Gate-Level Minimization

Dr. Bassem A. Abdullah

Computer and Systems Department

Lectures Prepared by Dr.Mona Safar, Edited and Lectured by Dr.Bassem A. Abdullah

Outline

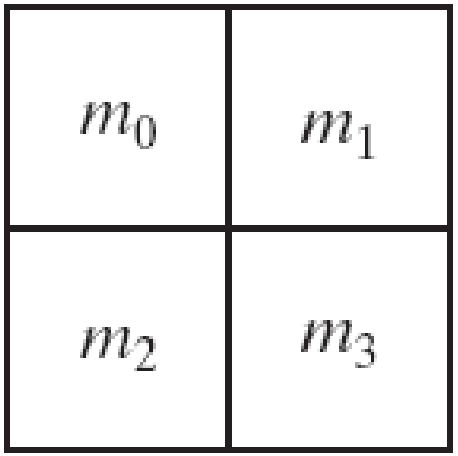
1. The Map Method
2. Four-variable Map
3. Five-variable Map
4. Product of Sums Simplification
5. Don’t-Care Conditions
6. NAND and NOR Implementation
7. Other Two-level Implementations
8. Exclusive-OR Function

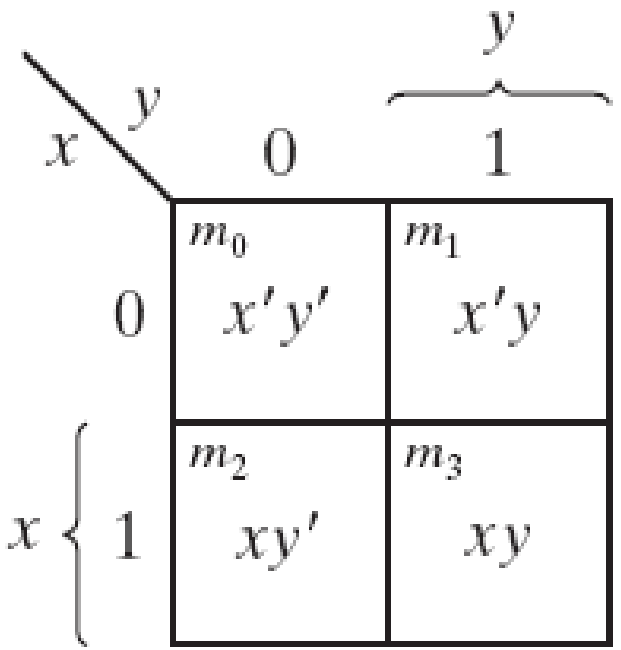
Karnaugh Map (K-Map) Simplification Method

1. A pictorial form of truth table
2. Express any Boolean function as sum of minterms (SoP)
3. Simplify a Boolean function in SoP or PoS standard forms

Two-variable K-map

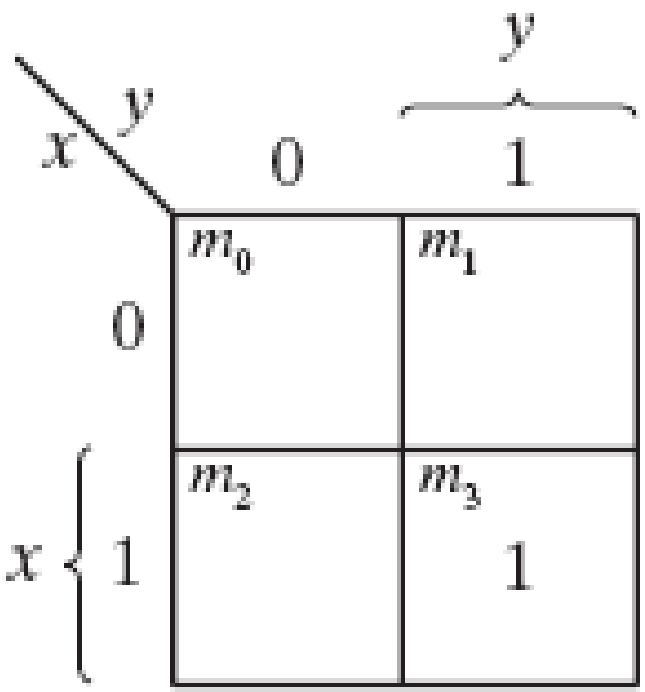
Presents the four minterms of a two-variable function as follows:





Representation of functions in a K-map

F = xy



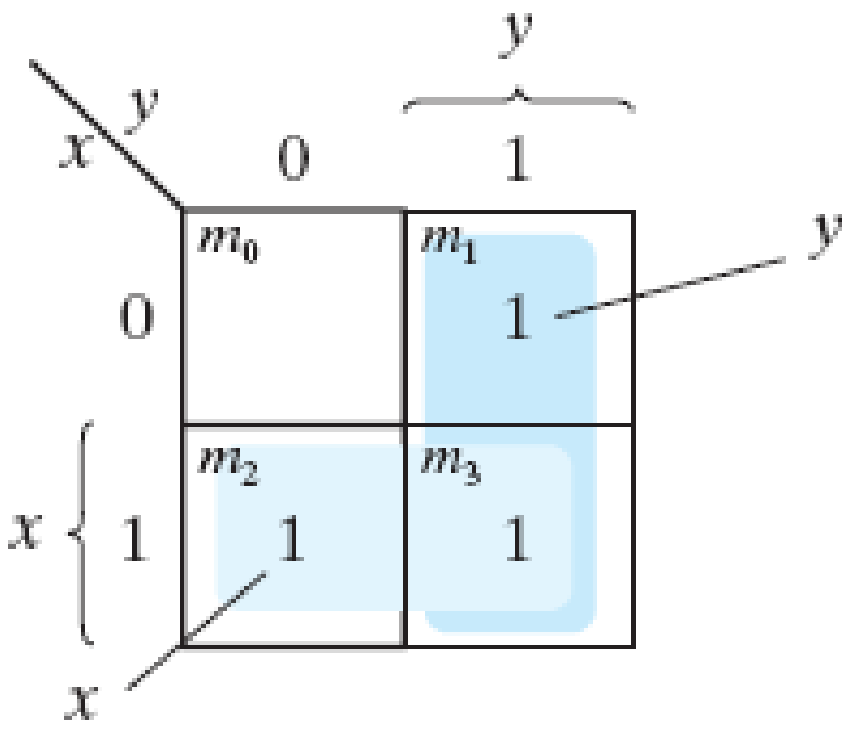
Representations of functions in a K-map

F = x + y

= xy' + xy + x'y

= $m_2 + m_3 + m_1$

= $\sum(1, 2, 3)$



Simplification of functions in a K-map

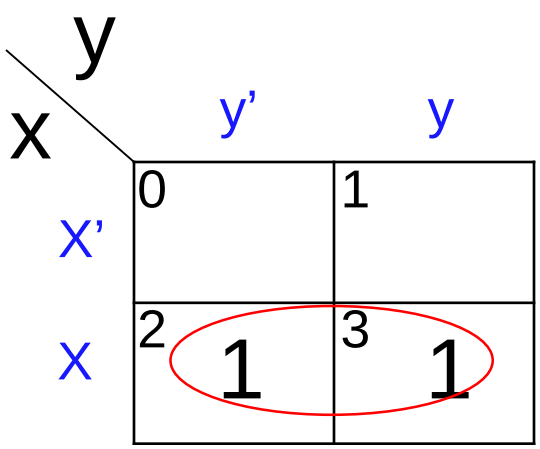
$F = \sum(2, 3)$

$= xy' + xy$

$= x(y+y')$

$= x.1$

$= x$



K-map Simplification of a Two-variable Function

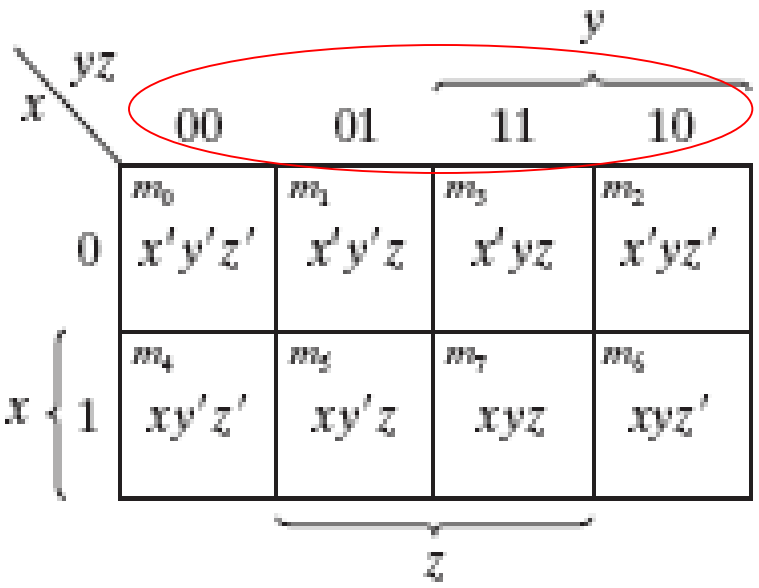
- **1** square (minterm) represents a term of **2** literals
- **2** adjacent squares represent a term of **1** literal
- **4** adjacent squares represent the function **1**

Three-variable K-map (F(x, y, z))

Presents the eight minterms of a three-variable function as follows:

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

Gray code: only one variable changes between 2 adjacent positions



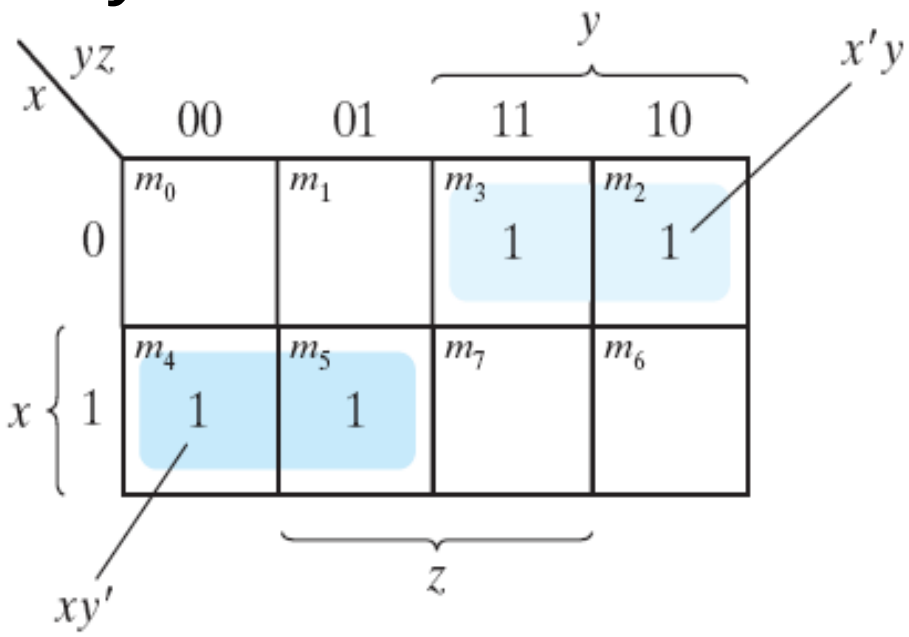
Representation of functions in a K-map

F = x'y + xy'

