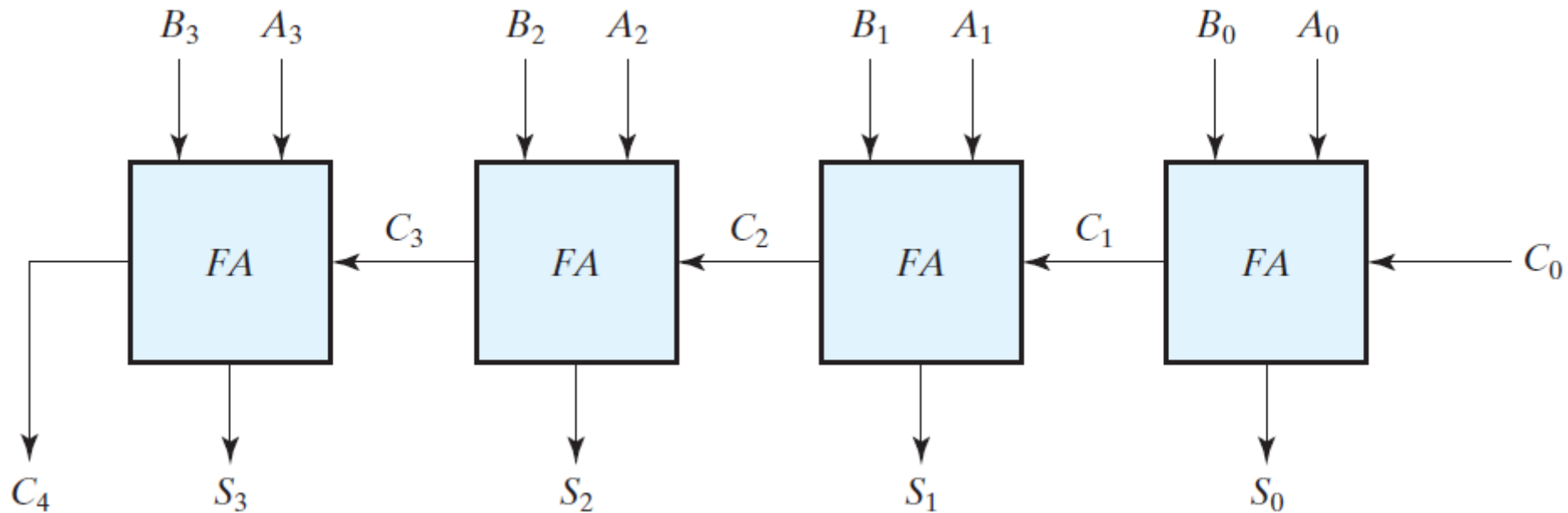
# 4-bit binary adder

- 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



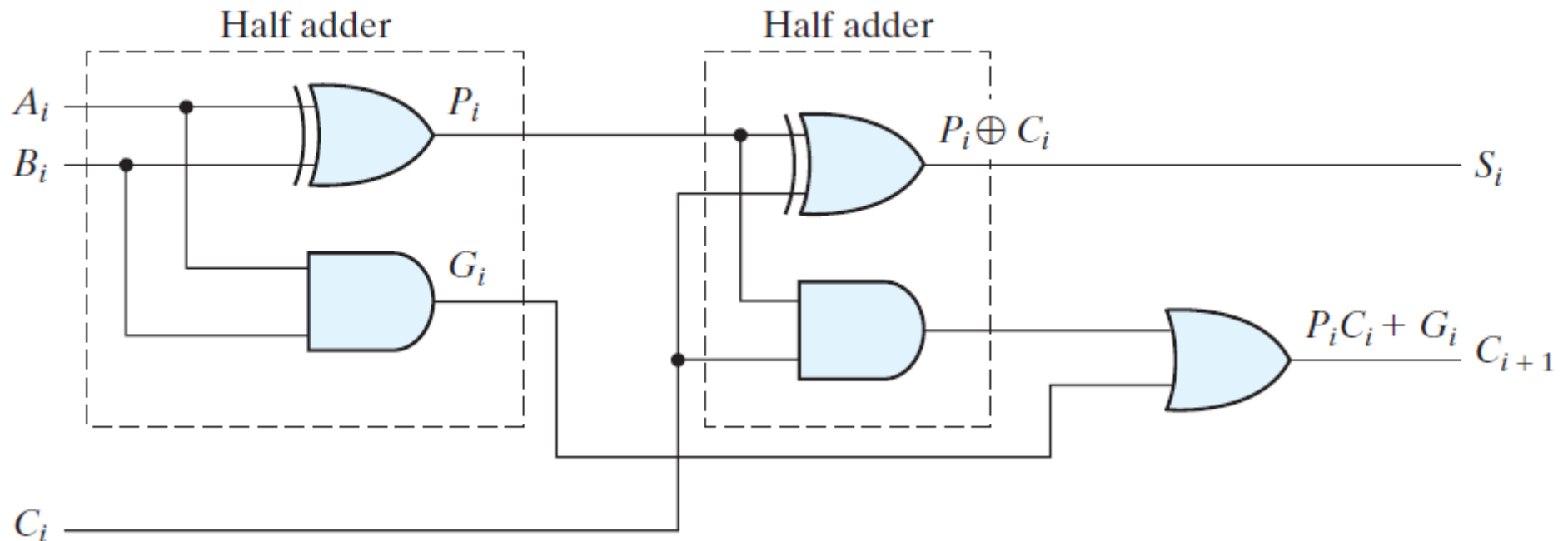
# Carry propagation problem

- 