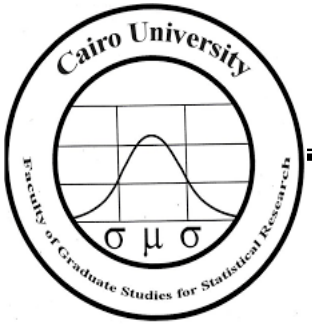


Computer Science Department

Cairo University

CS504

Digital Logic & Computer Organization

Lecture 7

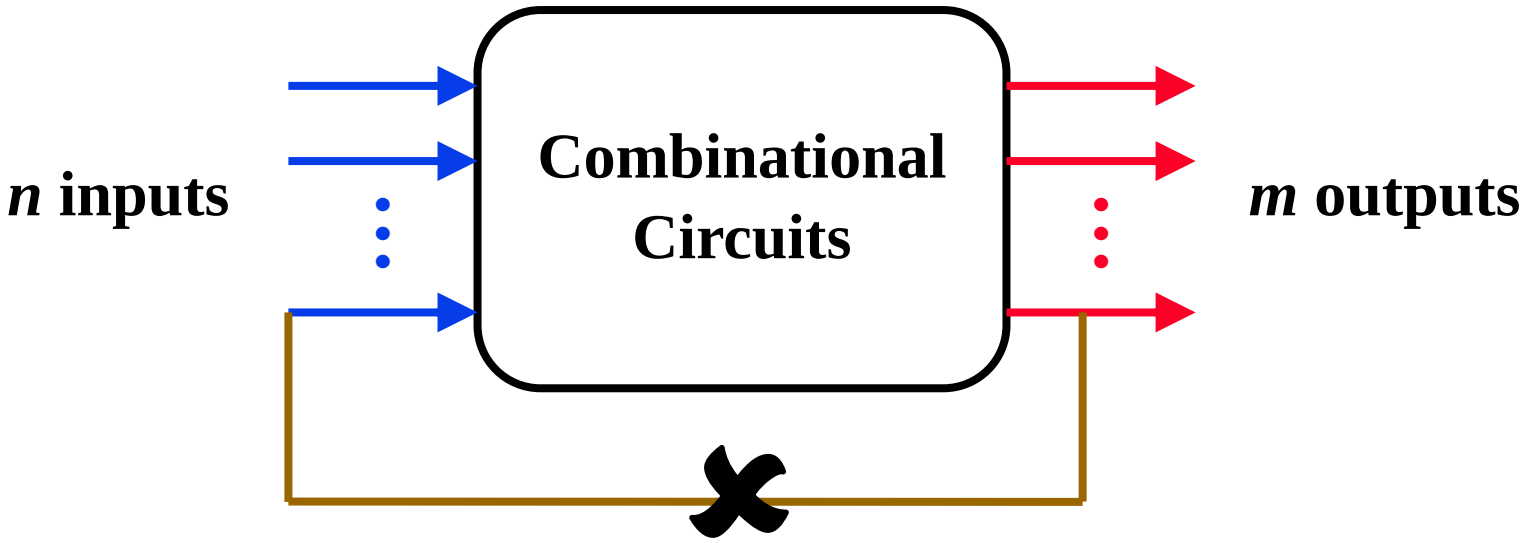
Lecture Outline (Chapter 4)

- ★ Combinational Circuits (Section 4.2)
- ★ Analysis Procedure (Section 4.3)
- ★ Design Procedure (Section 4.4)
  - BCD-to-Excess-3 Converter
  - Seven-Segment Decoder
- ★ Binary Adder (Section 4.5)
  - Half-Adder
  - Full Adder
  - Carry Propagate Adder
  - Carry Propagation
  - Carry Look-Ahead Generator
  - 4-bit Carry Look-Ahead Adder (CLAA)

Combinational Circuits

★ **Output is function of input only**

i.e. no feedback

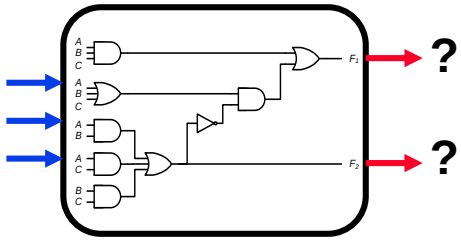


When input changes, output may change (after a delay)

Combinational Circuits (2)

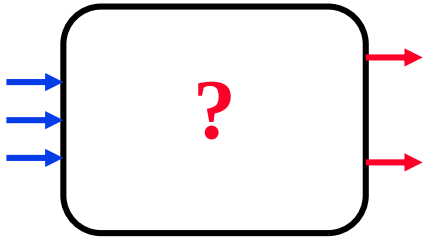
★ Analysis

- Given a circuit, find out its *function*
- Function may be expressed as:
  - ♦ Boolean function
  - ♦ Truth table



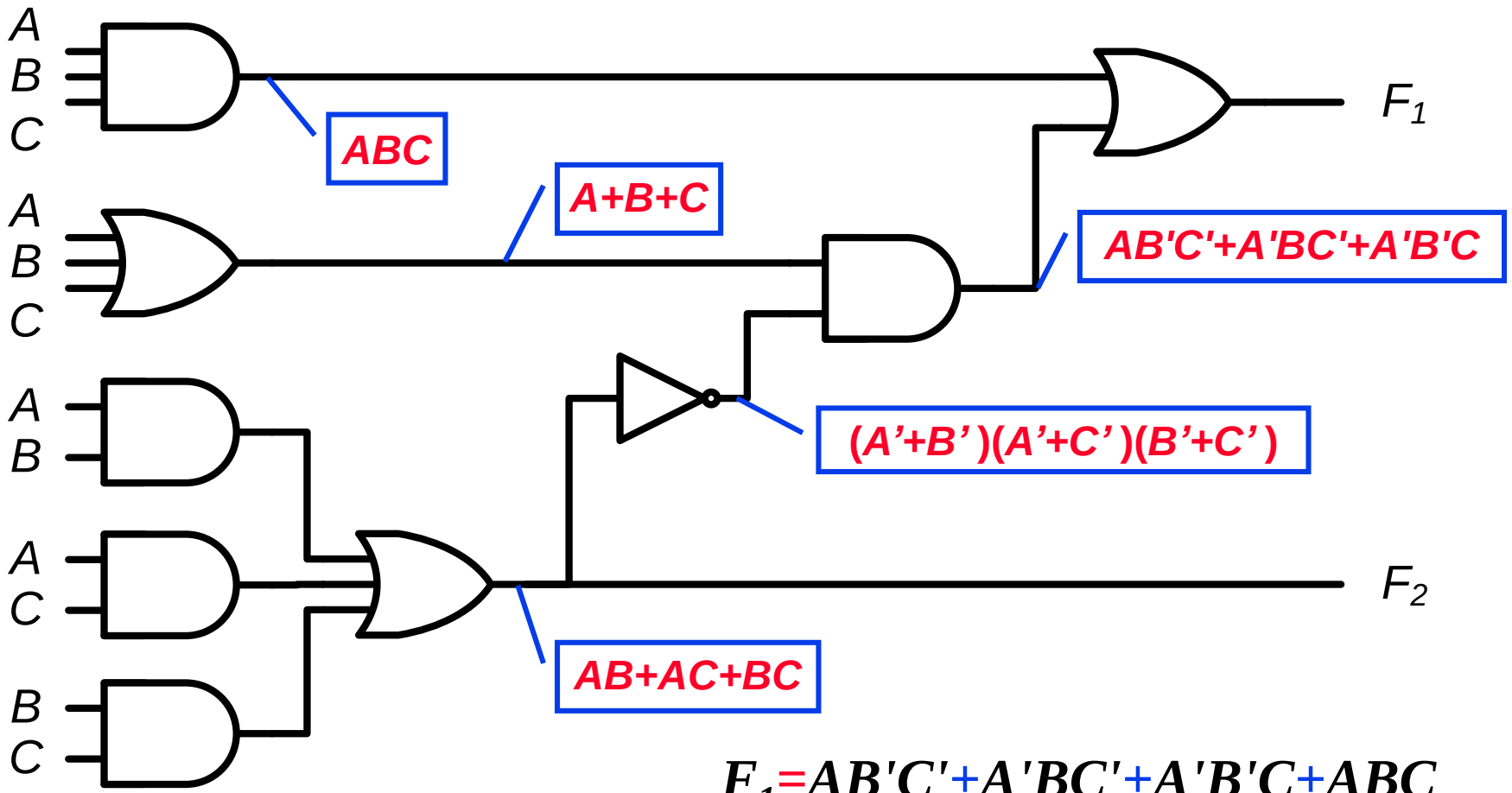
★ Design

- Given a desired function, determine its *circuit*
- Function may be expressed as:
  - ♦ Boolean function
  - ♦ Truth table



