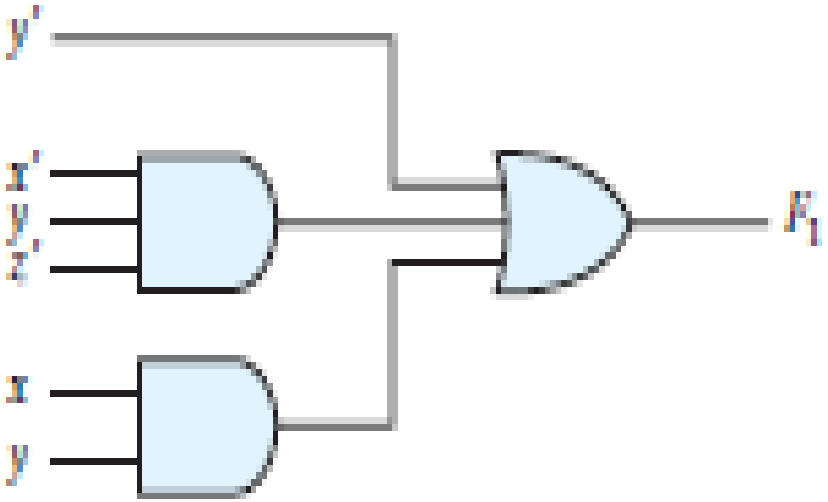
OTHER TWO-LEVEL IMPLEMENTATIONS

Two-level Logic

- Two level logic means that the logic design uses maximum two logic gates between input and output.
- This does not mean that the whole design will contain only two logic gates but the single path from input to output may contain no more than two logic gates.