= x'yz' + x'yz + xy'z' + xy'z

= $m_2 + m_3 + m_4 + m_5$

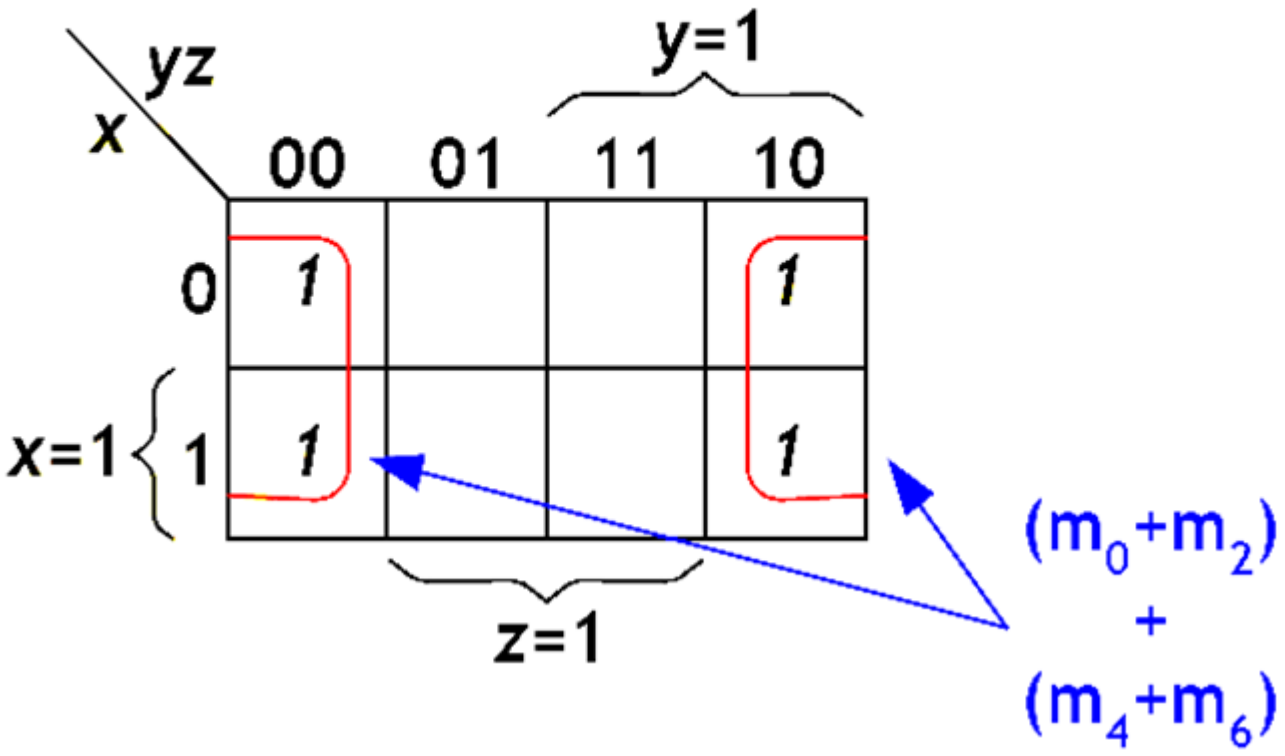
= $\sum(2, 3, 4, 5)$



Simplification of functions in a K-map

$F(x,y,z) = \Sigma(0,2,4,6)$

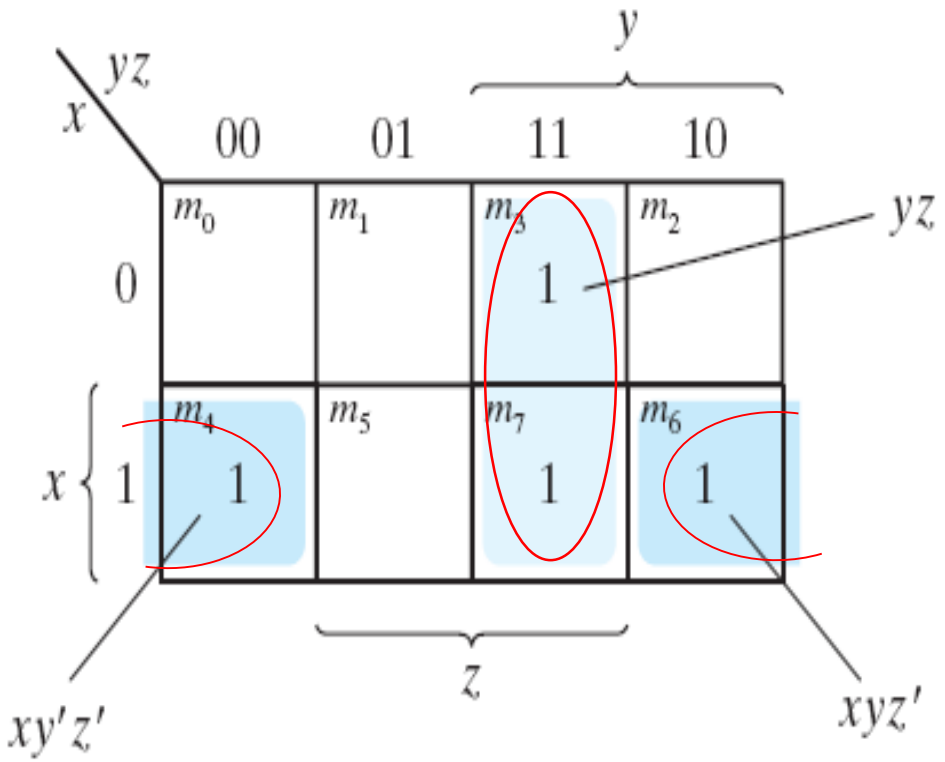
$= z'$



Simplification of functions in a K-map

F = ∑(3, 4, 6, 7)

= yz + xz’



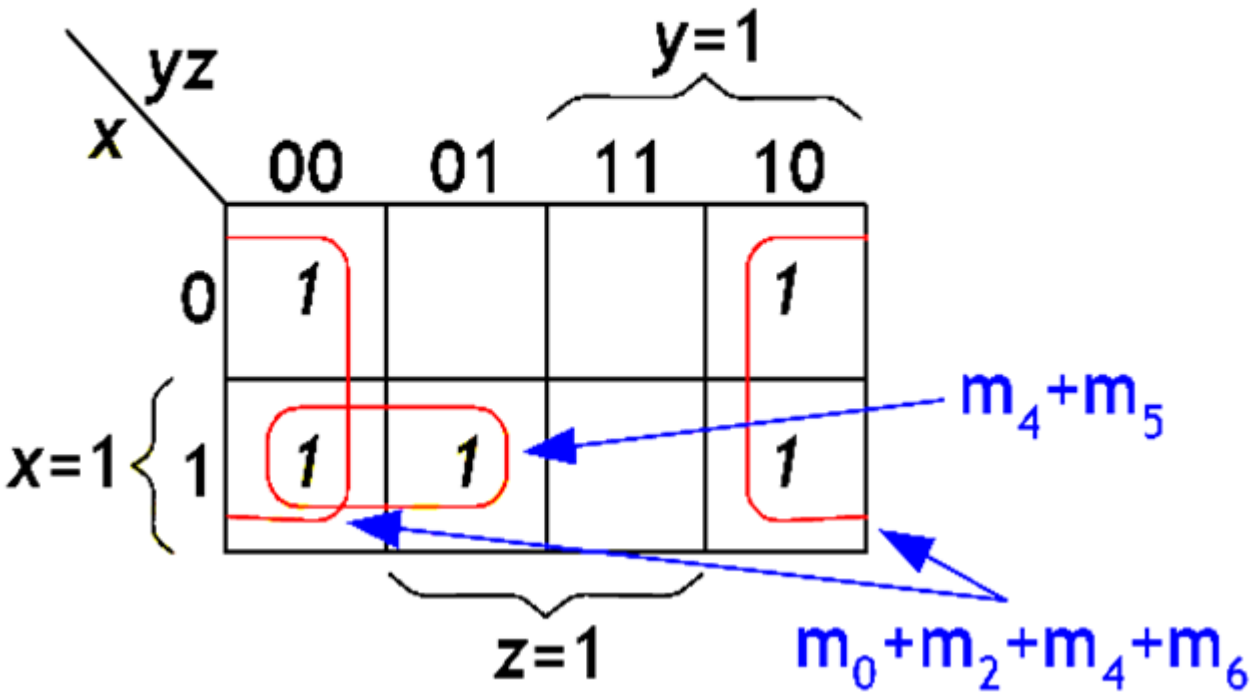
K-map Simplification of a Three-variable Function

- **1** square (minterm) represents a term of **3** literals
- **2** adjacent squares represent a term of **2** literal
- **4** adjacent squares represent a term of **1** literal
- **8** adjacent squares represent the function **1**

Example

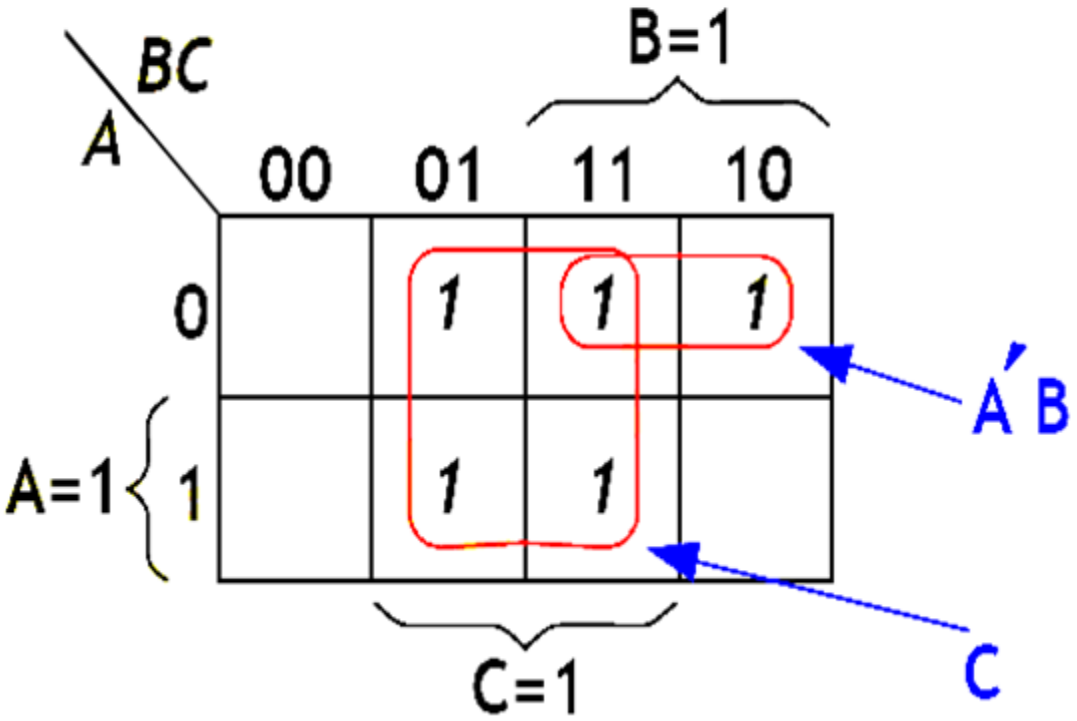
$F(x,y,z) = \Sigma(0,2,4,5,6)$

$= z' + xy'$



Example

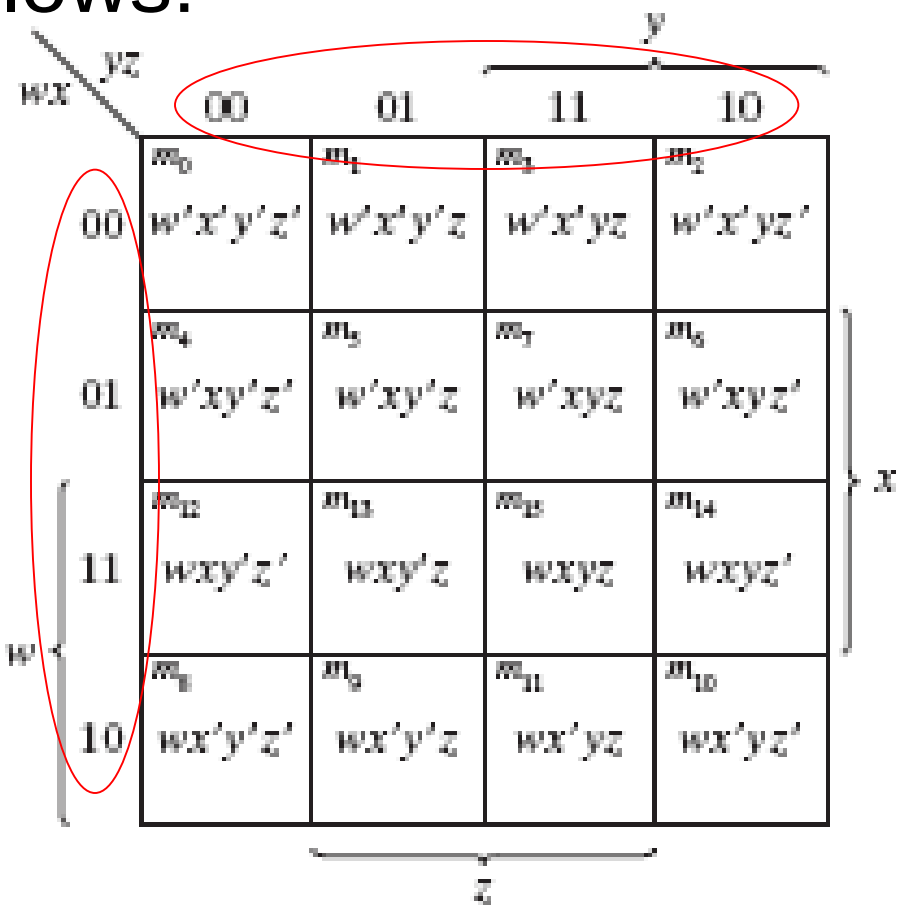
$$F(A,B,C) = A'C + A'B + AB'C + BC$$
$$= \Sigma\,(1,2,3,5,7)$$
$$= C + A'B$$