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



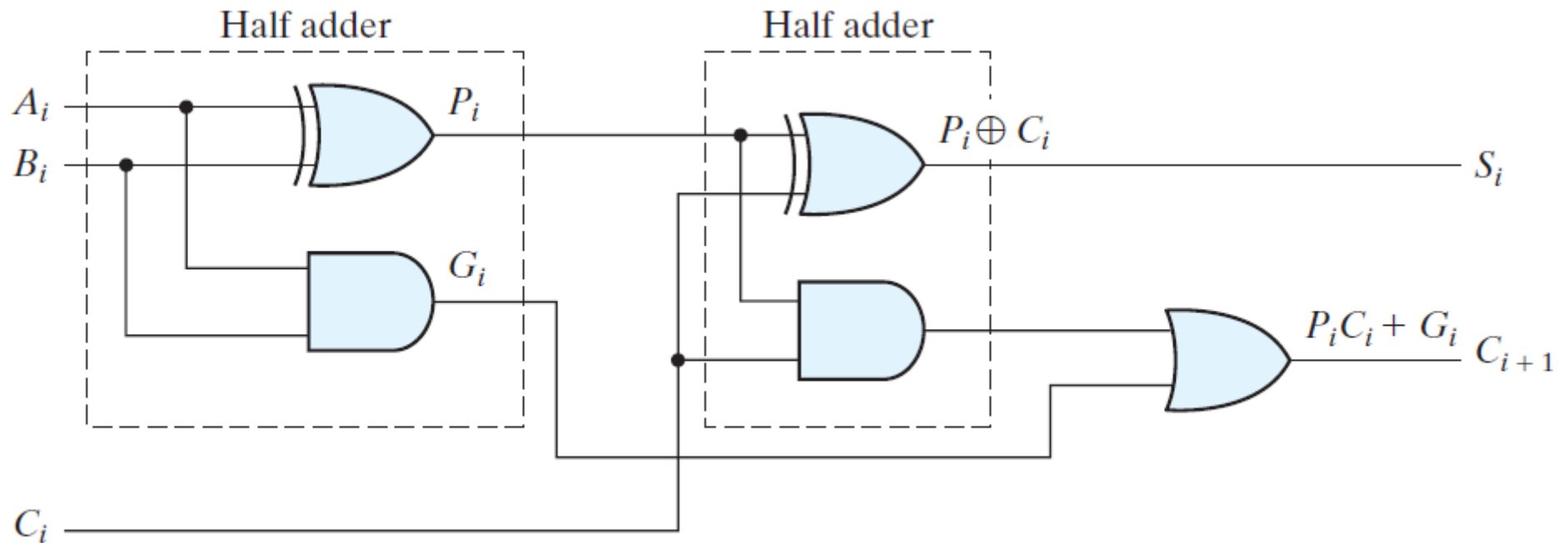
# Carry propagation problem

- 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*

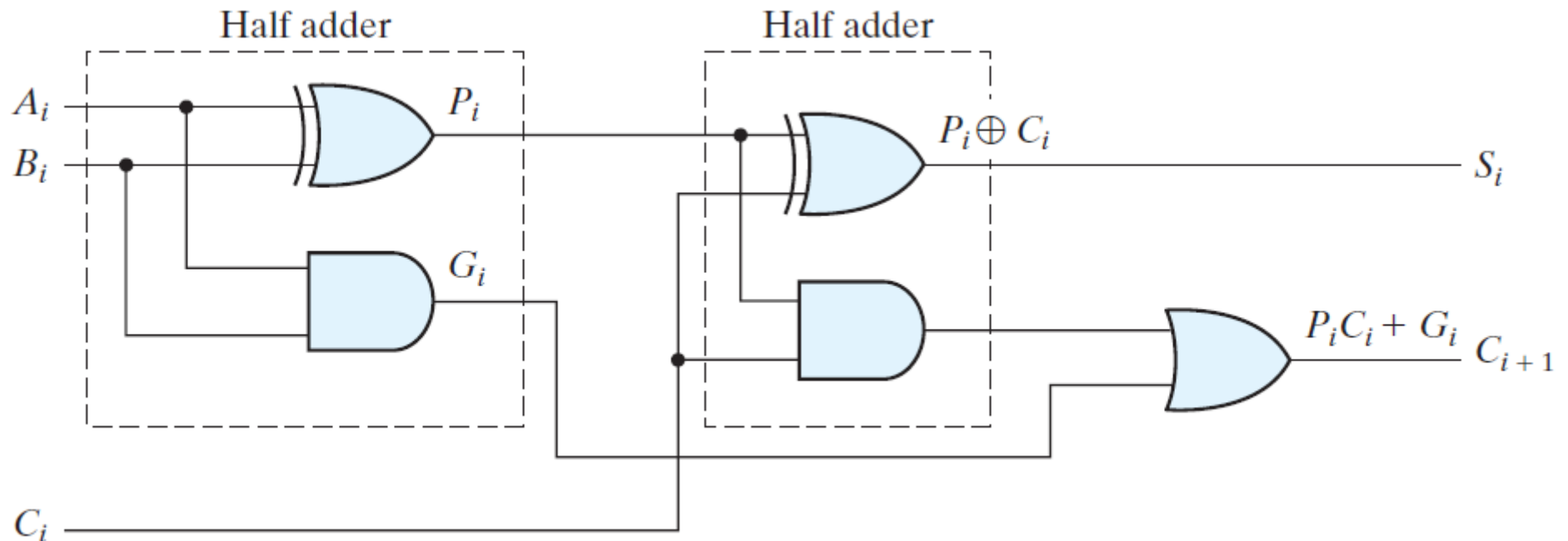