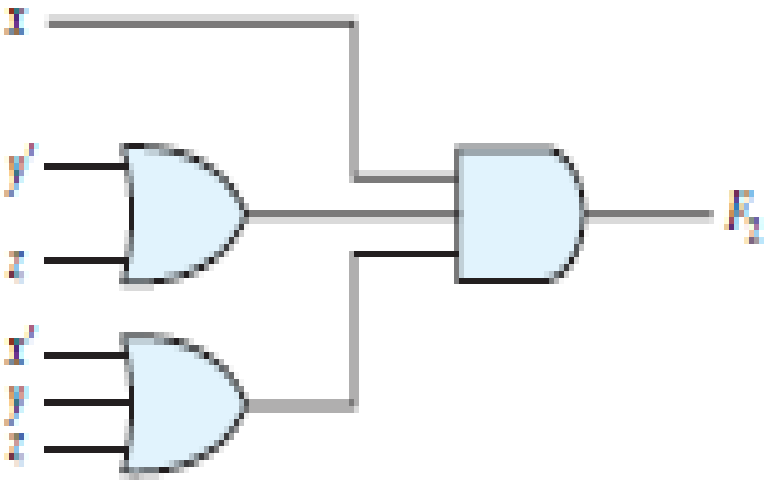
Two-level Implementation

$F_1 = y' + xy + x'yz'$



Sum of Products

$F_2 = x(y' + z)(x' + y + z)$

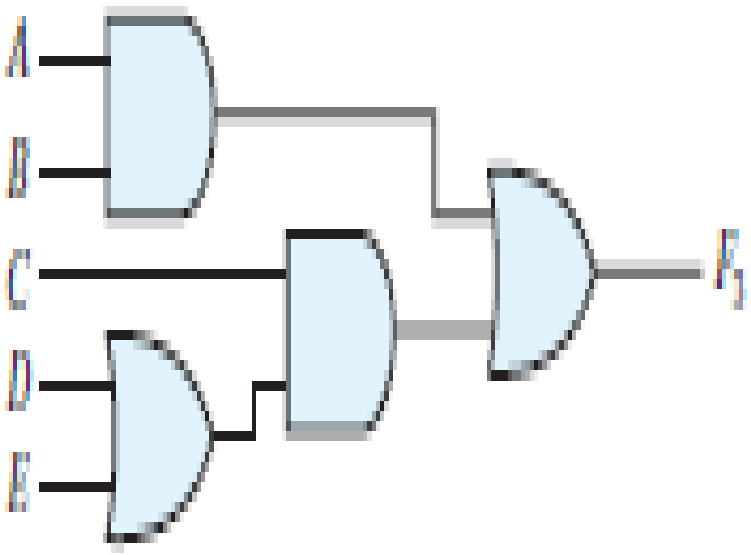


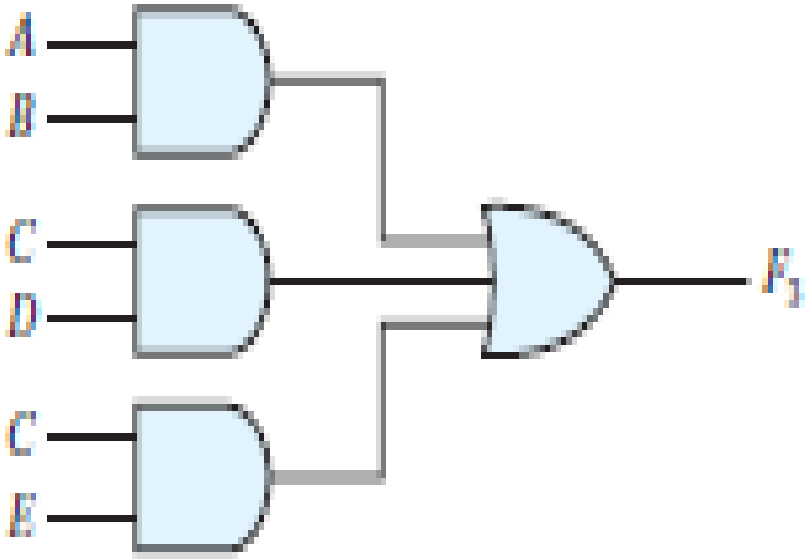
Product of Sums

Multi-level Implementation

$F_3 = AB + C(D + E)$

$F_3 = AB + C(D + E)$
$= AB + CD + CE$





- The standard type (SOP / POS) circuit configuration is referred to as a *two-level implementation*.
- In general, a two-level implementation is preferred because it produces the least amount of delay through the gates when the signal propagates from the inputs to the output.
- However, the number of inputs to a given gate might not be practical.

Other Two-Level Implementations

- For two-level logic implementation, we consider four types of gates: AND, OR, NAND, and NOR.
- If we assign one type of gate for the first level and one type for the second level, we find that there are 16 possible combinations of two-level forms.