Four-variable K-map (F(w, x, y, z))

Presents the sixteen minterms of a four-variable function as follows:

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



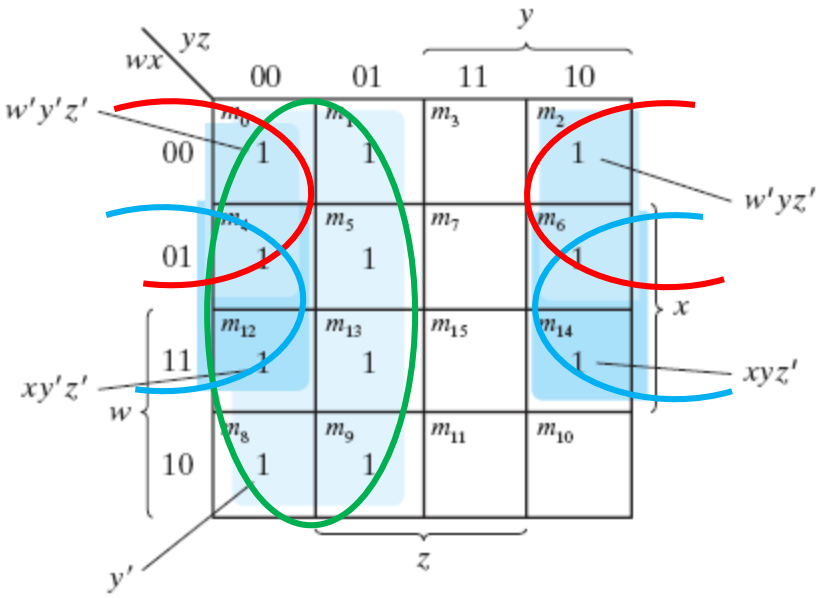
K-map Simplification of a Four-variable Function

- **1** square (minterm) represents a term of **4** literals
- **2** adjacent squares represent a term of **3** literal
- **4** adjacent squares represent a term of **2** literal
- **8** adjacent squares represent a term of **1** literal
- **16** adjacent squares represent the function **1**

Example

F(w, x, y, z) = ∑(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)

= y’ + w’z’ + xz’

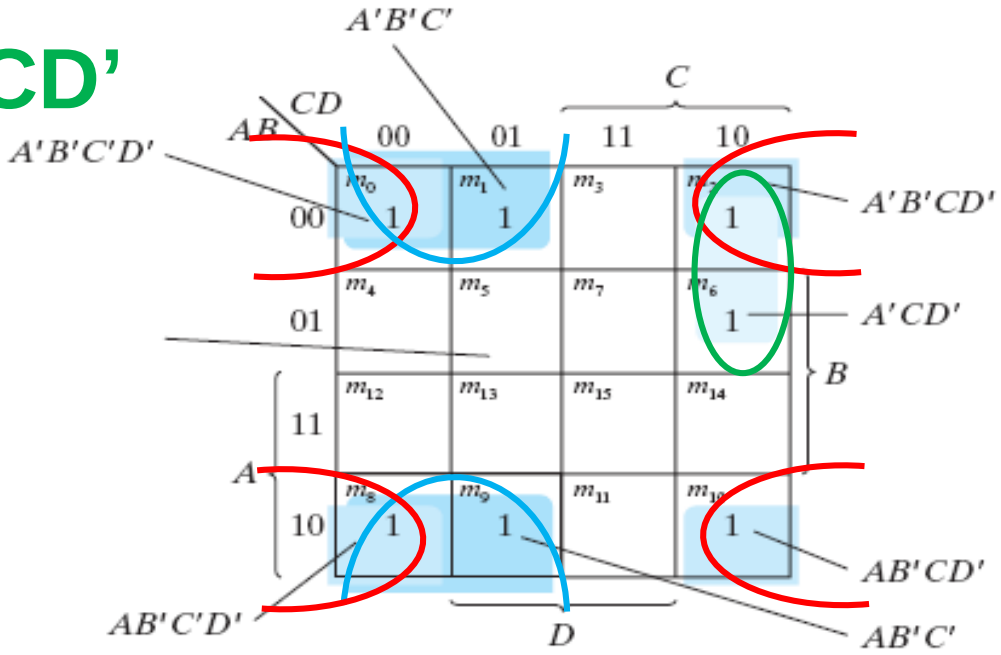


Example

F(A, B, C, D)

= A’B’C’ + B’C D’ + A’BCD’ + AB’C’

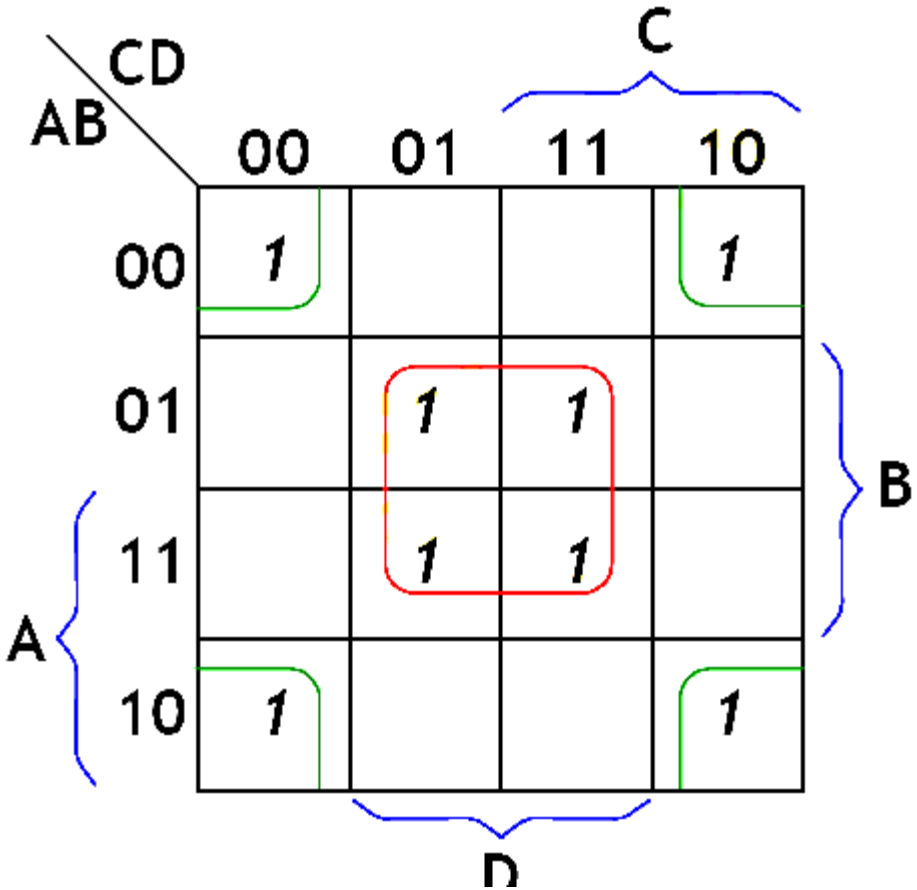
= **B’D’** + **B’C’** + **A’CD’**



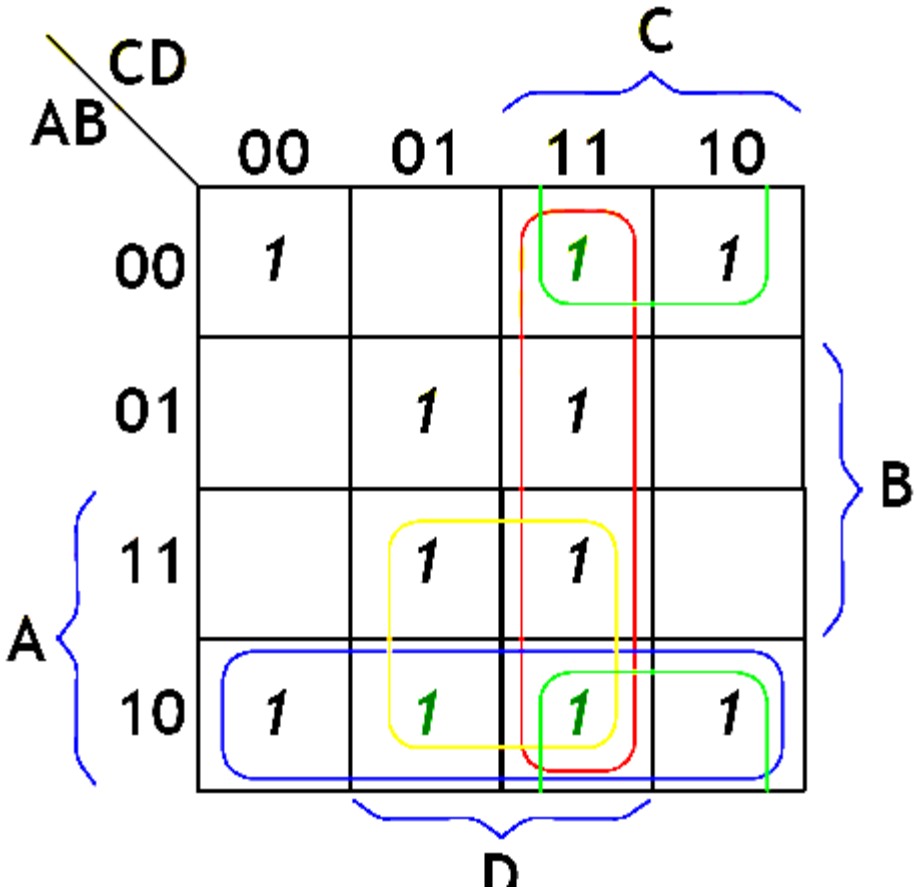
Prime Implicants and Essential Prime Implicants

- **Prime Implicant:** A product term containing the maximum possible number of adjacent squares in the k-map.
- **Essential Prime Implicant:** A prime implicant that contains a minterm that is covered by only one prime implicant.