# Carry propagation problem

- 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

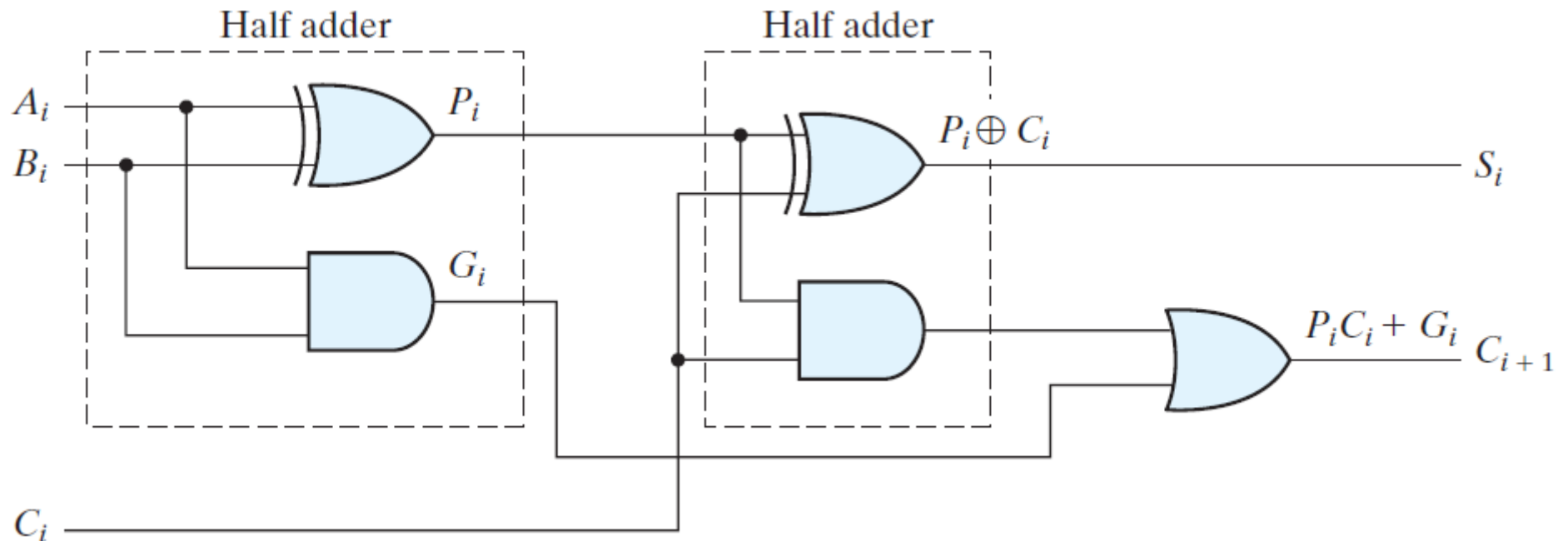
- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each  $C_i$  from the previous equations:

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

$$C_1 = G_0 + P_0C_0$$

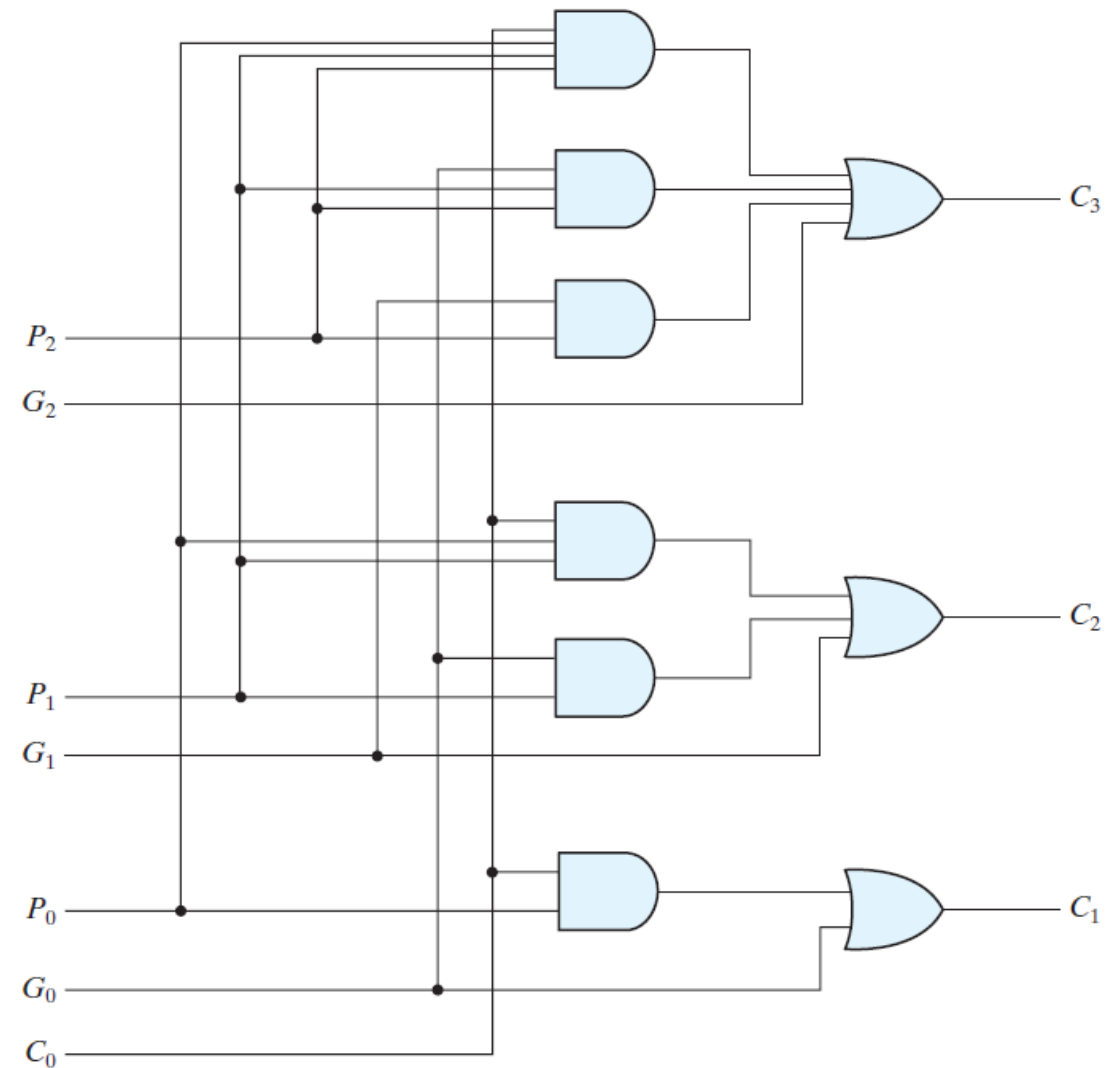
$$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$$