- Each two-level combination implements different logic functions. There are two main types in these 16 combinations.
  - Degenerate Form
  - Non-Degenerate Form

Degenerate Form

- The two-level combination that degenerates into a single logic function as known as degenerate form.
- There are 8 degenerate forms in those 16 combinations. Each of these degenerate forms is given below.

AND-AND	NAND-NOR
OR-OR	NOR-NAND
AND-NAND	NAND-OR
OR-NOR	NOR-AND

Non-Degenerate Form

- Those combinations of Two-level logic, which implements Sum of Product form or Product of sum form are Non-degenerate forms.
- The remaining 8 of the total 16 are all non-degenerate forms which are given below.

AND-OR	OR-AND
NAND-NAND	NOR-NOR
AND-NOR	OR-NAND
NAND-AND	NOR-OR

AOI Logic and OAI Logic

- The logic function

  $F = (AB + CD)' = (A' + B')(C' + D')$

  is called an AND–OR–INVERT function.

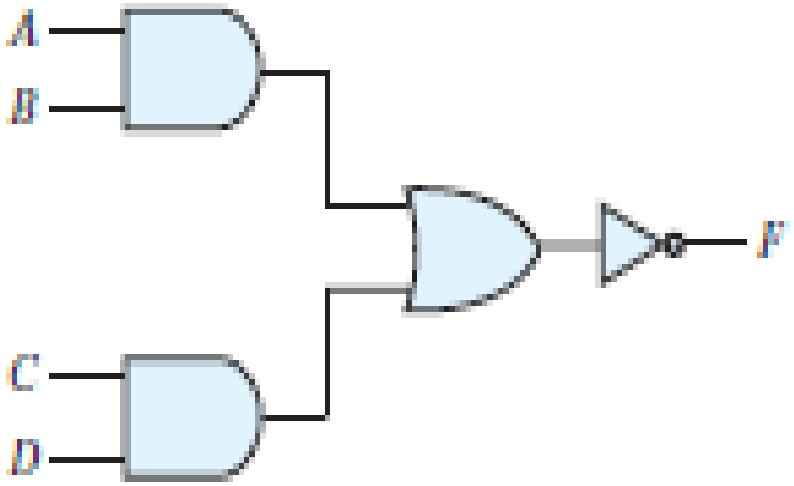
- The logic function

  $F = [(A + B)(C + D)]' = A'B' + C'D'$

  is called an OR–AND-INVERT function.