Essential Prime Implicants:
BD and B'D'



Prime Implicants:
CD, B'C, AD, and AB'



Example

$F = \sum(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

$= BD + B'D' + CD + AD$

$= BD + B'D' + CD + AB'$

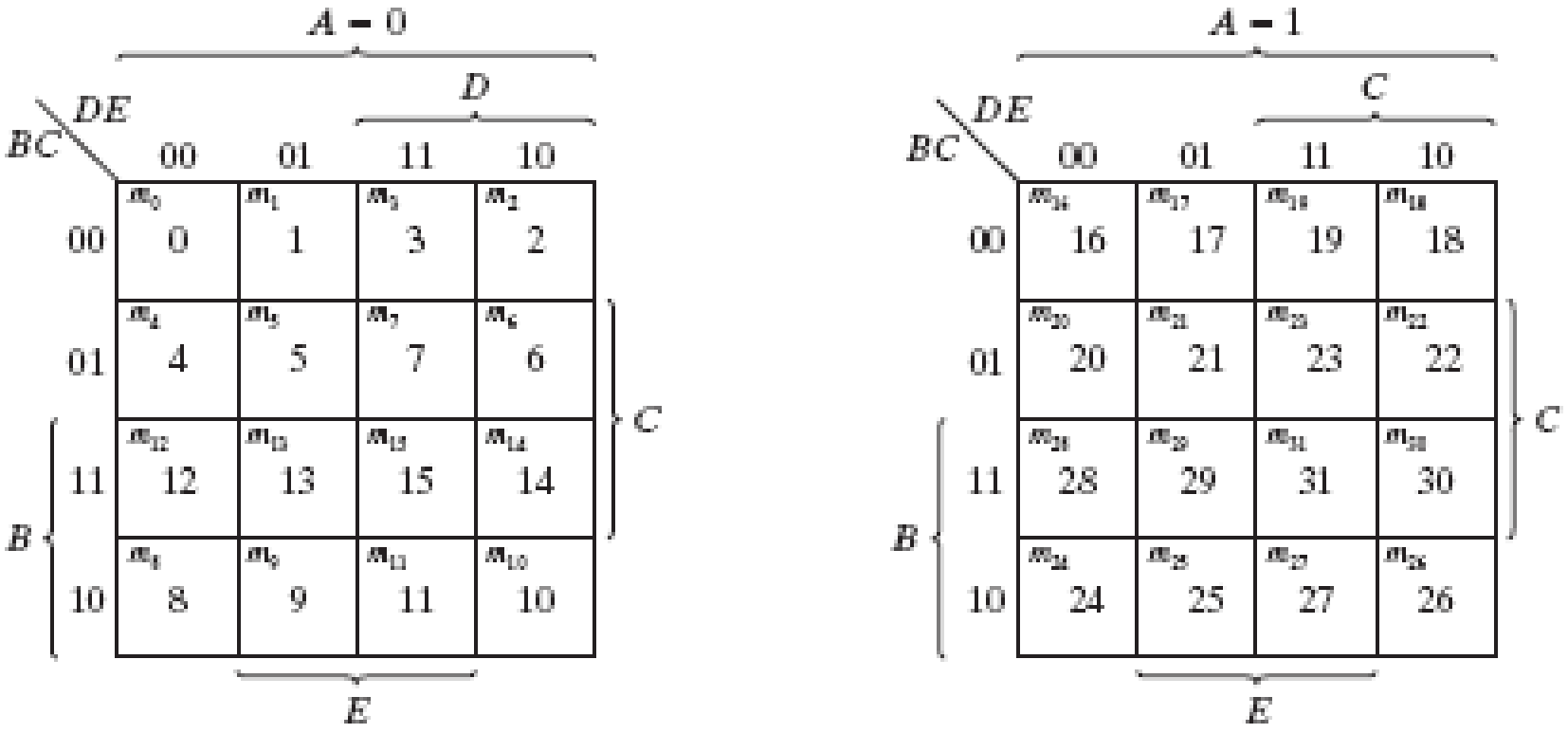
$= BD + B'D' + B'C + AD$

$= BD + B'D' + B'C + AB'$

Essential Prime Implicants

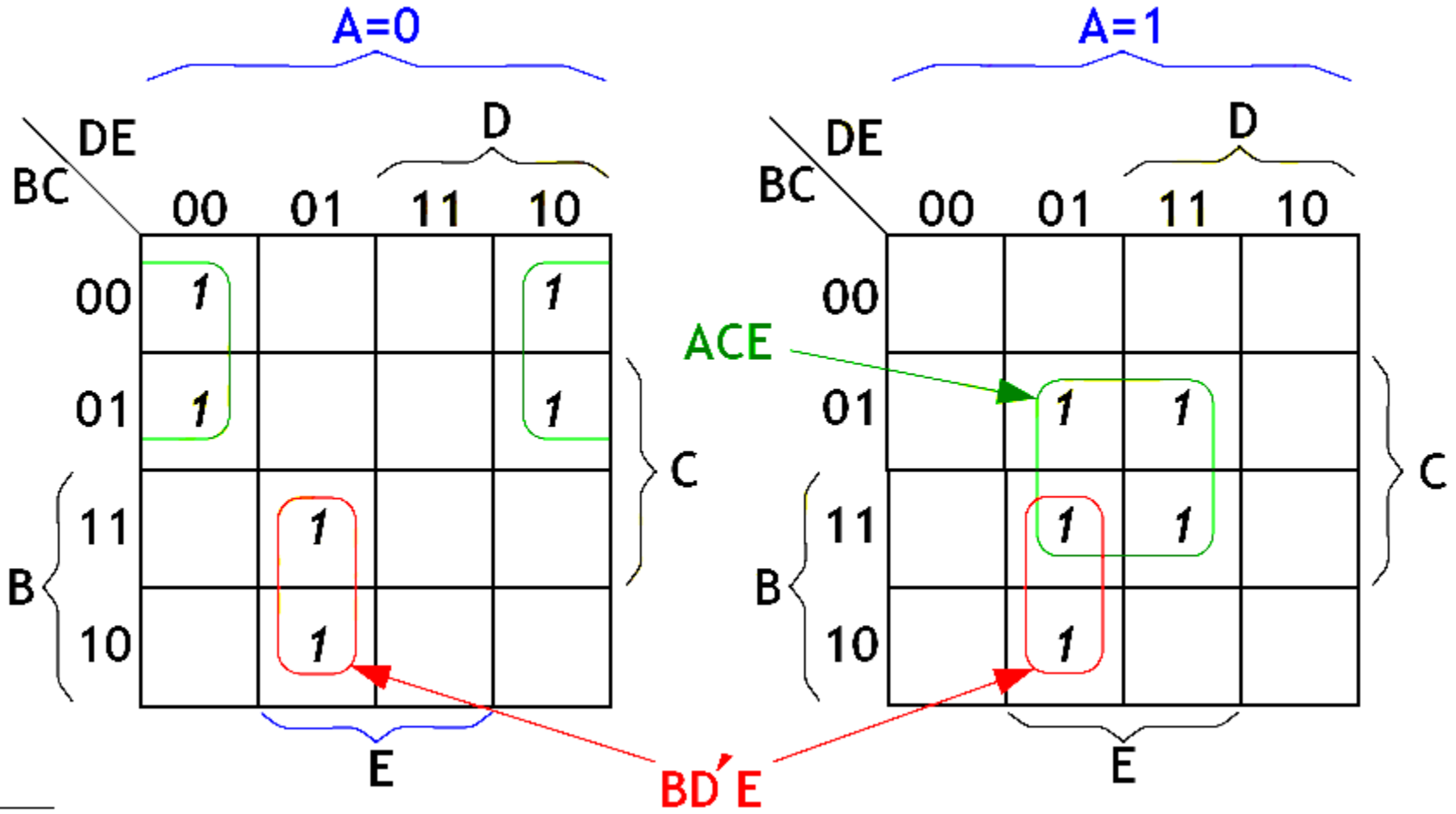
Five-variable K-map (F(A, B, C, D, E))

Presents the 32 minterms of a five-variable function as follows:



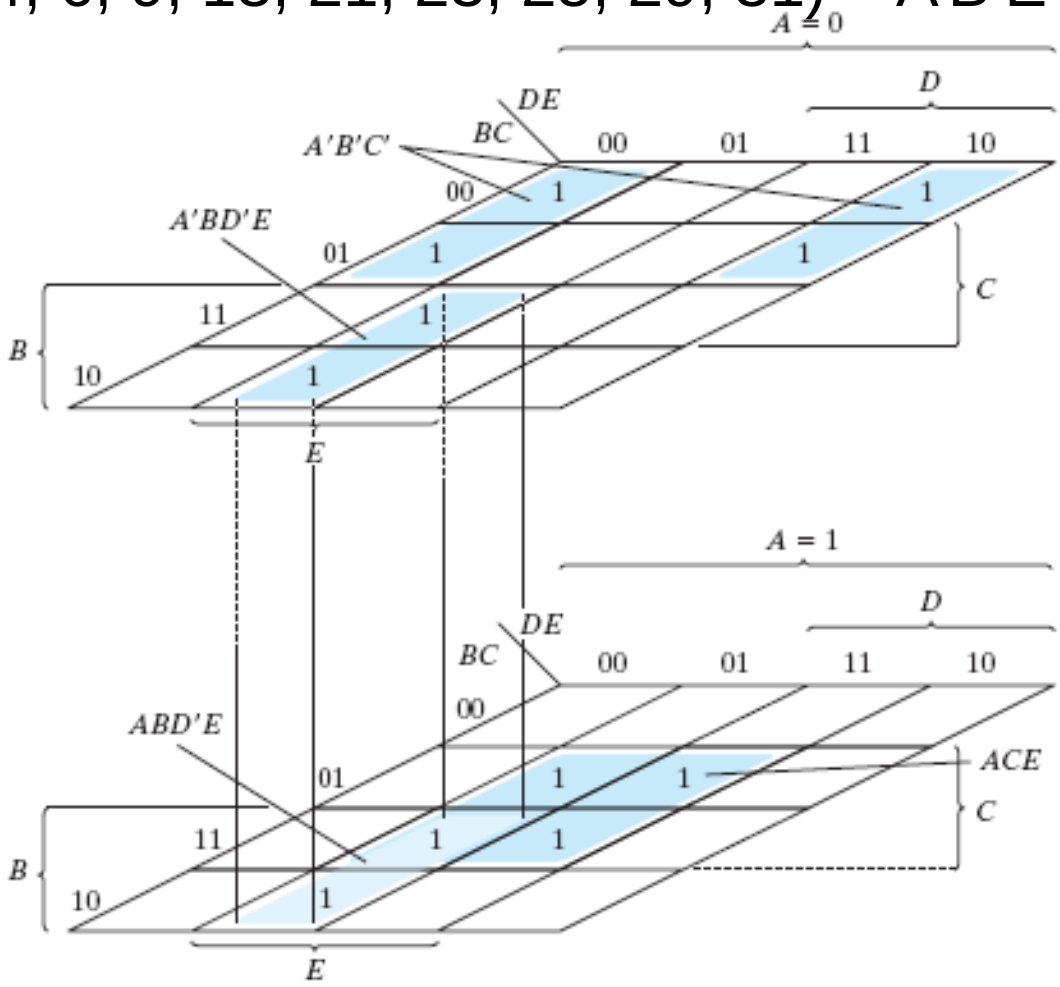
Example

F = ∑(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31) = A'B'E' + BD'E + ACE



Example

F = ∑(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31) = A'B'E' + BD'E + ACE



Simplify function in SoP and PoS

$$F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$$
$$= B'D' + B'C' + A'C'D$$