Analysis Procedure

★ Boolean Expression Approach

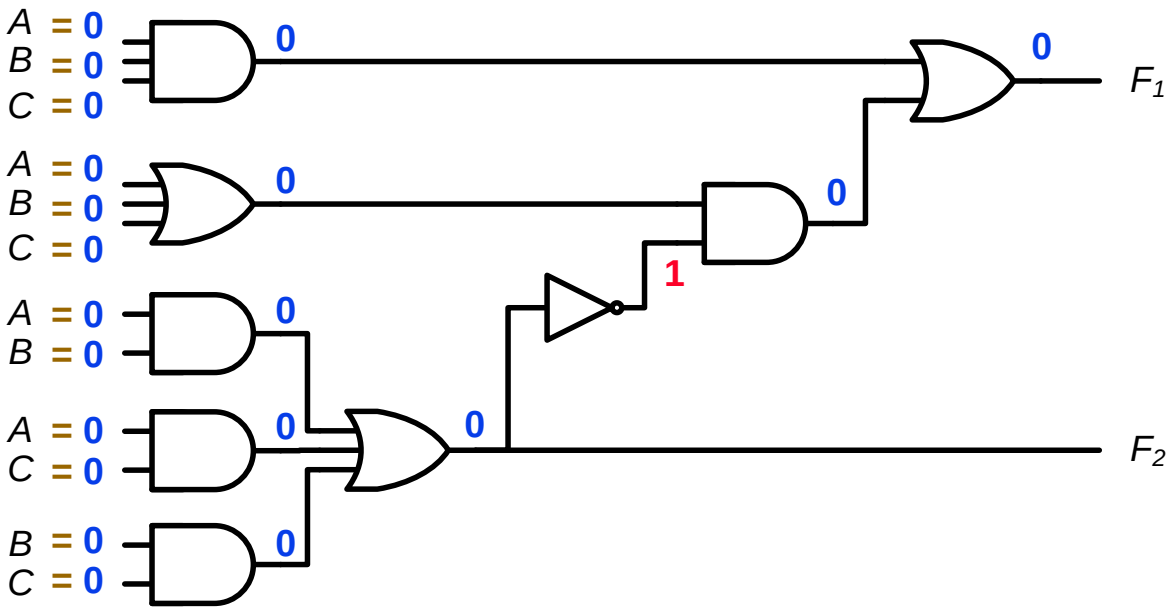


$F_1 = AB'C' + A'BC' + A'B'C + ABC$

$F_2 = AB + AC + BC$

Analysis Procedure (2)

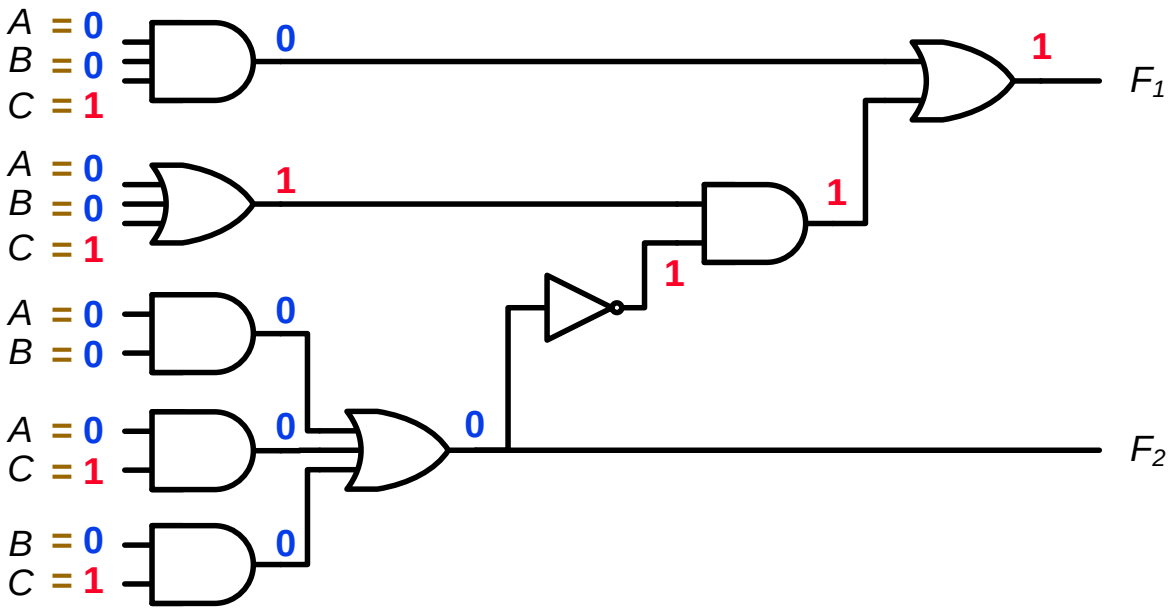
★ Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0

Analysis Procedure (3)

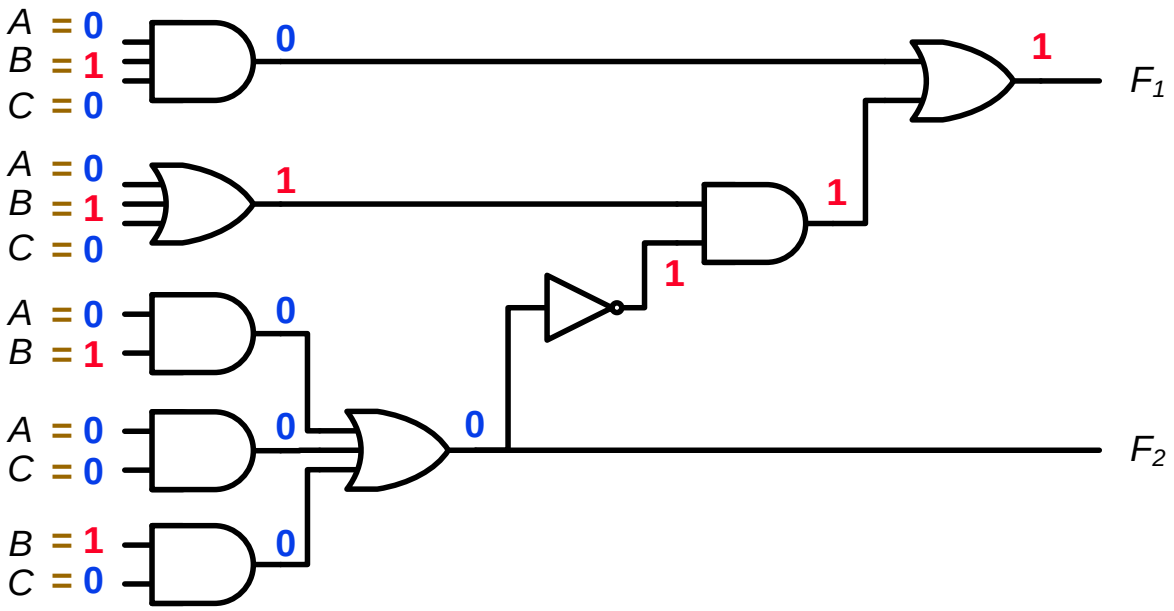
★ Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0

Analysis Procedure (4)

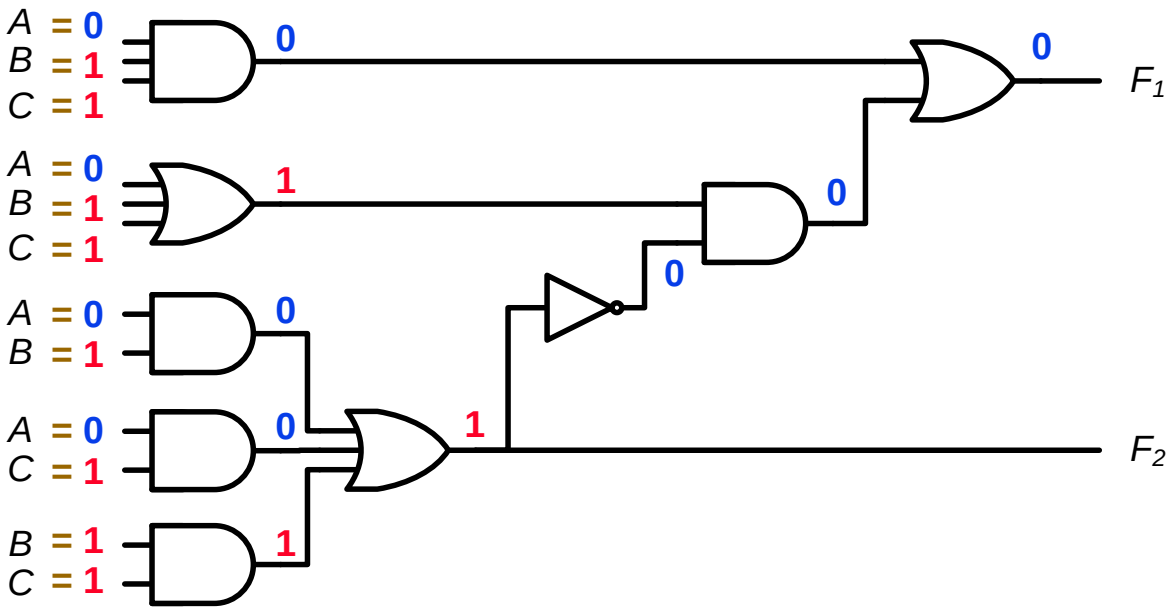
★ Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0