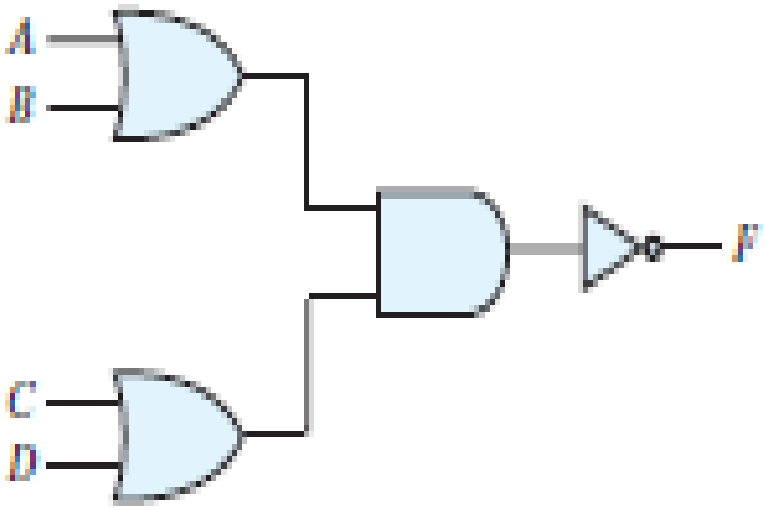
AND–OR–INVERT function

$F = (AB + CD)'$



OR–AND-INVERT function

$F = [(A + B)(C + D)]'$

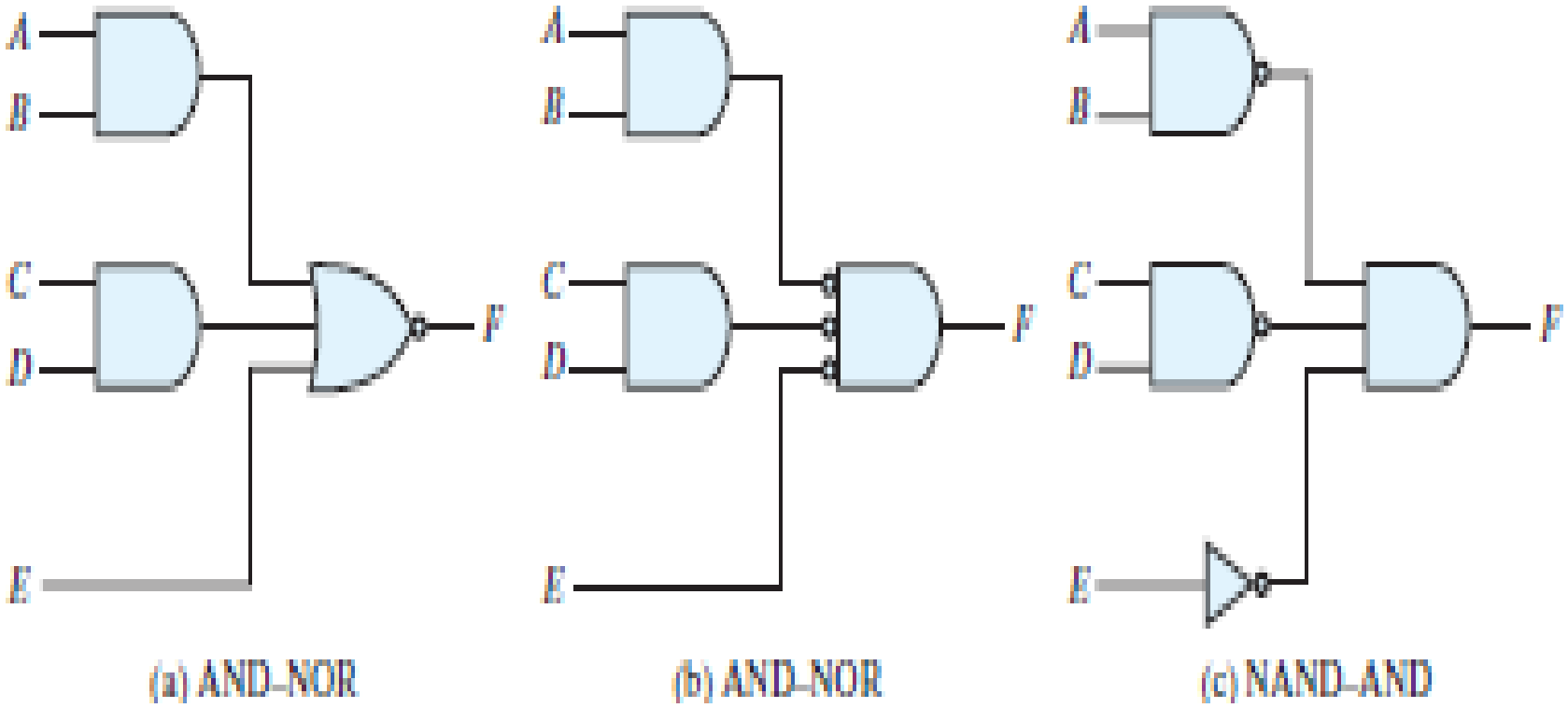


AND–OR–INVERT Implementation

- The two forms, NAND–AND and AND–NOR, are equivalent and can be treated together. Both perform the AND–OR–INVERT function.
- The AND–NOR form resembles the AND–OR form, but with an inversion done by the bubble in the output of the NOR gate.
- It implements the function $F = (AB + CD + E)'$
- An AND–OR implementation requires an expression in sum-of-products form. The AND–OR–INVERT implementation is similar, except for the inversion.