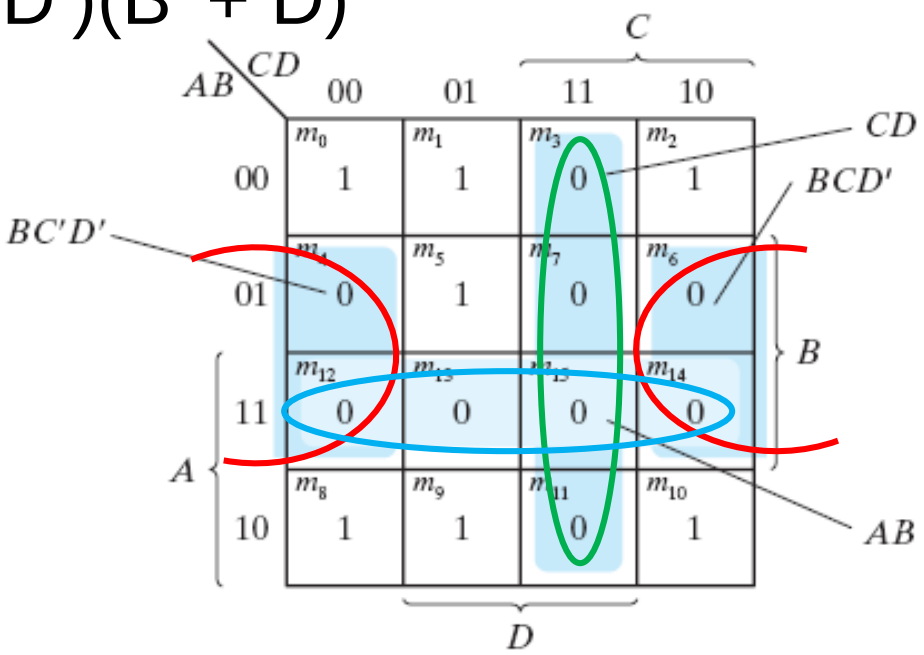
Simplify function in SoP and PoS

$F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

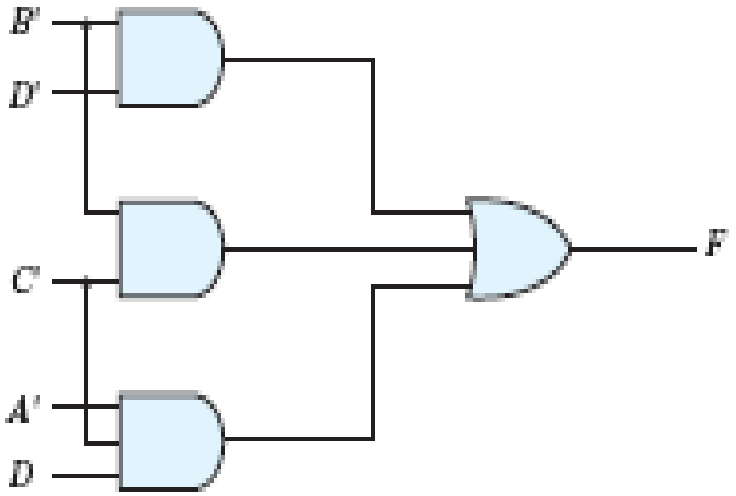
$= B'D' + B'C' + A'C'D$

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

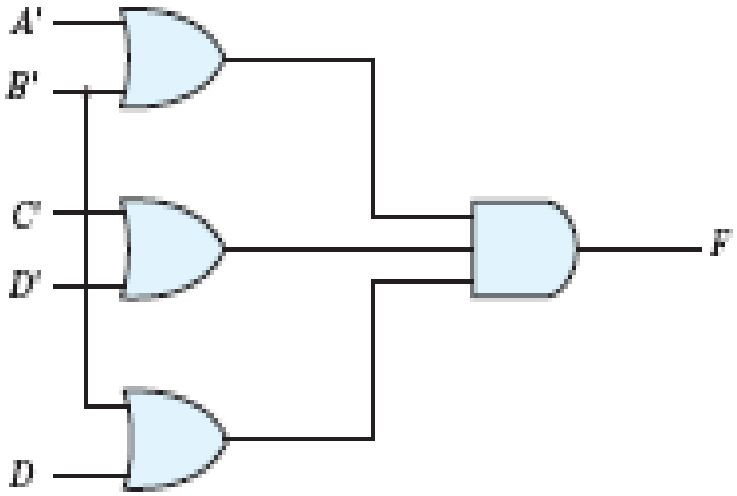
$= (A' + B')(C' + D')(B' + D)$



Gate Implementation



(a) $F = B'D' + B'C' + A'C'D$



(b) $F = (A' + B')(C' + D')(B' + D)$

Don’t-care Terms (x)

- A don’t care term is a combination of variables whose logical value is not specified.
- Example:

  BCD has 6 don’t-care terms

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	x
11	x
12	x
13	x
14	x
15	x

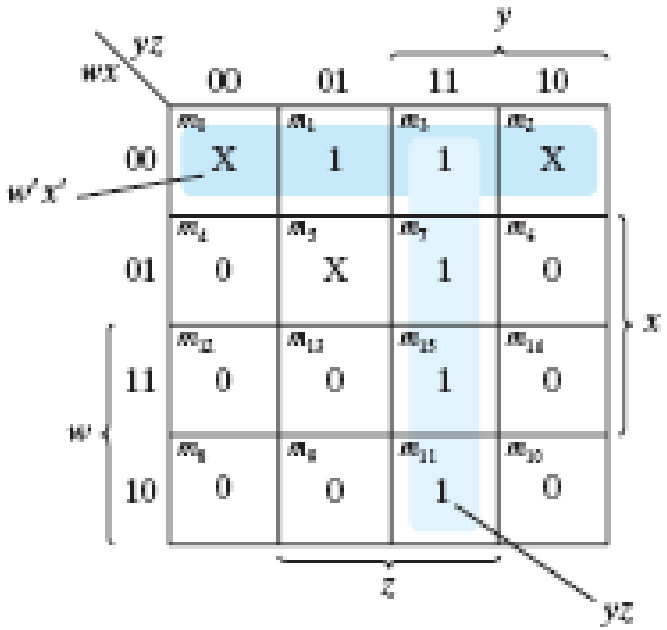
Simplification in Presence of Don’t-care Terms (x)

- A don’t care term can be assigned logic 1 or zero for simplification

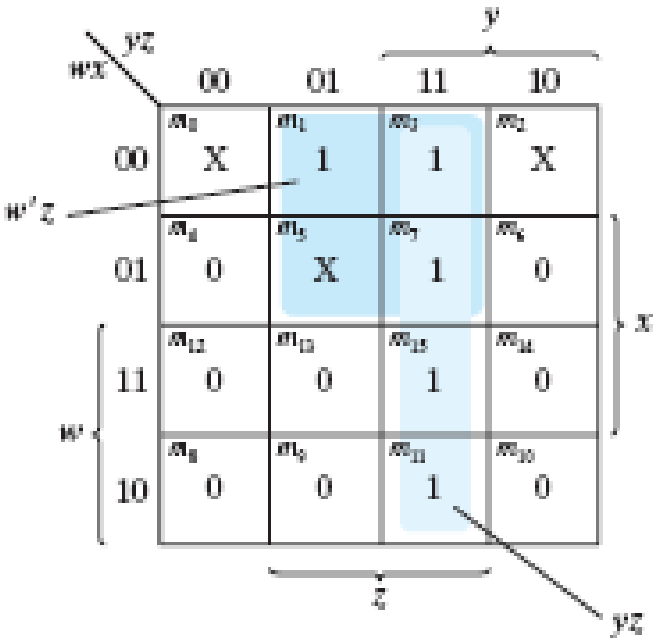
Example

F(w, x, y, z) = ∑(1, 3, 7, 11, 15)

With don't care terms d = ∑(0, 2, 5)



(a) $F = yz + w'x'$

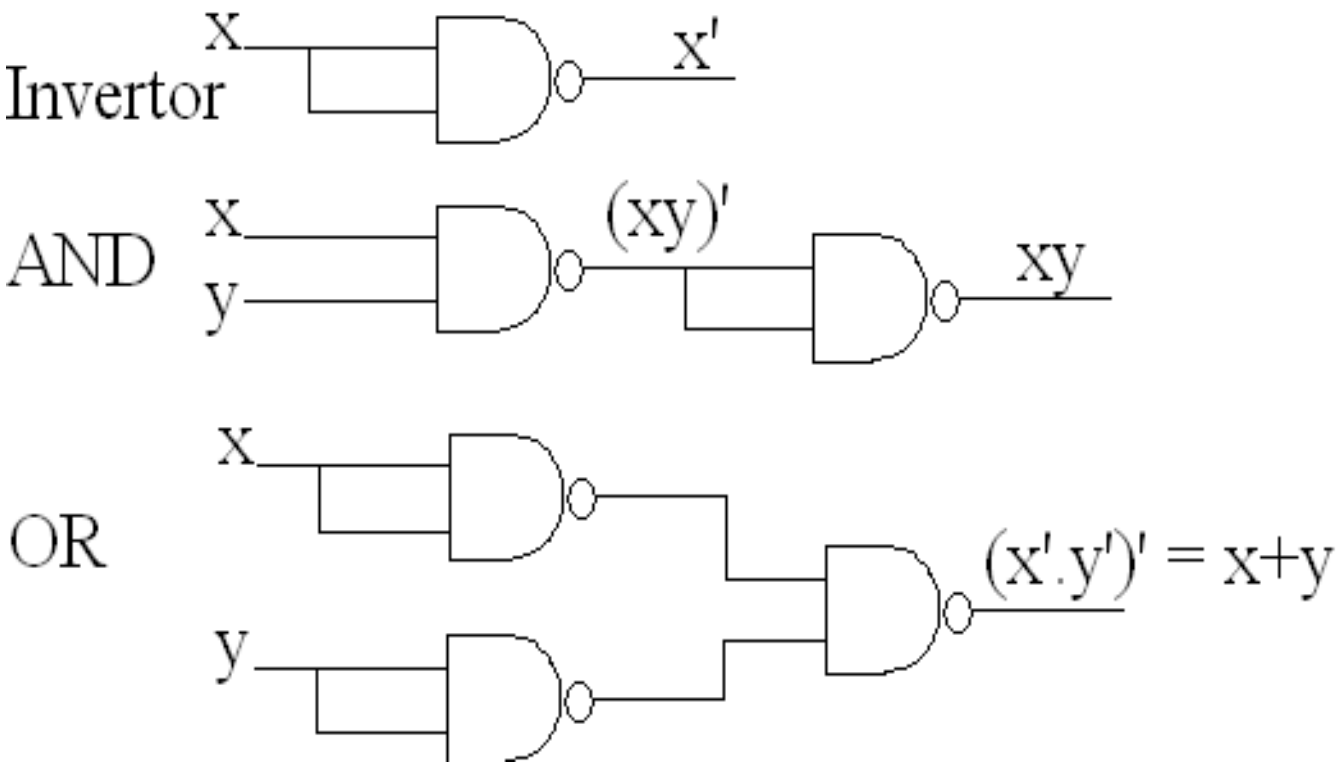


(b) $F = yz + w'z$

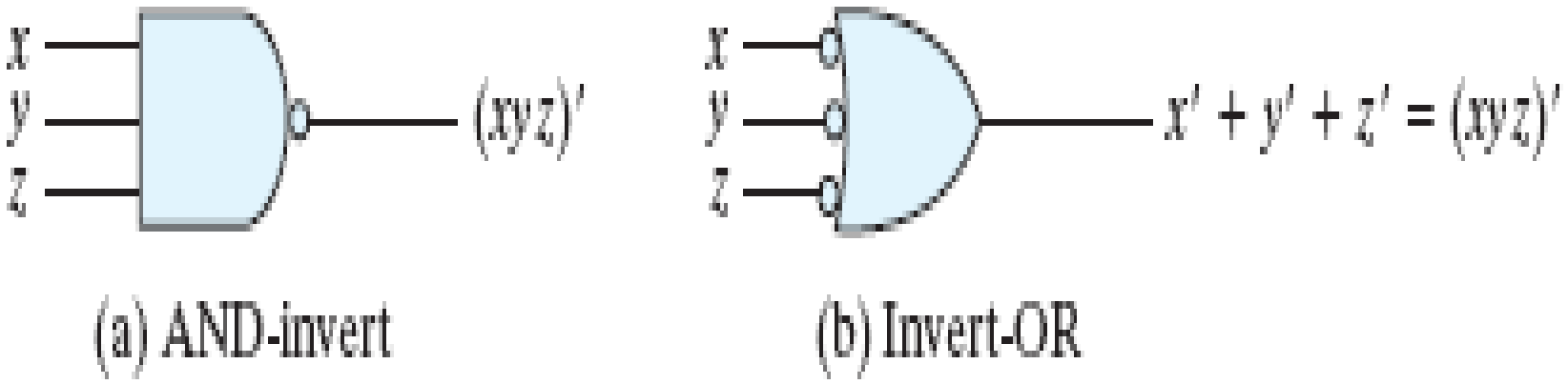
NAND and NOR Gates

- NAND and NOR gates are easier to fabricate with electronic components.
- NAND and NOR gates are universal gates; any combinational digital system can be implemented with NAND gates only or NOR gates only.
- NAND and NOR gates are the basic gates in all IC digital family.