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



# Carry propagation problem

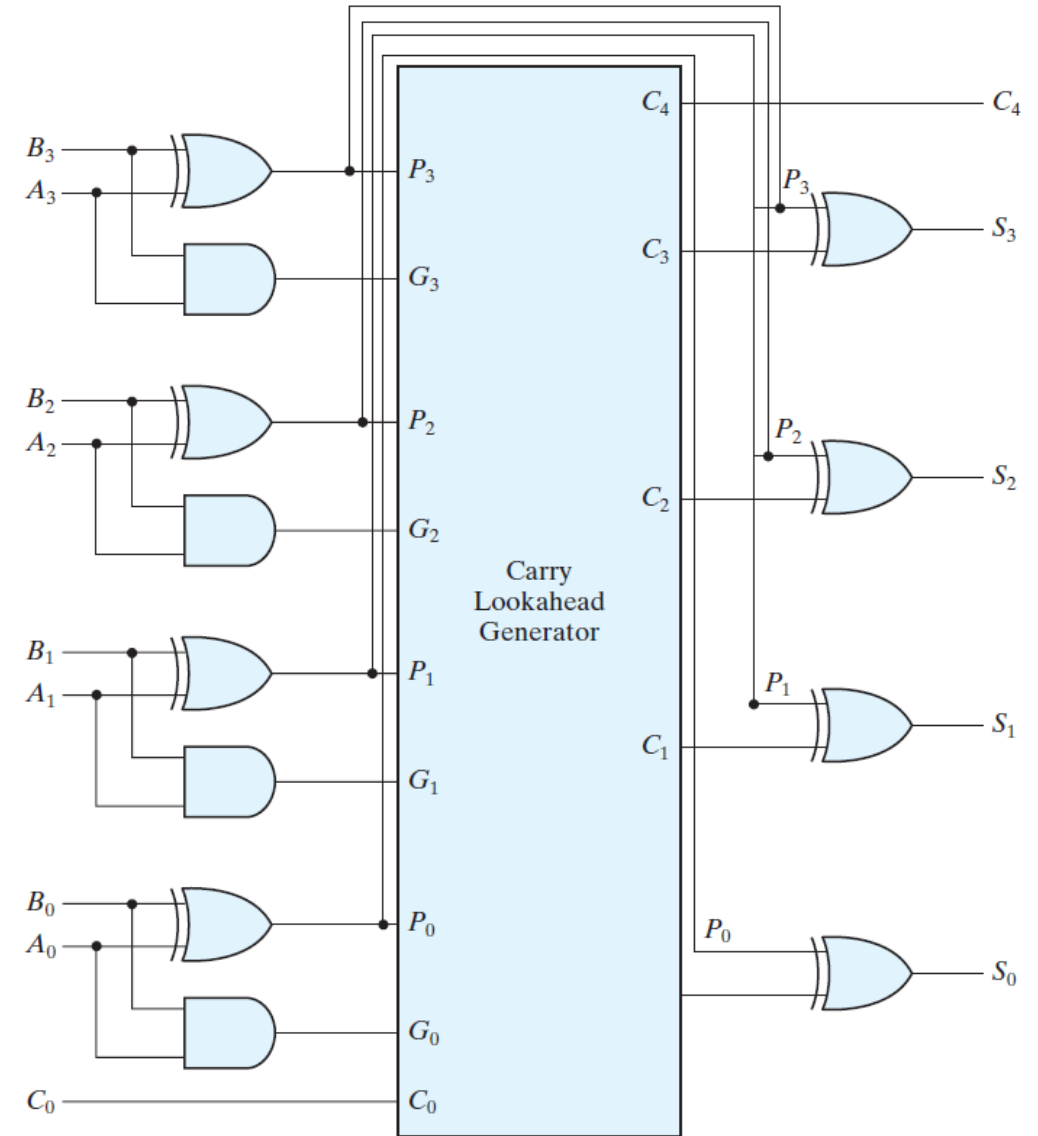
- Since the Boolean function for each output carry is expressed in sum-of-products form only dependent on  $P$  and  $G$ , each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND)
- Note that this circuit can add in less time because  $C_3$  does not 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