AND–OR–INVERT circuits,

$F = (AB + CD + E)'$

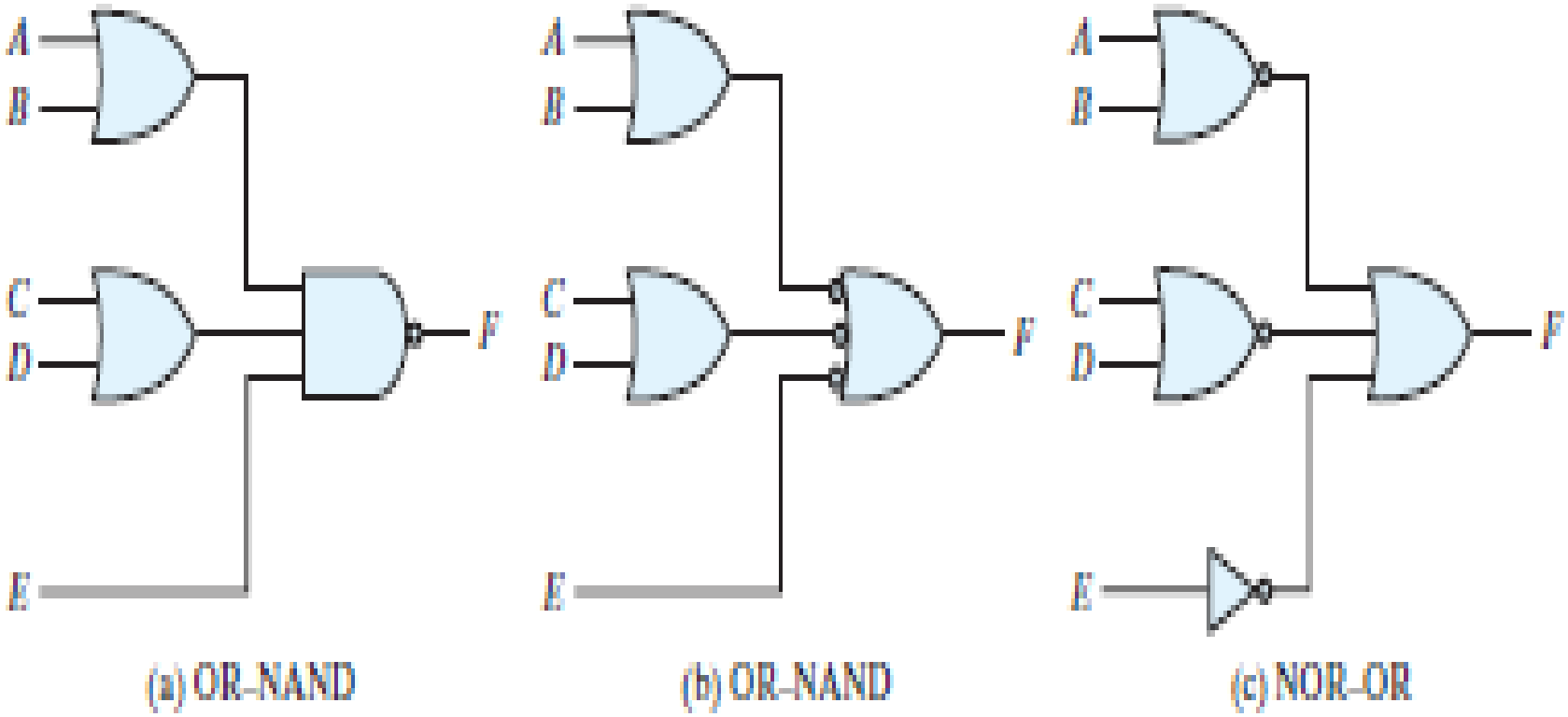


OR–AND–INVERT Implementation

- The OR–NAND and NOR–OR forms perform the OR–AND–INVERT function.
- The OR–NAND form resembles the OR–AND form, except for the inversion done by the bubble in the NAND gate.
- It implements the function $F = [(A + B)(C + D)E]'$
- The OR–AND–INVERT implementation requires an expression in product-of-sums form.

OR–AND–INVERT circuits,

$F = [(A + B)(C + D)E]'$



(a) OR–NAND (b) OR–NAND (c) NOR–OR

Implementation with Other Two-Level Forms

- Table summarizes the procedures for implementing a Boolean function in any one of the four 2-level forms.

Equivalent Nondegenerate Form		Implements the Function	Simplify F' into	To Get an Output of
(a)	(b)*			
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0's in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1's in the map and then complementing.	F

*Form (b) requires an inverter for a single literal term.

Example: Implement the function $F = x'y'z' + xyz'$ with the four 2-level forms AND–NOR, NAND–AND, OR–NAND and NOR–OR.

Solution:

- AND–OR–INVERT form

$F = x'y'z' + xyz'$

$F' = (x'y'z' + xyz')' = (x'y'z')'\,(xyz')'$

$= (x + y + z)\,(x' + y' + z)$

$= xx' + xy' + xz + x'y + yy' + yz + x'z + y'z + zz$

$= x'y + xy' + xz + x'z + yz + y'z + z$

$= x'y + xy' + (x + x')z + (y + y')z + z$

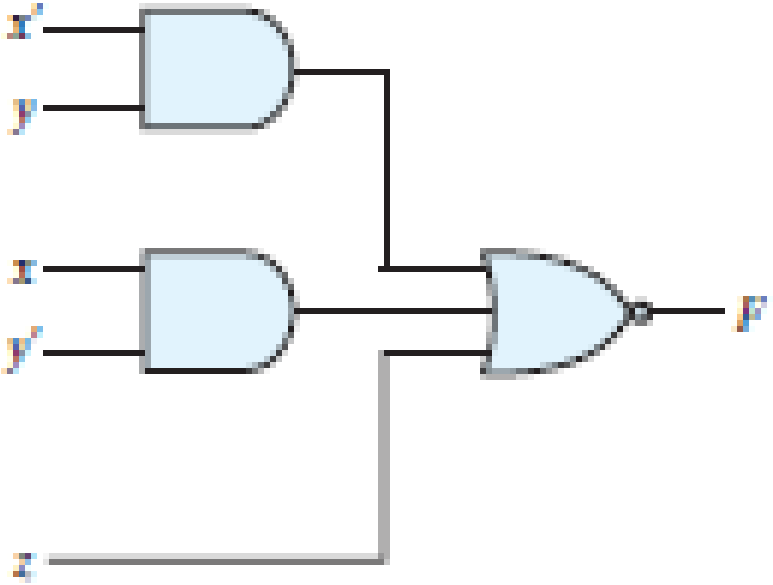
$= x'y + xy' + z + z + z = x'y + xy' + z$