NAND Gate



Two Symbols for NAND Gates

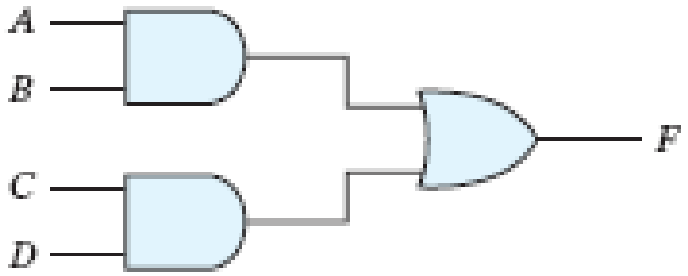


(a) AND-invert (b) Invert-OR

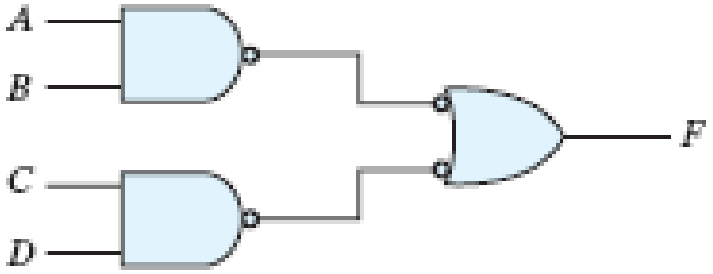
DeMorgan's: (xyz)' = x' + y' +z'

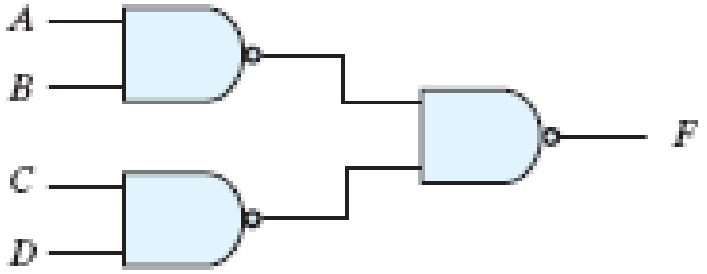
Two-Level Implementation with NAND Gates

F = AB + CD = ⌈(AB)'(CD)'⌉'



(a)

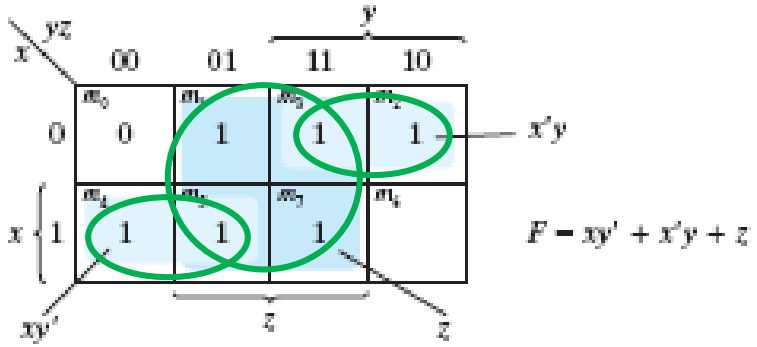


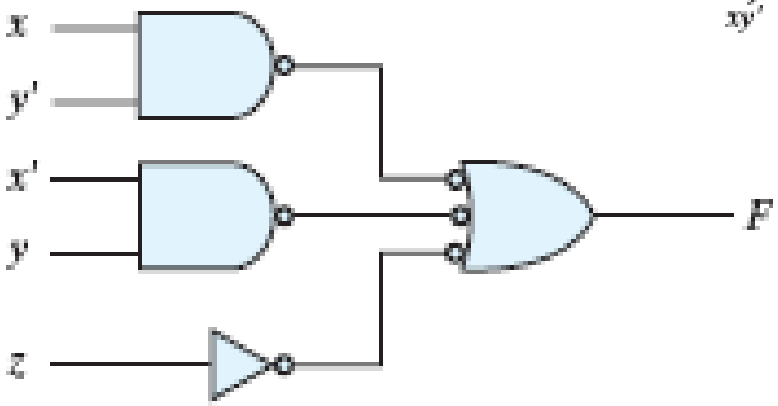


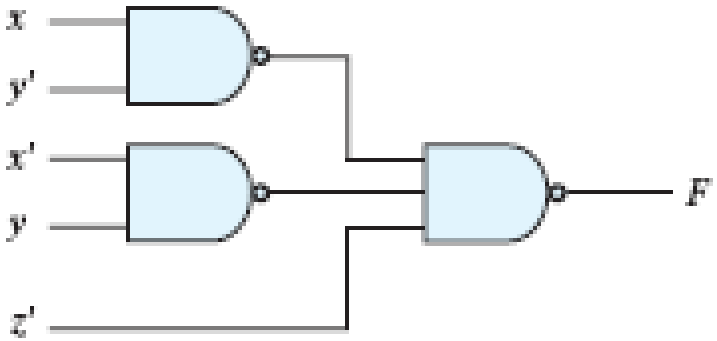
Example:

Implement F(x, y, z) = ∑(1, 2, 3, 4, 5, 7)

F = xy’ + x’y + z

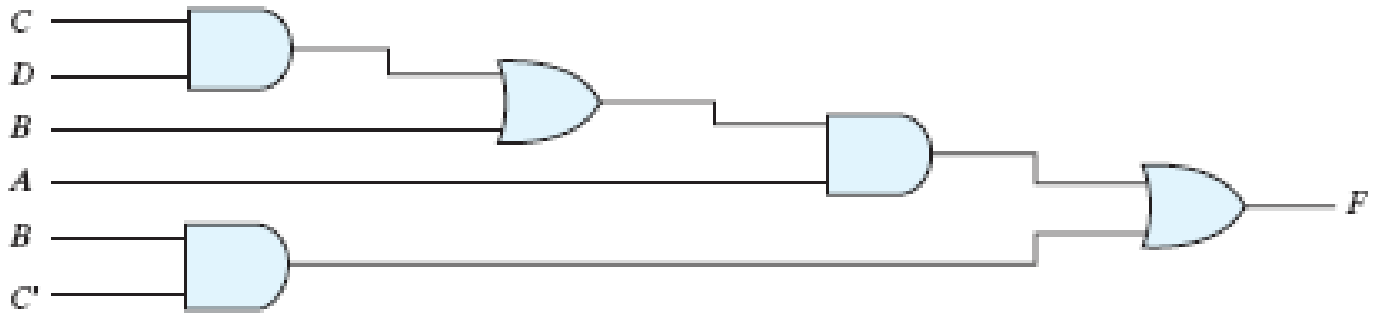




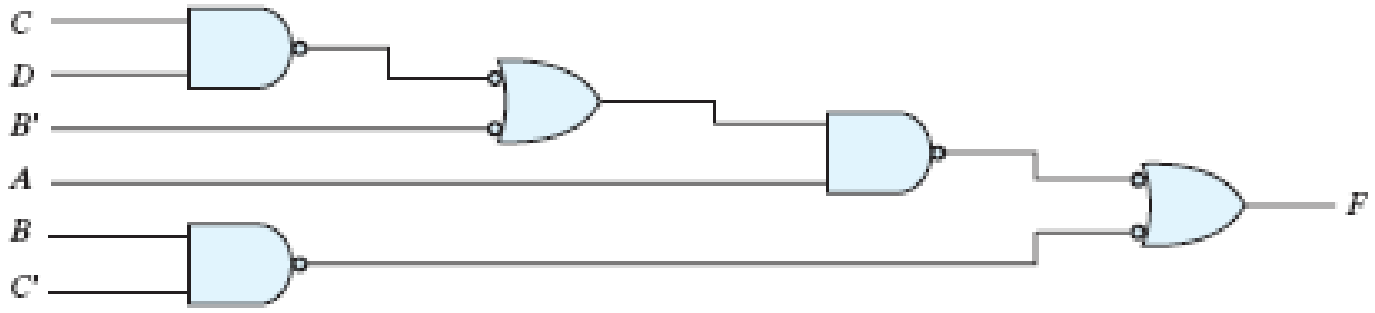


Example

F(A, B, C, D) = A(CD + B) + BC’



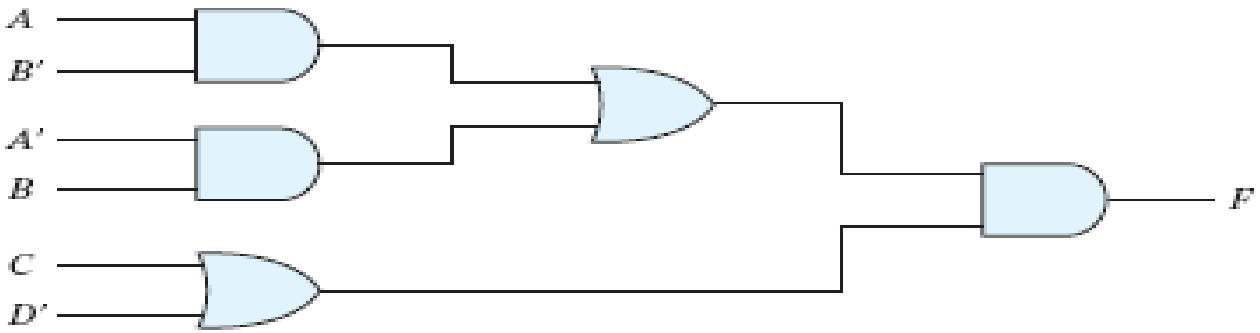
(a) AND–OR gates



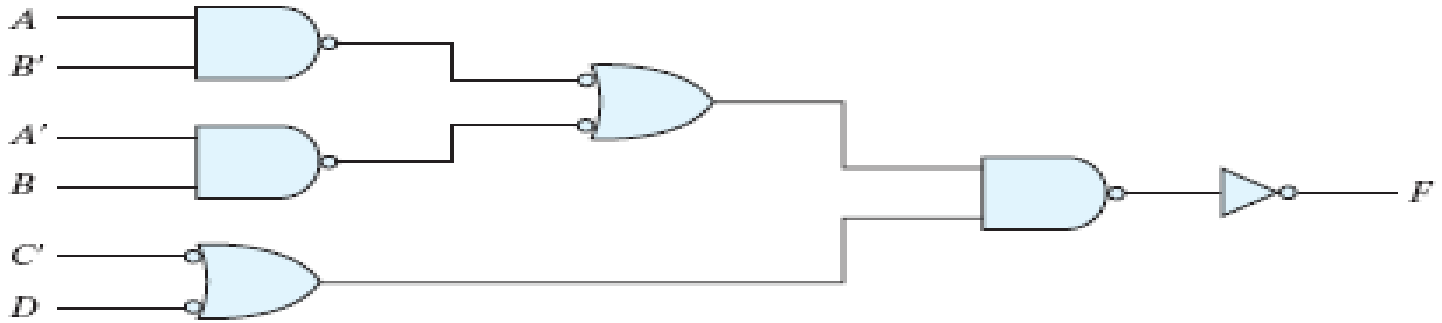
(b) NAND gates

Example

F(A, B, C, D) = (AB’ + A’B)(C+D’)