Analysis Procedure (5)

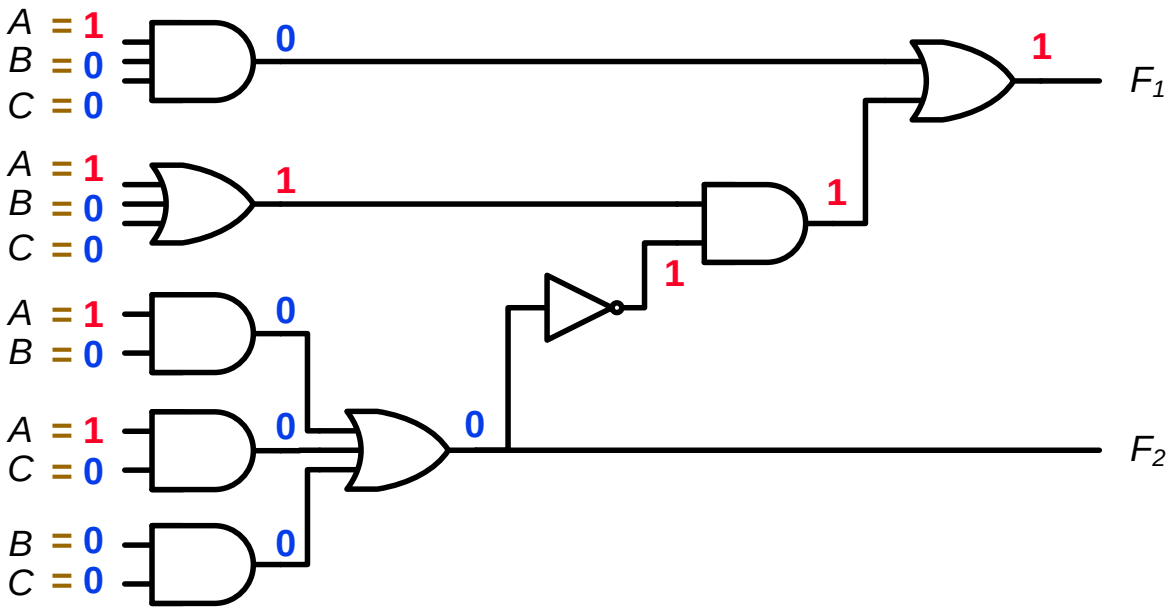
★ Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1

Analysis Procedure (6)

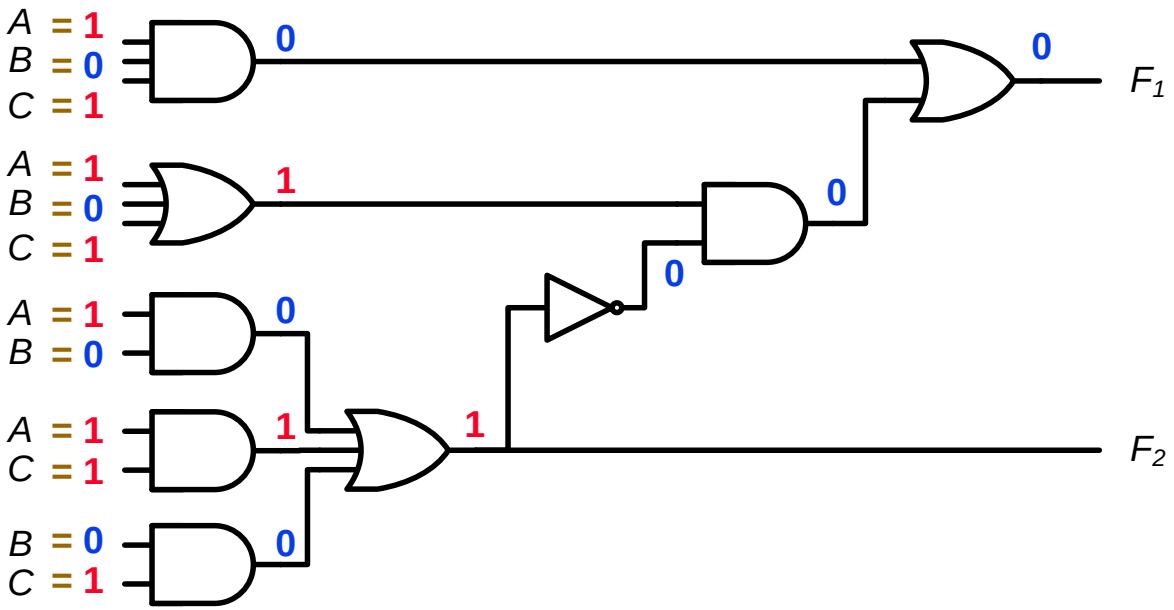
★ Truth Table Approach



A	B	C	F_1	F_2
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

Analysis Procedure (7)

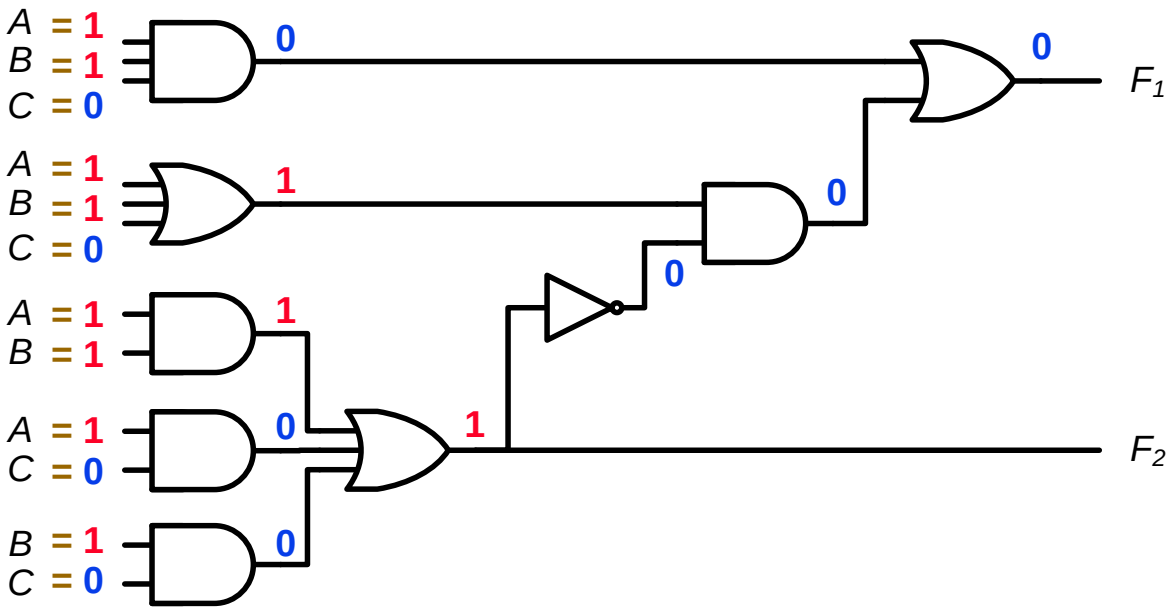
★ Truth Table Approach



A	B	C	F_1	F_2
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

Analysis Procedure (8)

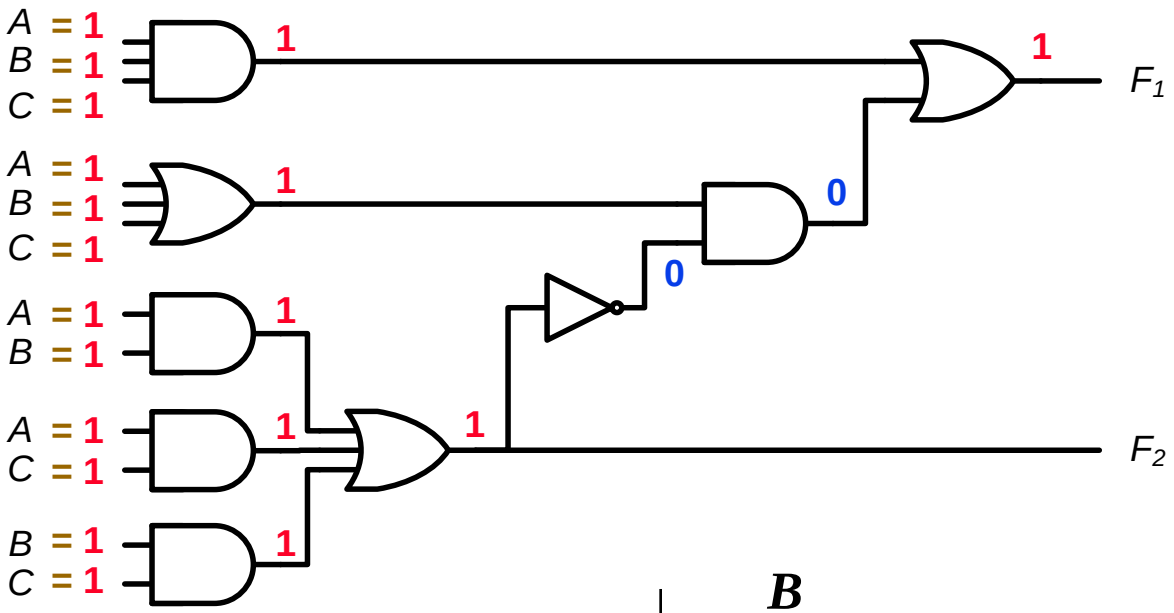
★ Truth Table Approach



A	B	C	F_1	F_2
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

Analysis Procedure (9)