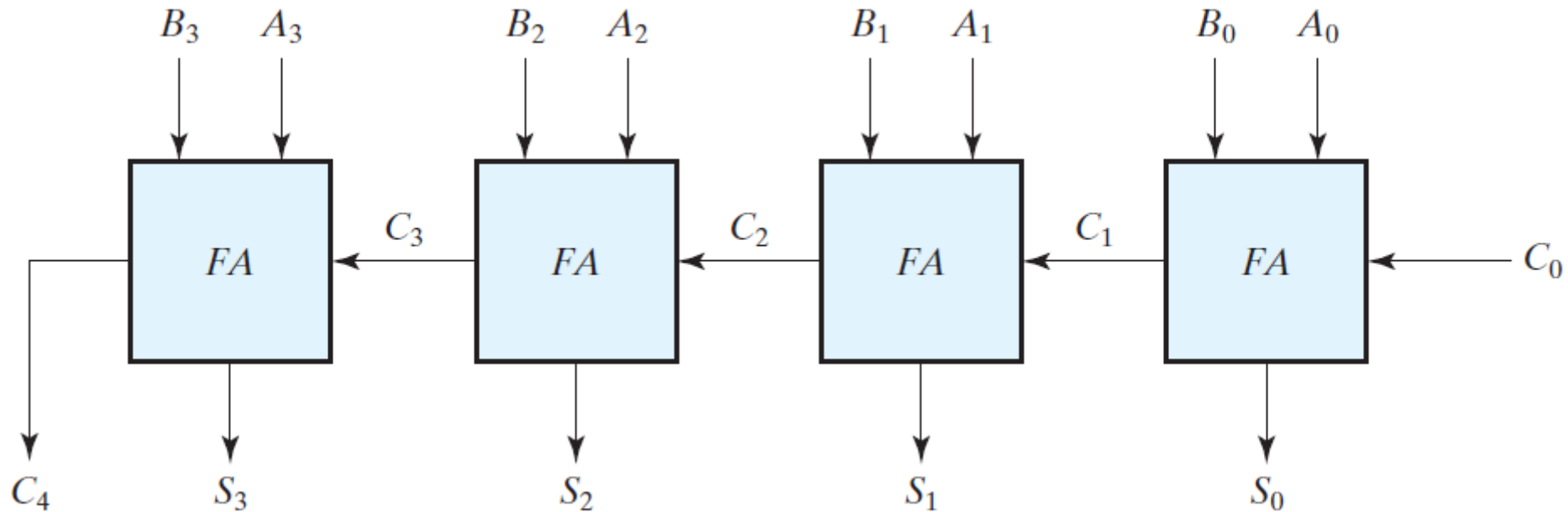
# *The Binary Subtractor*

# Binary subtractor

- The subtraction of unsigned binary numbers can be done most conveniently by means of complements
- Remember that the subtraction  $A - B$  can be done by taking the 2's complement of  $B$  and **adding** it to  $A$
- The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits
- The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry
- The circuit for subtracting  $A - B$  consists of an adder with inverters placed between each data input  $B$  and the corresponding input of the full adder
- **The input carry  $C_0$  must be equal to 1 when subtraction is performed**
- The operation thus performed becomes  $A$ , plus the 1's complement of  $B$ , plus 1. This is equal to  $A$  plus the 2's complement of  $B$
- That gives  $A - B$  if  $A \geq B$  or the 2's complement of  $B - A$  if  $A < B$

# Can we combine the binary adder & subtractor

- Both use the 4-bit full adder. In one case, we use B and in another we use inverted B



# Binary adder-subtractor

- Here is some magic: The addition and subtraction operations can be combined into one circuit
- The mode input  $M$  controls the operation
- When  $M = 0$ , the circuit is an adder, and when  $M = 1$ , the circuit becomes a subtractor
- When  $M = 0$ , the full adders receive the value of  $B$ , the input carry is 0, and the circuit performs  $A + B$
- When  $M = 1$ , the full adders receive  $B'$  and  $C_0 = 1$
- Thus, the  $B$  inputs are all complemented and a 1 is added through the input carry
- The circuit performs the operation  $A$  plus the 2's complement of  $B$

# Binary adder-subtractor