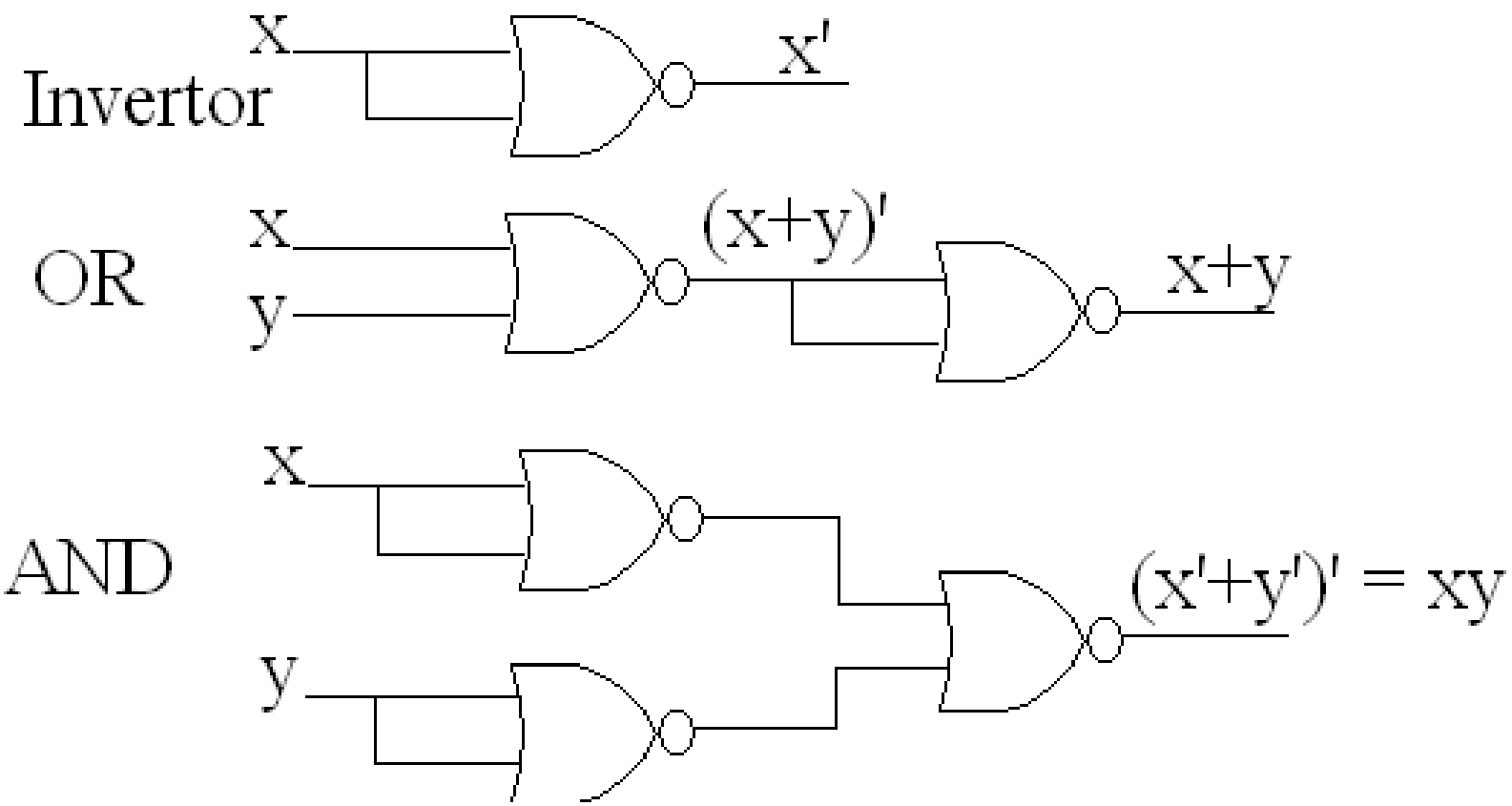


(a) AND–OR gates

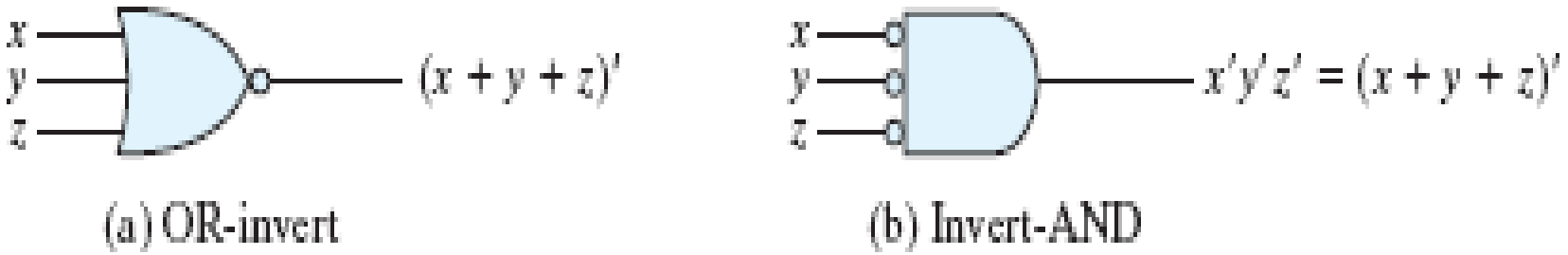


(b) NAND gates

NOR Gate



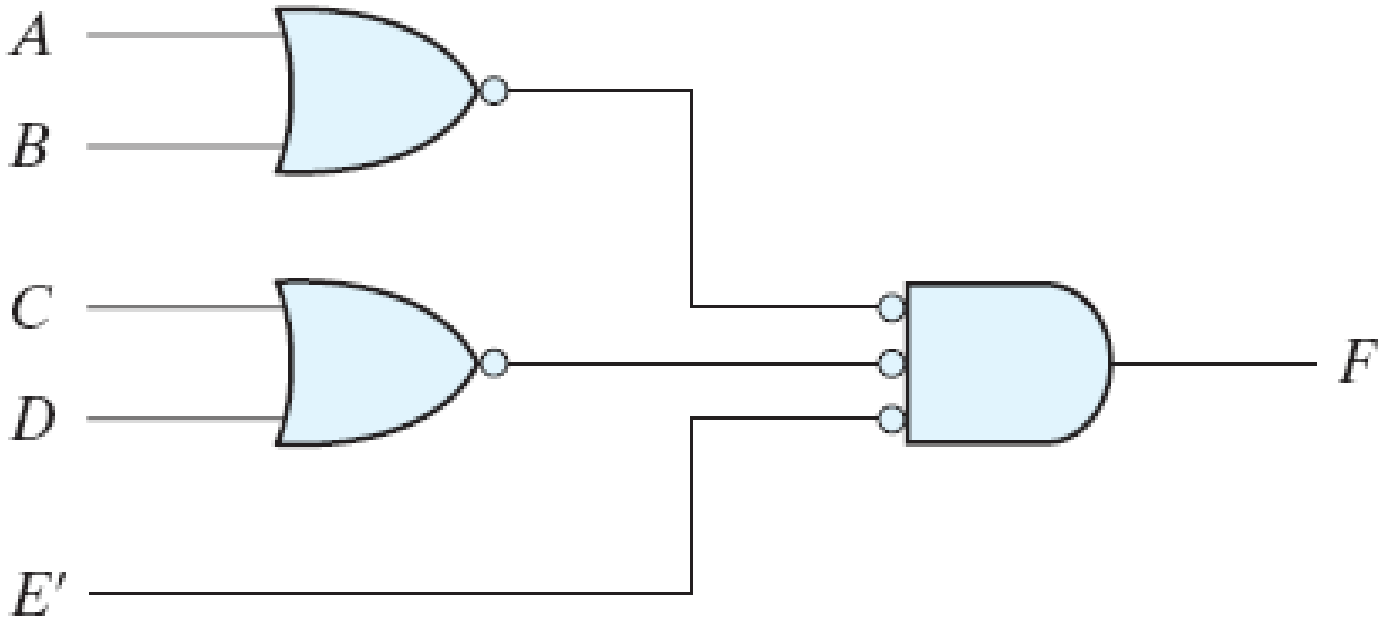
Two Symbols for NOR Gates



DeMorgan's: (x+y+z)' = x' . y' . z'

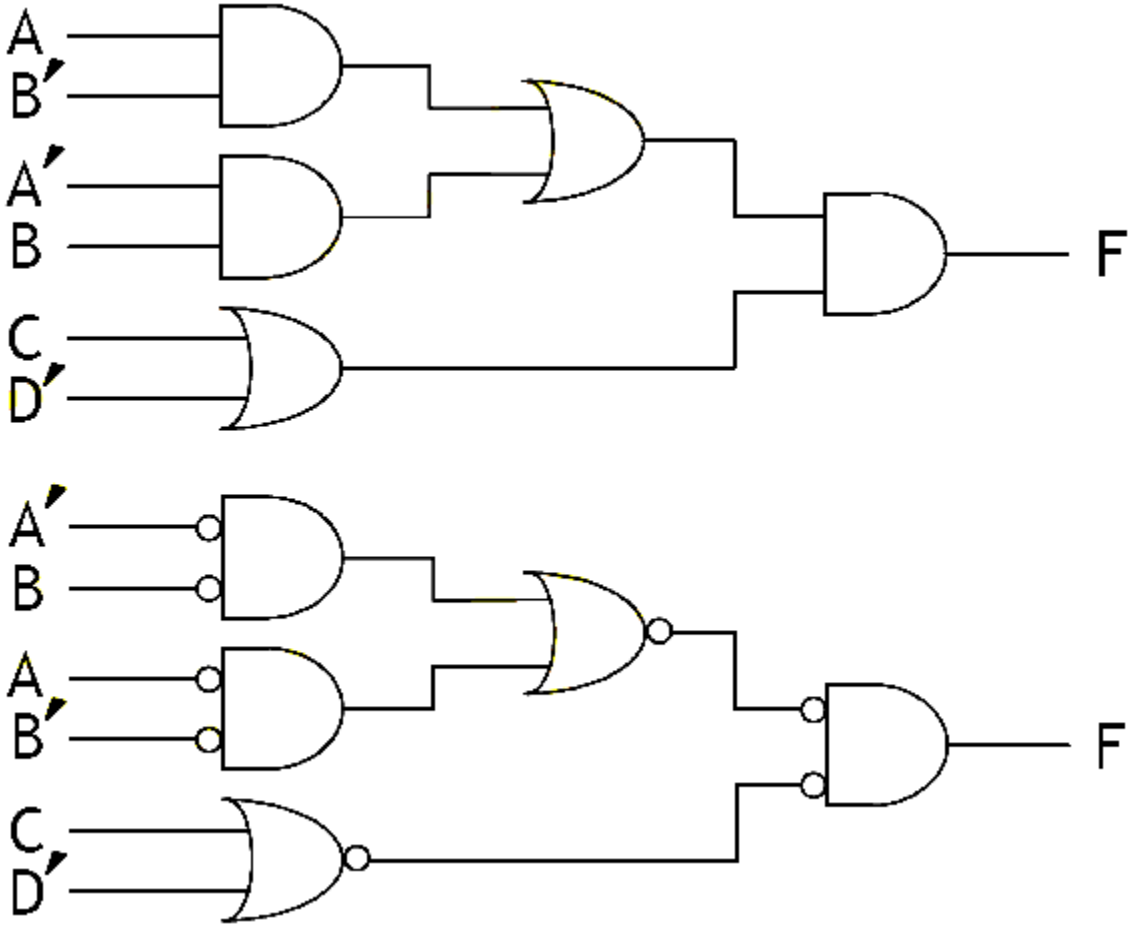
Two-Level Implementation with NOR Gates

F = (A+B)(C+D)E



Example

F(A, B, C, D) = (AB’ + A’B)(C+D’)



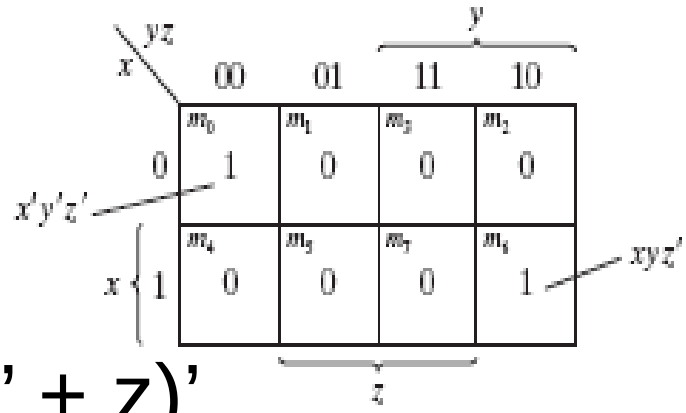
Implementation with other two-level forms

Implementation with Other Two-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.