★ Truth Table Approach



A B C	F_1	F_2
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

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$F_1 = AB'C' + A'BC' + A'B'C + ABC$

$F_2 = AB + AC + BC$

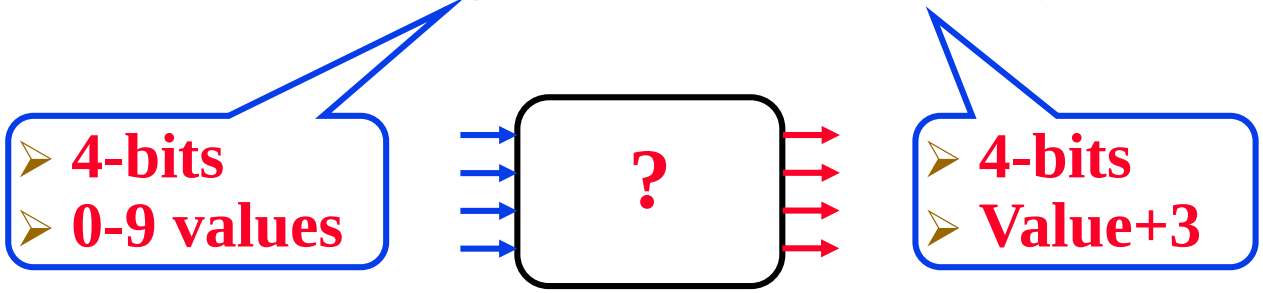
Design Procedure

★ Given a problem statement:

- Determine the number of *inputs* and *outputs*
- Derive the truth table
- Simplify the Boolean expression for each output
- Produce the required circuit

Example:

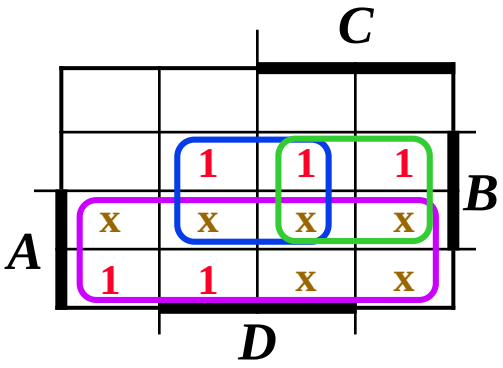
Design a circuit to convert a "BCD" code to "Excess 3" code



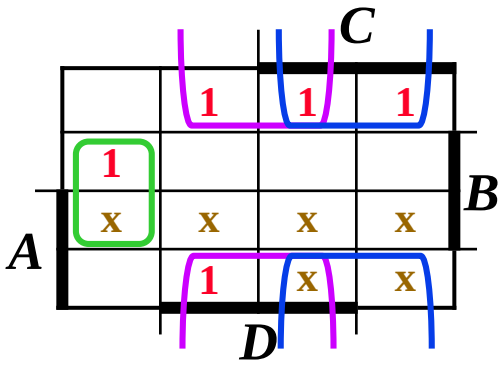
Design Procedure (2)

★ BCD-to-Excess-3 Converter

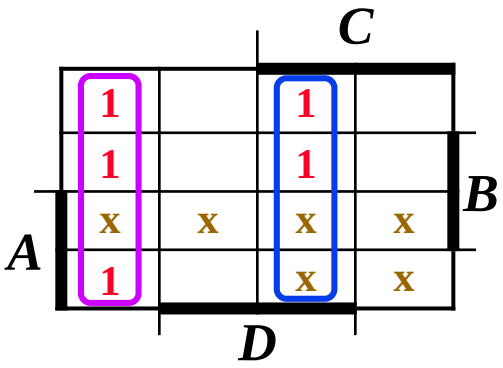
A B C D	w x y z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0
1 0 1 0	x x x x
1 0 1 1	x x x x
1 1 0 0	x x x x
1 1 0 1	x x x x
1 1 1 0	x x x x
1 1 1 1	x x x x



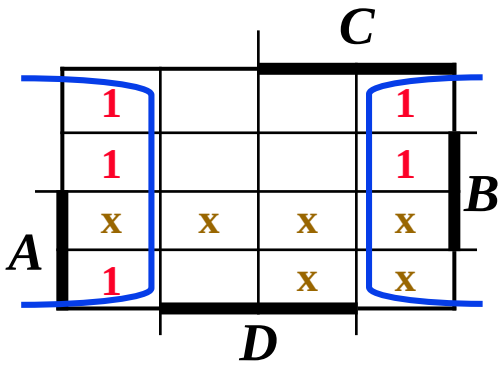
$w = A + BC + BD$



$x = B'C + B'D + BC'D'$



$y = C'D' + CD$



$z = D'$

Design Procedure (3)

★ The simplified functions in SOP form

$z = D'$

$y = CD + C'D'$

$x = B'C + B'D + BC'D'$

$w = A + BC + BD$

This two-level implementation requires 7 AND gates, 3 OR gates and 3 inverter gates for variables *B, C, and D.*

★ Another implementation by algebraic manipulation

$z = D'$

$y = CD + C'D' = CD + (C+D)'$

$x = B'C + B'D + BC'D' = B'(C+D) + B(C+D)'$

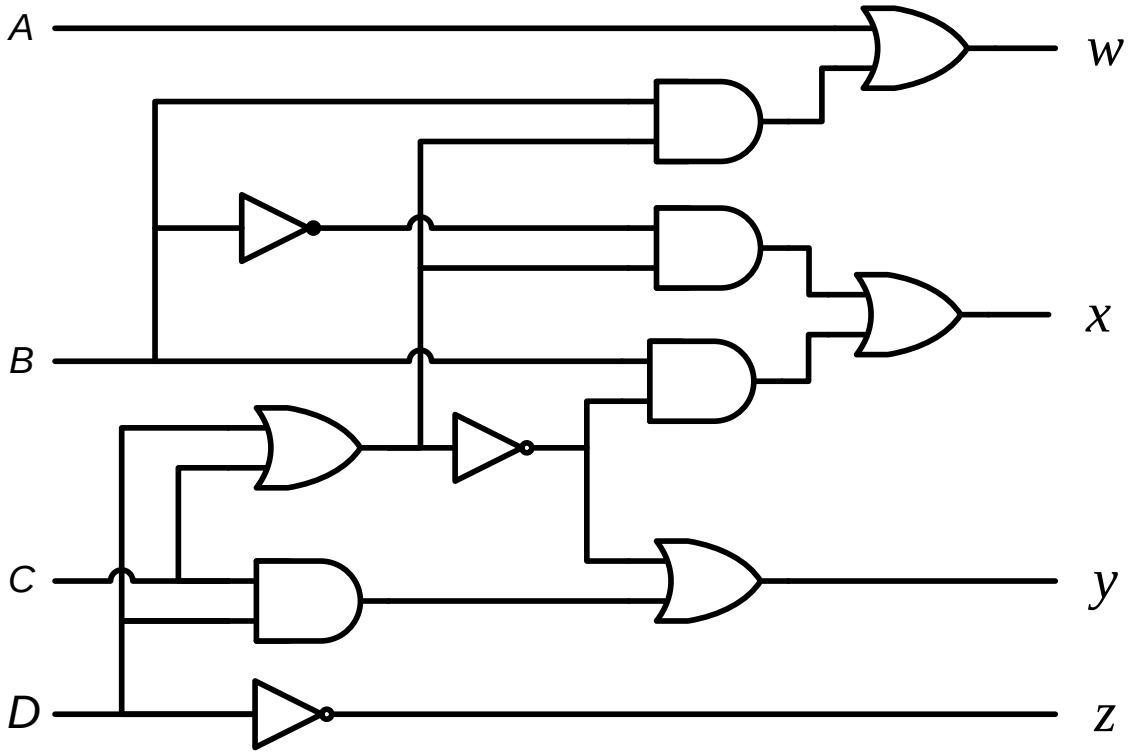
$w = A + BC + BD = A + B(C+D)$

This three-level implementation requires 4 AND gates, 4 OR gates and 3 inverter gates for variables *B, D and the sum term (C+D).* (Better)

Design Procedure (4)