$F' = x'y + xy' + z$

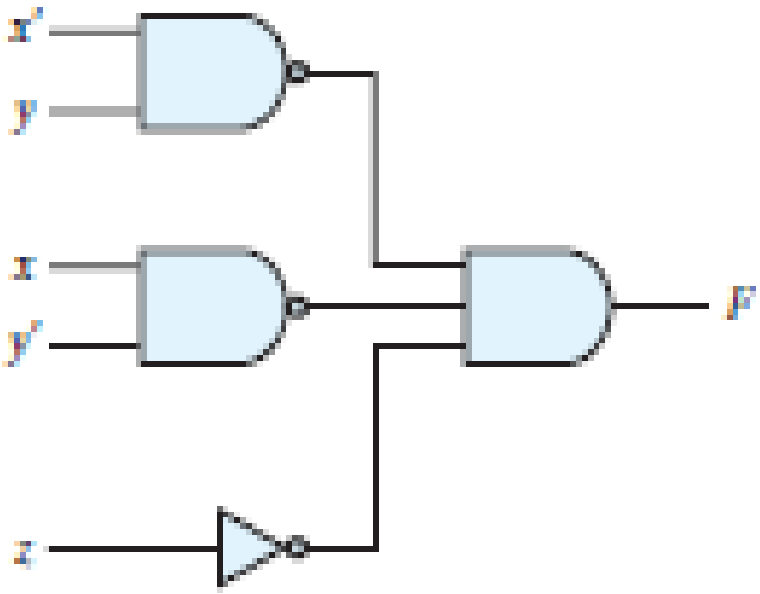
$F = (x'y + xy' + z)'$

AND–NOR and NAND–AND Implementations

$F = x'y'z' + xyz' = (x'y + xy' + z)'$



AND–NOR



NAND–AND

- OR–AND–INVERT form

$F = x'y'z' + xyz'$

$F' = (x'y'z' + xyz')'$

$= (x'y'z')'\,(xyz')'$

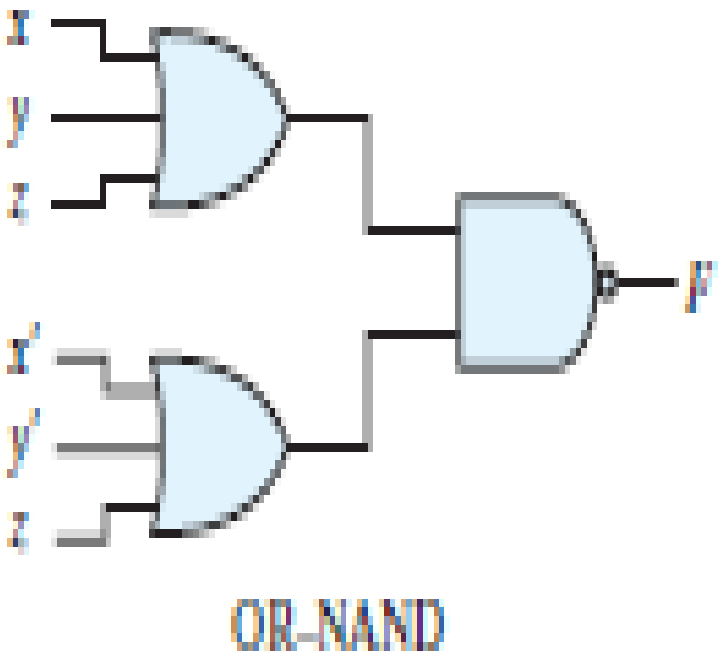
$= (x + y + z)\,(x' + y' + z)$

$F' = (x + y + z)\,(x' + y' + z)$

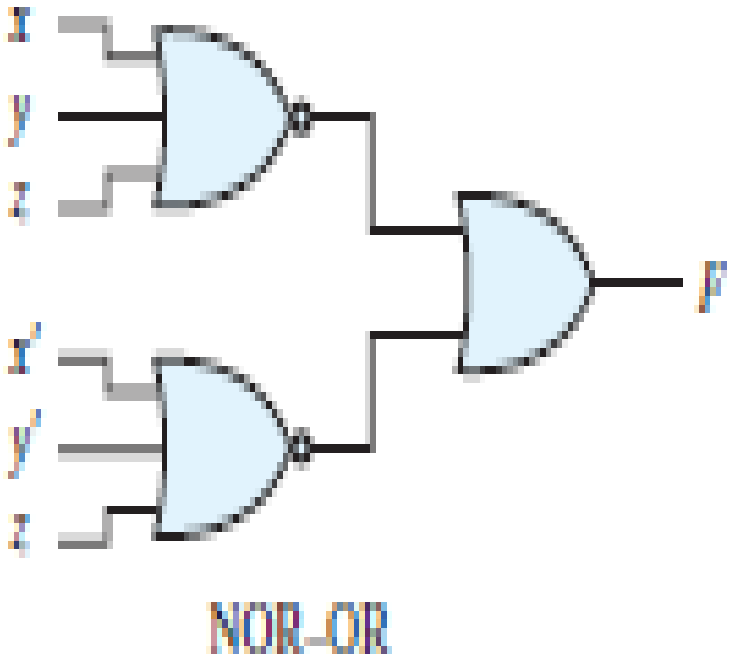
$F = [(x + y + z)\,(x' + y' + z)]'$

OR–NAND and NOR–OR Implementations

$F = x'y'z' + xyz' = [(x + y + z)(x' + y' + z)]'$



OR–NAND



NOR–OR