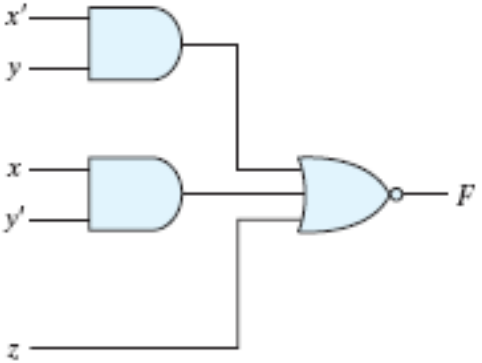
Implementation F with the 4 two-level forms



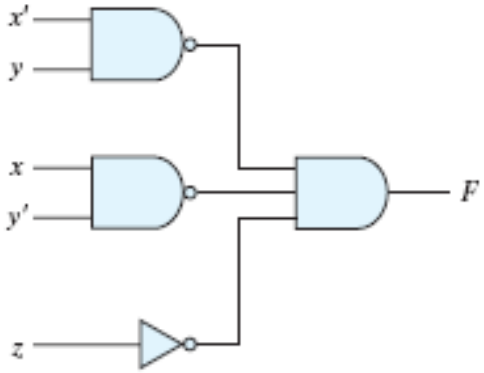
$F = x'y'z' + xyz'$

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

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

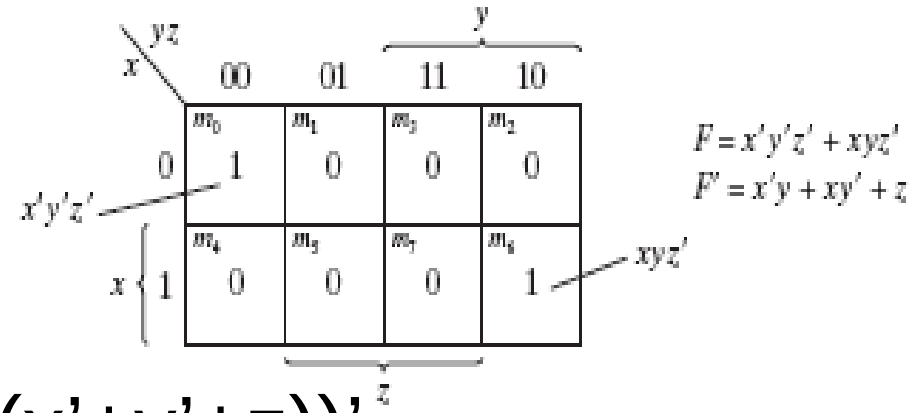


AND-NOR

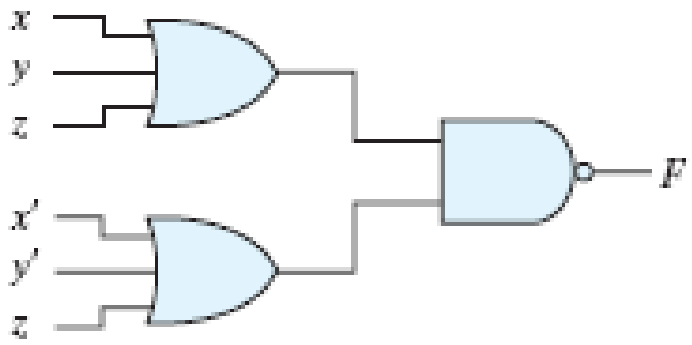


NAND-AND

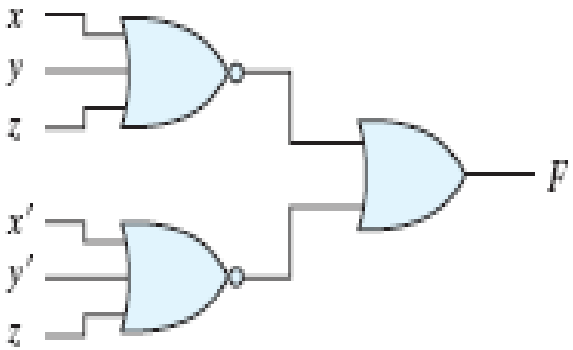
Implementation F with the 4 two-level forms



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



OR-NAND



NOR-OR

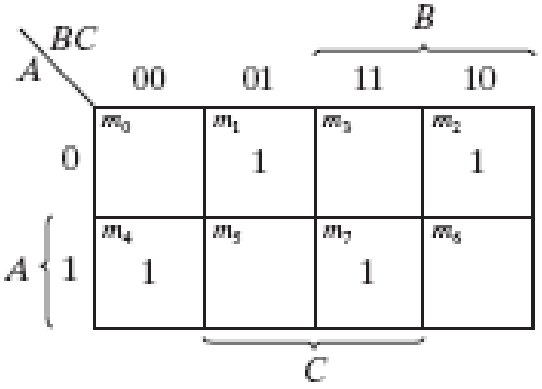
Exclusive-OR (XOR)

Exclusive-OR	$x \oplus y$	$xy' + x'y$
Exclusive-NOR	$(x \oplus y)'$	$xy + x'y'$

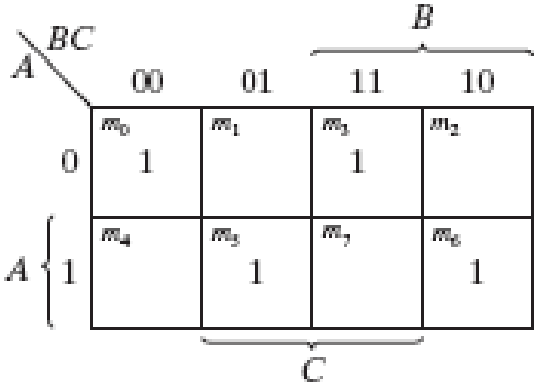
Commutative	$x \oplus y = y \oplus x$
Associative	$(x \oplus y) \oplus z = x \oplus (y \oplus z)$

Primitive Exclusive-OR Operations	$x \oplus 0 = x$
	$x \oplus 1 = x'$
	$x \oplus x = 0$
	$x \oplus x' = 1$
	$x \oplus y' = x' \oplus y = (x \oplus y)'$

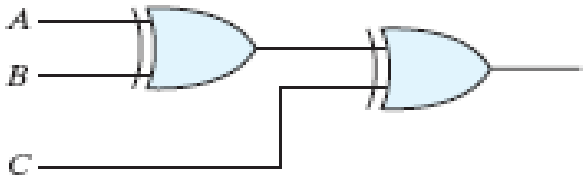
Odd versus Even function



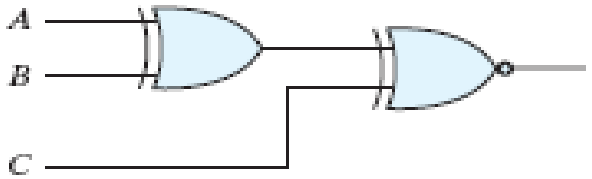
(a) Odd function $F = A \oplus B \oplus C$



(b) Even function $F = (A \oplus B \oplus C)'$

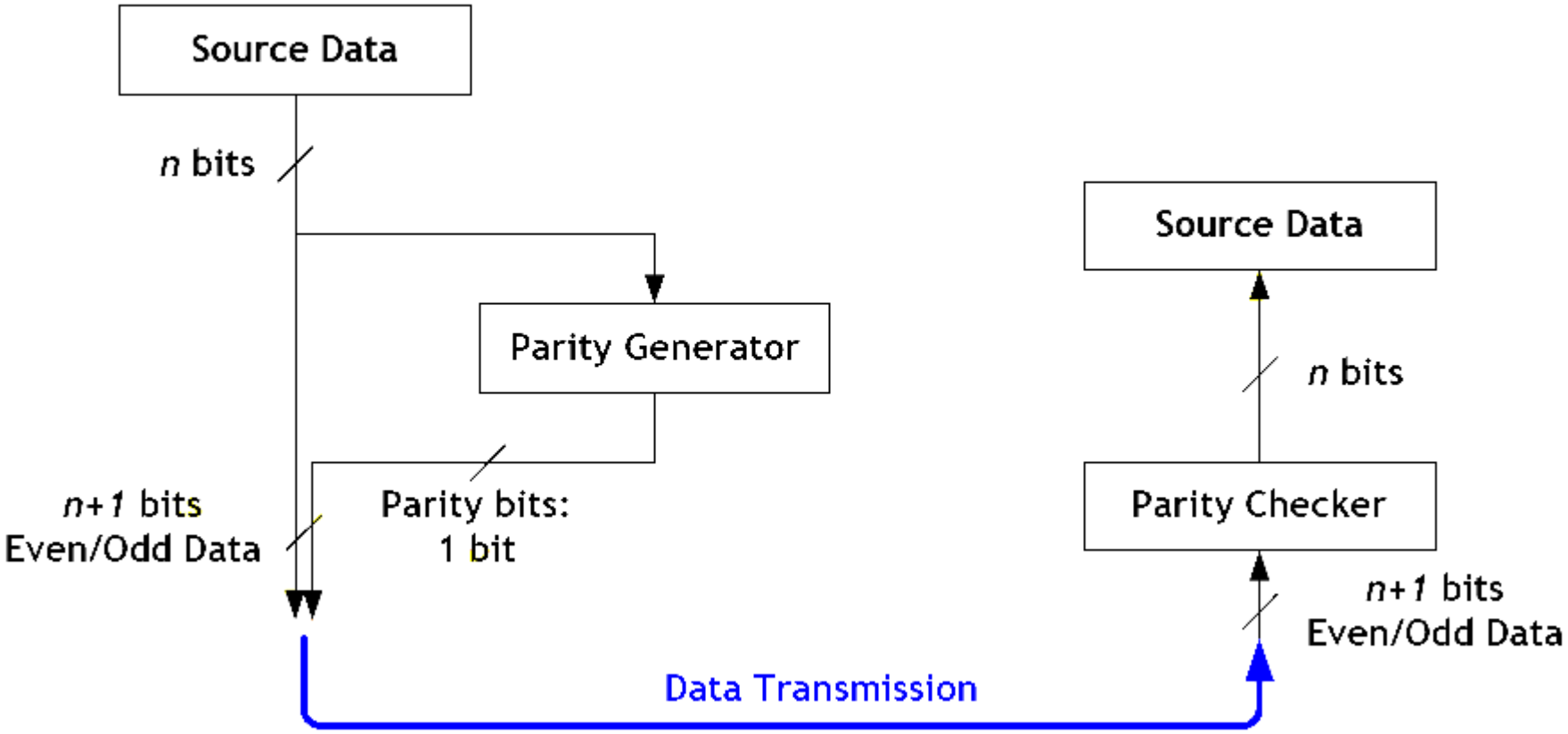


(a) 3-input odd function



(b) 3-input even function

Parity Data transmission System (Parity Generator/Checker)



Even Parity Generator

Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Even Parity Checker

Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Logic Implementation of Parity Generator and Checker

$P = x \oplus y \oplus z$

$C = x \oplus y \oplus z \oplus P$