★ BCD-to-Excess-3 Converter

A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x



$w = A + B(C+D)$

$x = B'(C+D) + B(C+D)'$

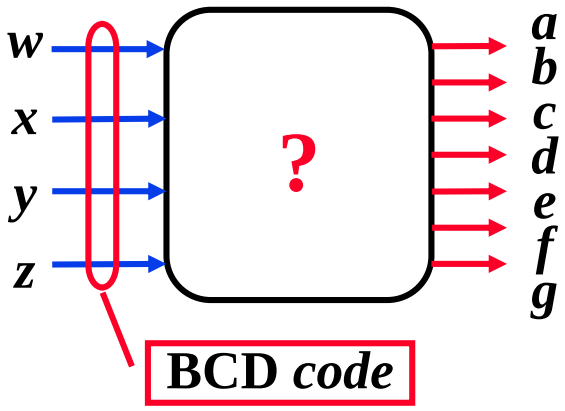
$y = (C+D)' + CD$

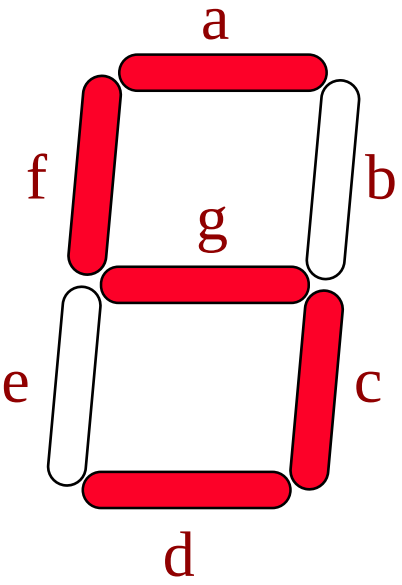
$z = D'$

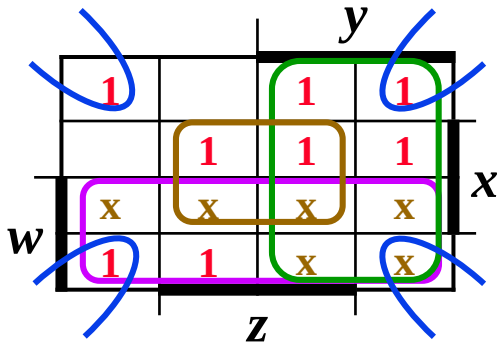
Design Procedure (5)

★ Seven-Segment Decoder

w x y z	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 0	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
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x







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

$b = \ldots$

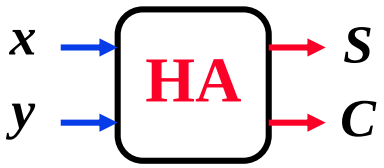
$c = \ldots$

$d = \ldots$

Binary Adder

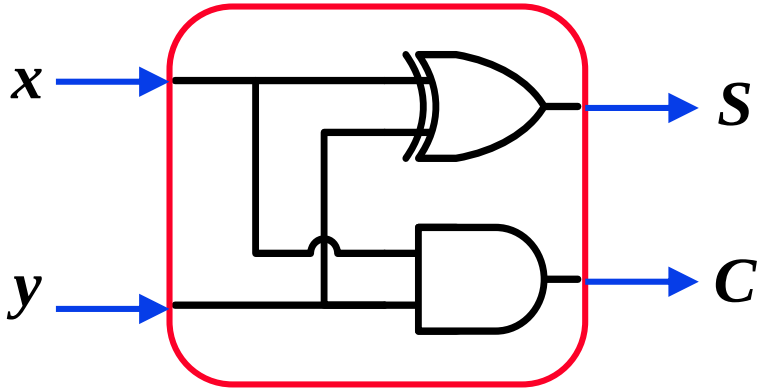
★ Half Adder

- Adds 1-bit plus 1-bit
- Produces Sum and Carry



$$\begin{array}{rr} & x \\ + & y \\ \hline C & S \end{array}$$

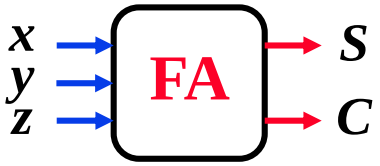
x y	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0



Binary Adder (2)

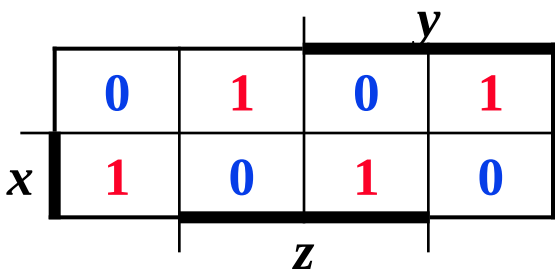
★ Full Adder

- Adds 1-bit plus 1-bit plus 1-bit
- Produces Sum and Carry

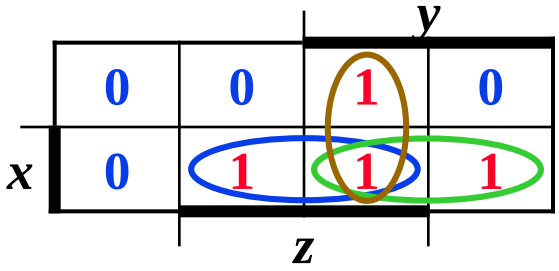


x y z	C S
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

$$\begin{array}{rr} & x \\ + & y \\ + & z \\ \hline C & S \end{array}$$



$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$



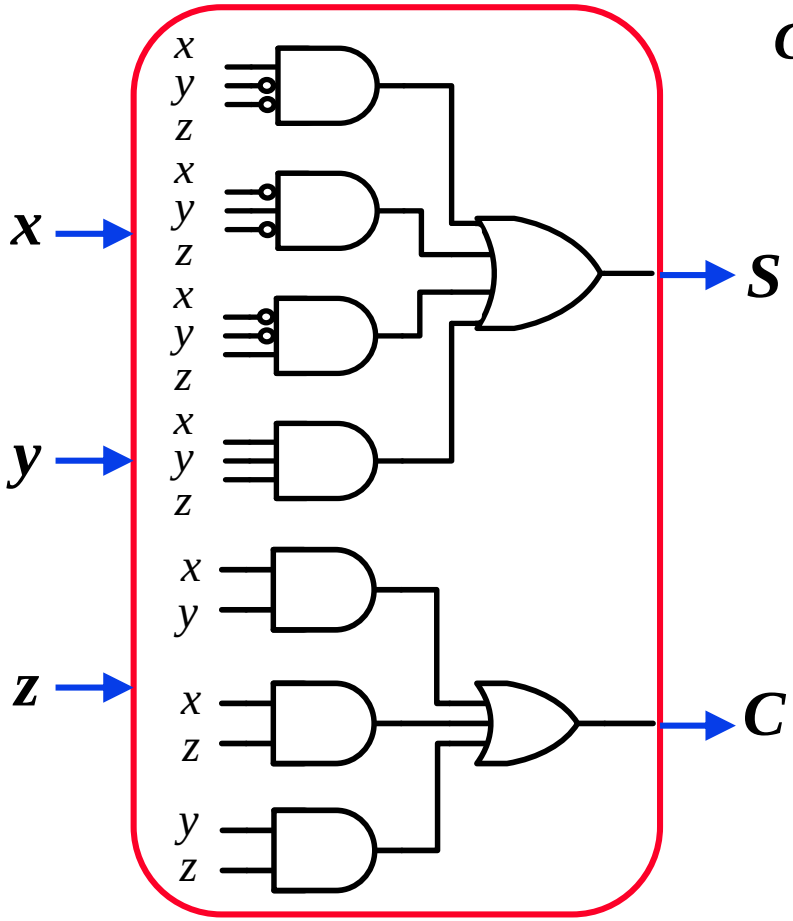
$$C = xy + xz + yz$$

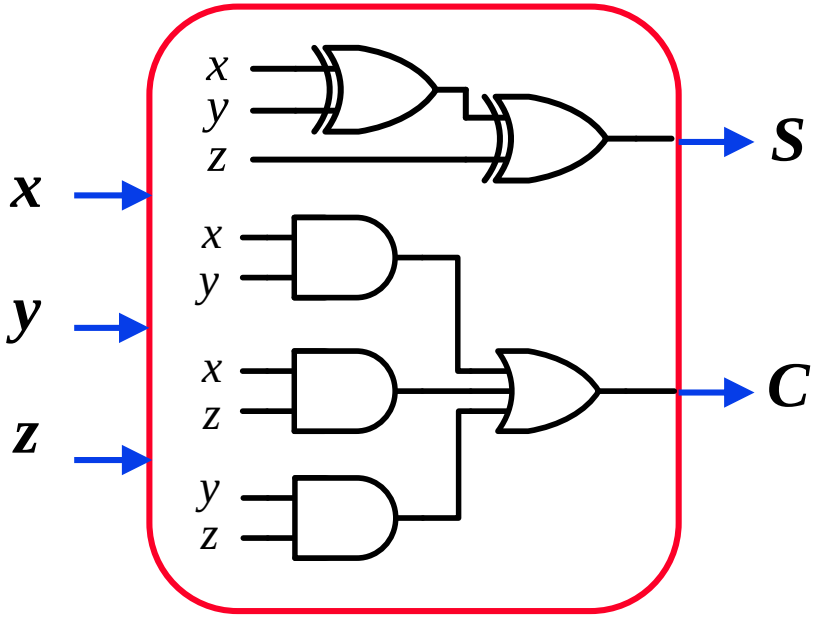
Binary Adder (3)

★ Full Adder

$S = xy'z'+x'yz'+x'y'z+xyz = x \oplus y \oplus z$

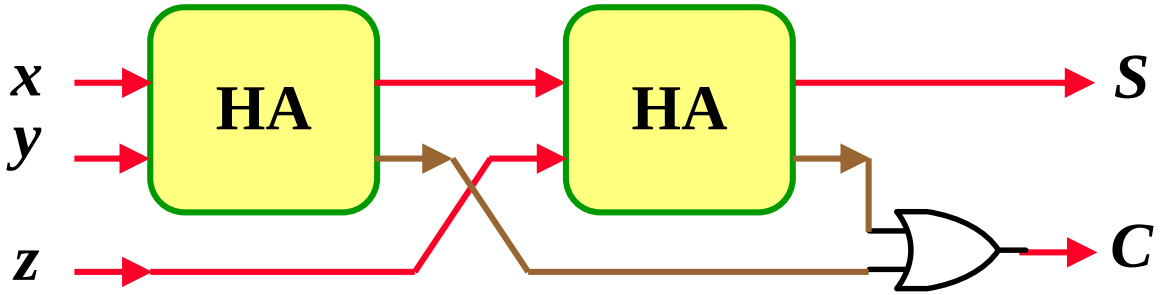
$C = xy + xz + yz$

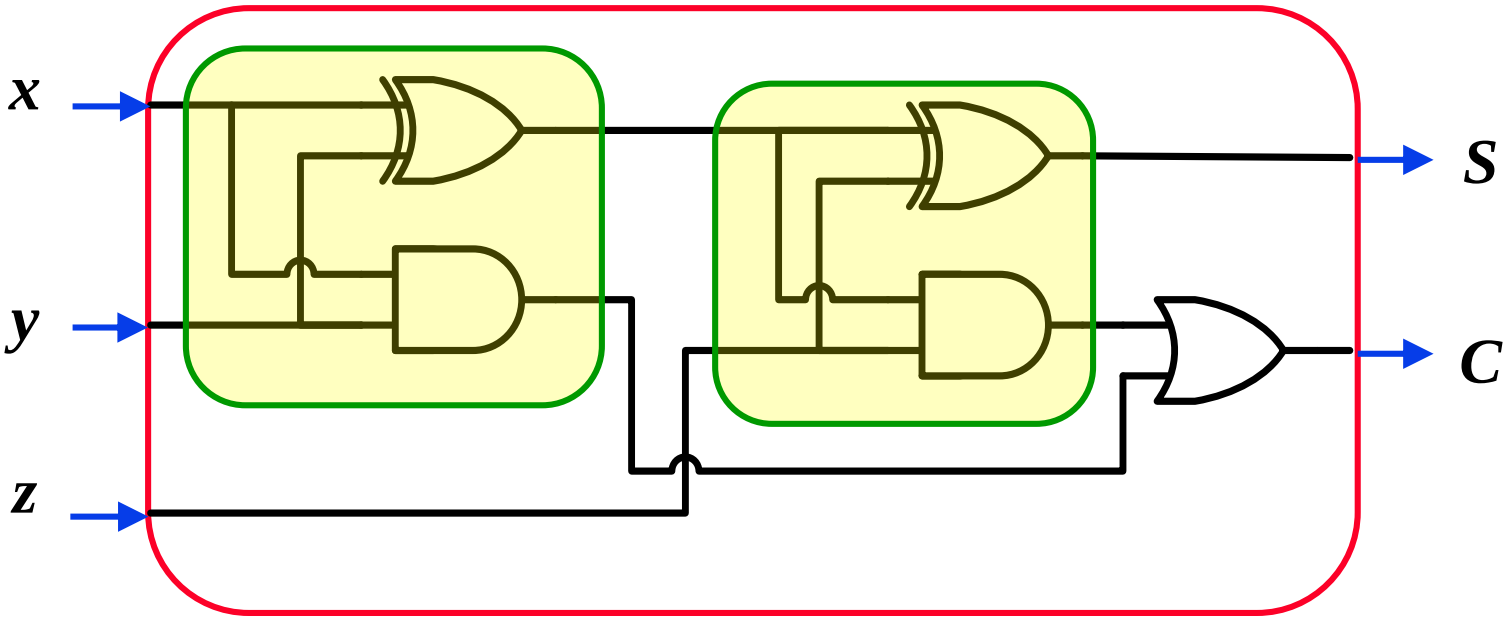




Binary Adder (4)

★ Full Adder





Binary Adder (5)

★ Full Adder

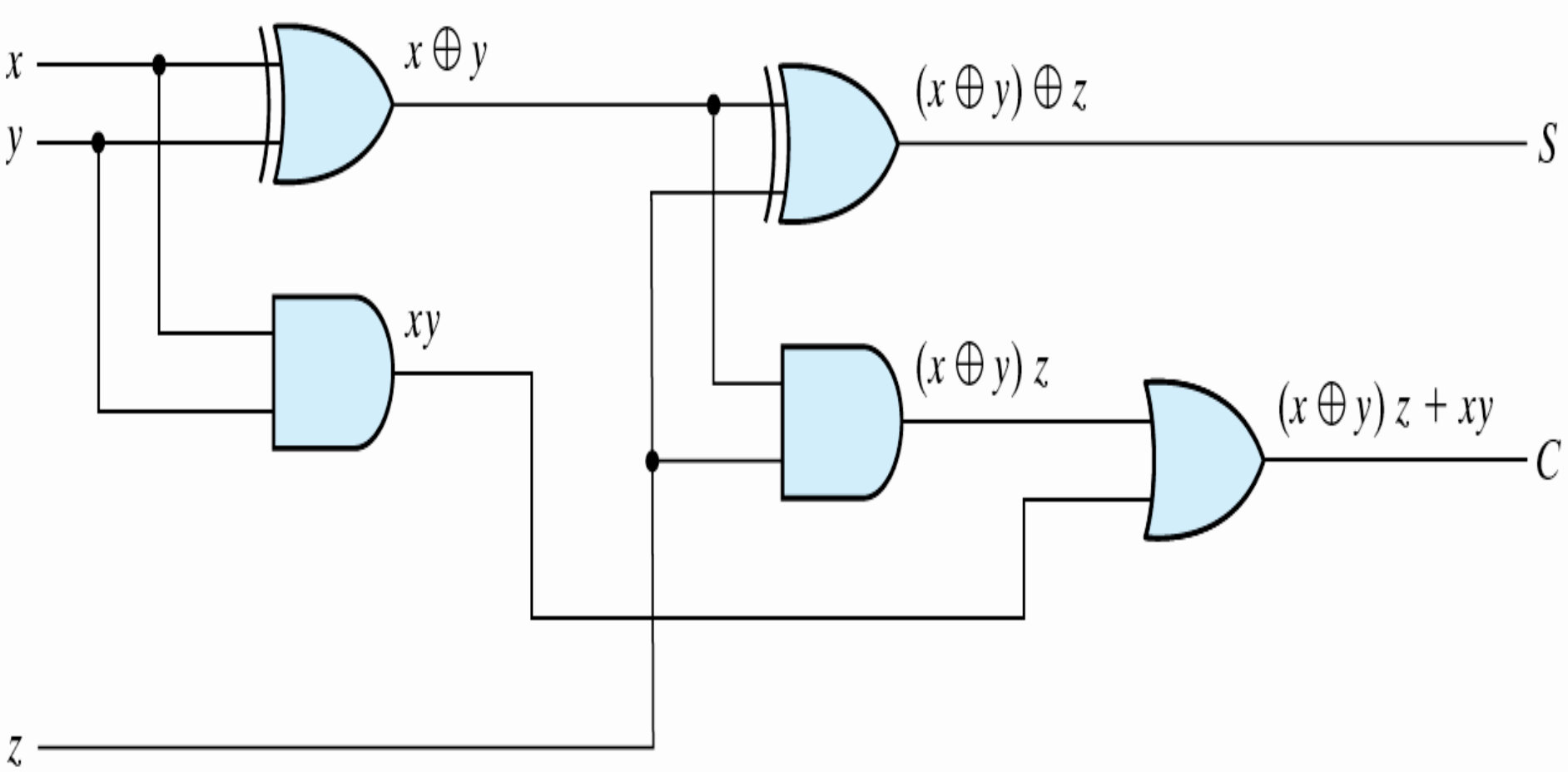
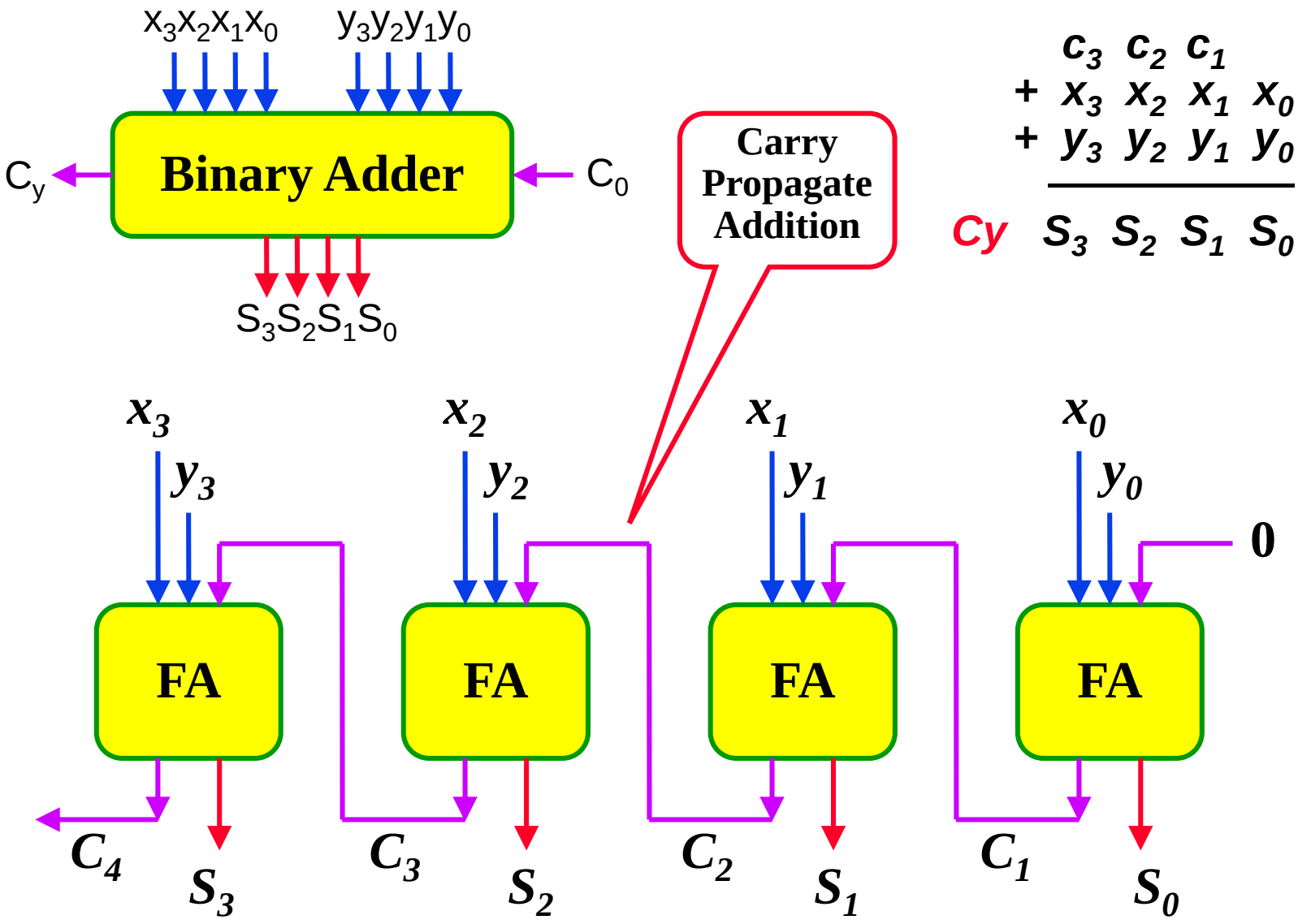


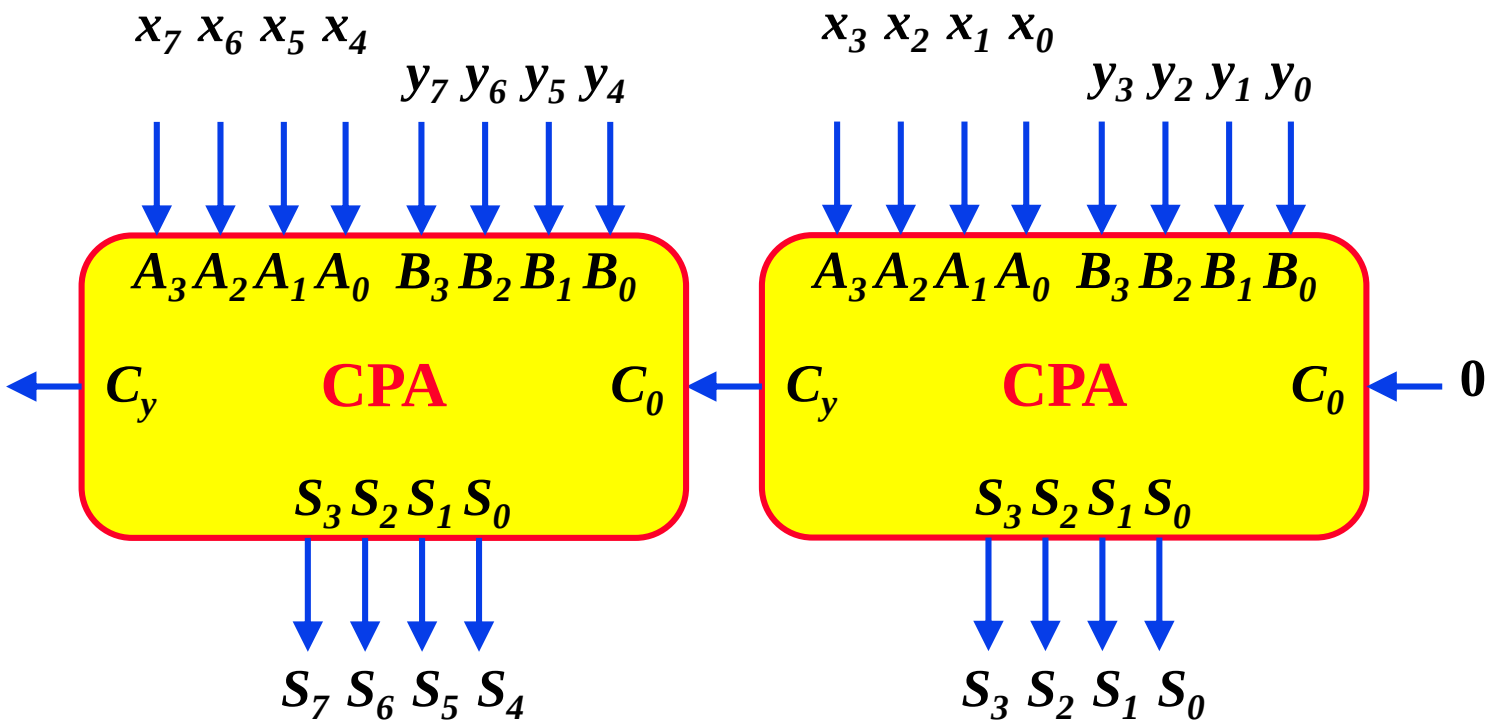
Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary Adder (6)



Binary Adder (7)

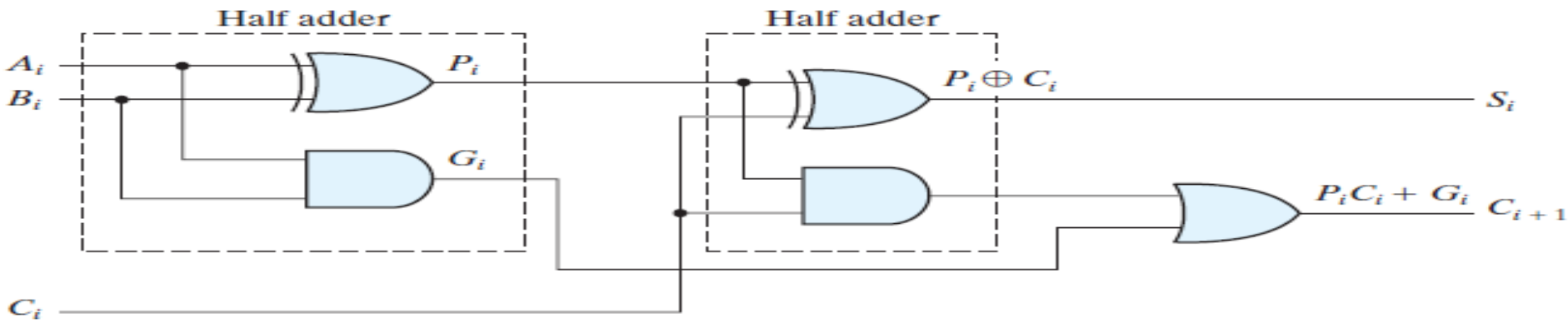
★ Carry Propagate Adder



Carry Propagation

- ★ This is also called **Ripple Carry Adder (RCA)**, because of the construction with full adders are connected in cascade.
- ★ This causes a unstable factor on carry bit, and produces a longest propagation delay.
- ★ The signal from C_i to the output carry C_{i+1}, propagates through an AND and OR gates, so, for an n-bit RCA, there are 2n gate levels for the carry to propagate from input to output.
- ★ Because the propagation delay will affect the output signals on different time, so the signals are given enough time to get the precise and stable outputs.
- ★ The most widely used technique employs the principle of **carry look-ahead** to improve the speed of the algorithm.

Carry Look-Ahead Generator



$P_i = A_i \oplus B_i$ steady state value

$G_i = A_i B_i$ steady state value

G_i : Carry Generate because it produces a carry regardless of the input carry C_i

P_i : Carry Propagate because it determines whether a carry into a stage *i* will propagate into a stage *i+1*

Output sum and carry

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

C_0 = *input carry*

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

★ C_3 does not have to wait for C_2 and C_1 to propagate.

Carry Look-Ahead Generator (2)

★ C_3 is propagated at the same time as C_2 and C_1.

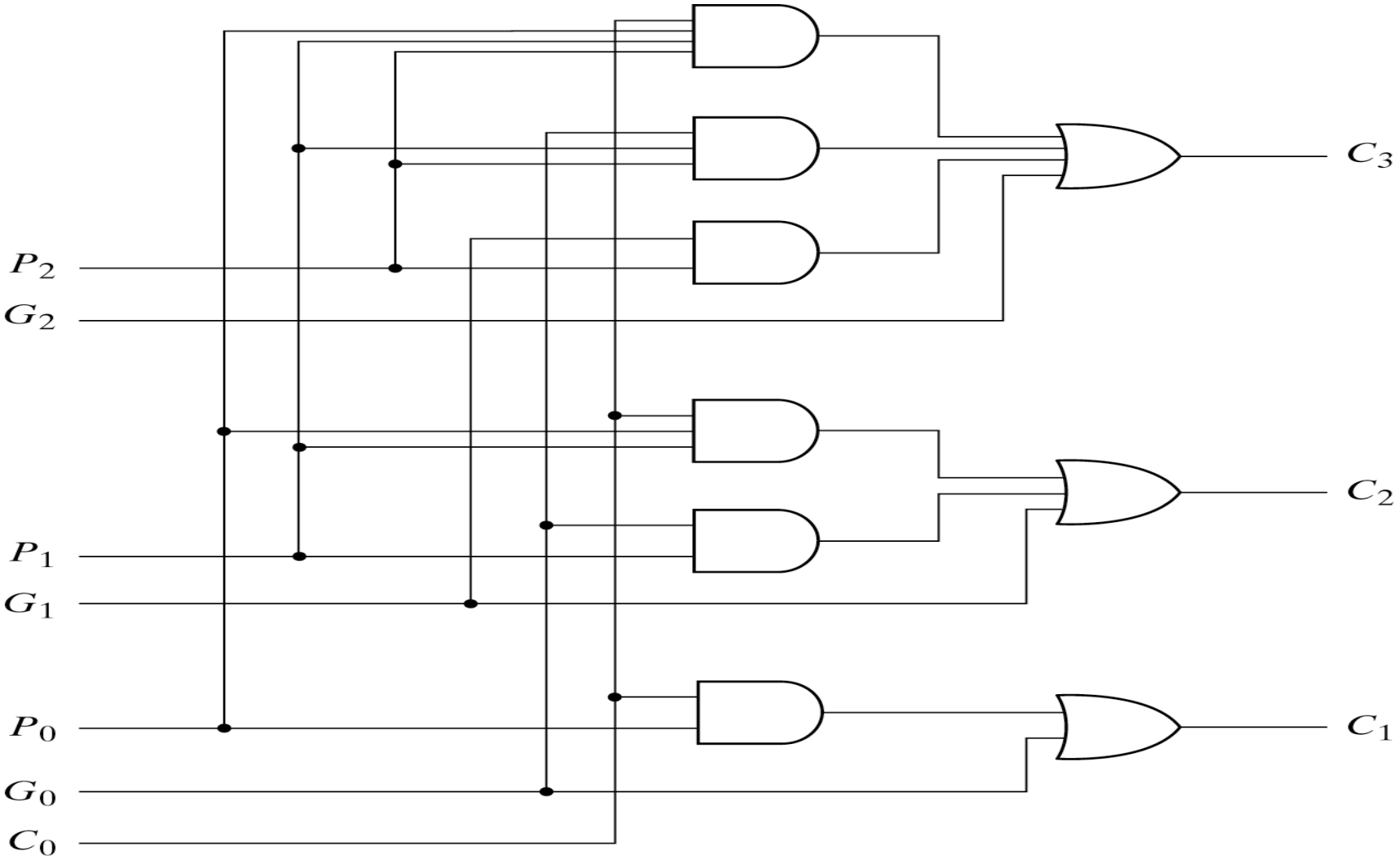


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4-bit Carry Look-Ahead Adder (CLAA)

★ **Delay time of n-bit CLAA = XOR + (AND + OR) + XOR**

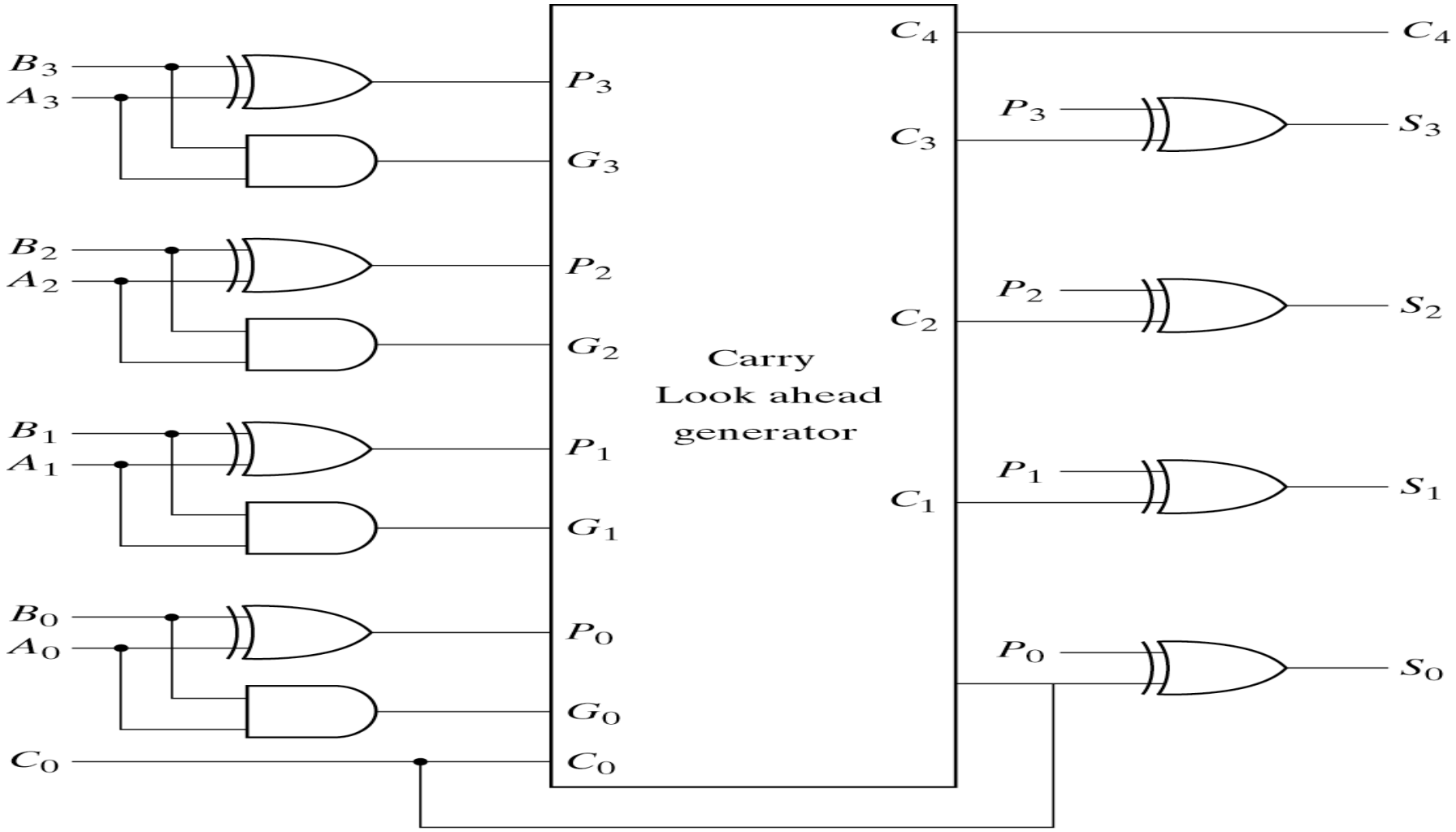


Fig. 4-12 4-Bit Adder with Carry Lookahead

The End

Questions?