Logic Design

Dr. Yosry A. Azzam

Gate Level Minimization

- Chapter 3

- Agenda
  - Simplification of Boolean Functions (The K-Map Method)
  - Don't Care Condition
  - Synthesis with NAND & NOR Gate
  - Brief on Gate Implementation
- Main Reading
  - Mano: Ch 3

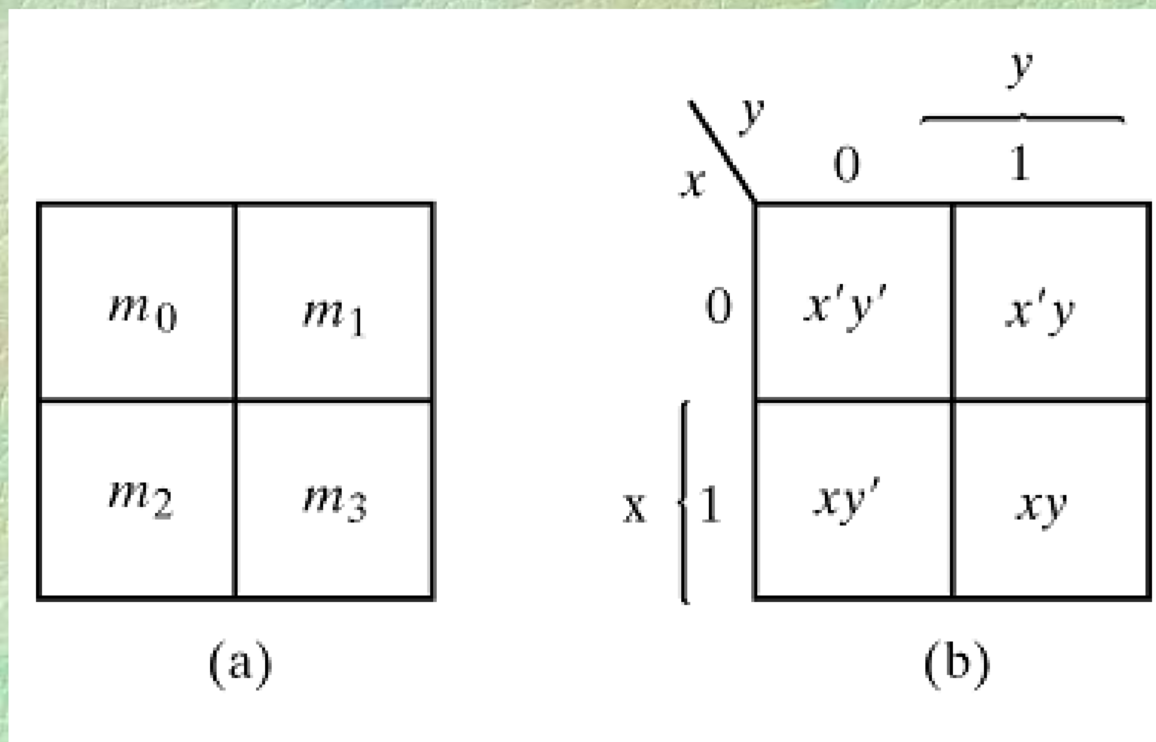
- Objectives
  - Understand the procedure of simplifying Boolean functions
  - Understand and able to perform the K-Map method
  - Understand the Don't Care Condition and their place in K-Map Method
  - Understand and able to implement design in NAND and NOR Gate
  - Understand the basic of Gate Implementation

The Map Method

- Provides a simple straightforward procedure for minimizing Boolean functions
  - Proposed by Veitch (Veitch Diagram), modified by Karnaugh (Karnaugh Map)
  - Why bother?
    - Simplifying the function = minimizing the amount of gates
    - Industrial requirements for efficiency in mass production

2-Variable Map

- The Map represents a visual diagram of all possible ways a function may be expressed in a standard form

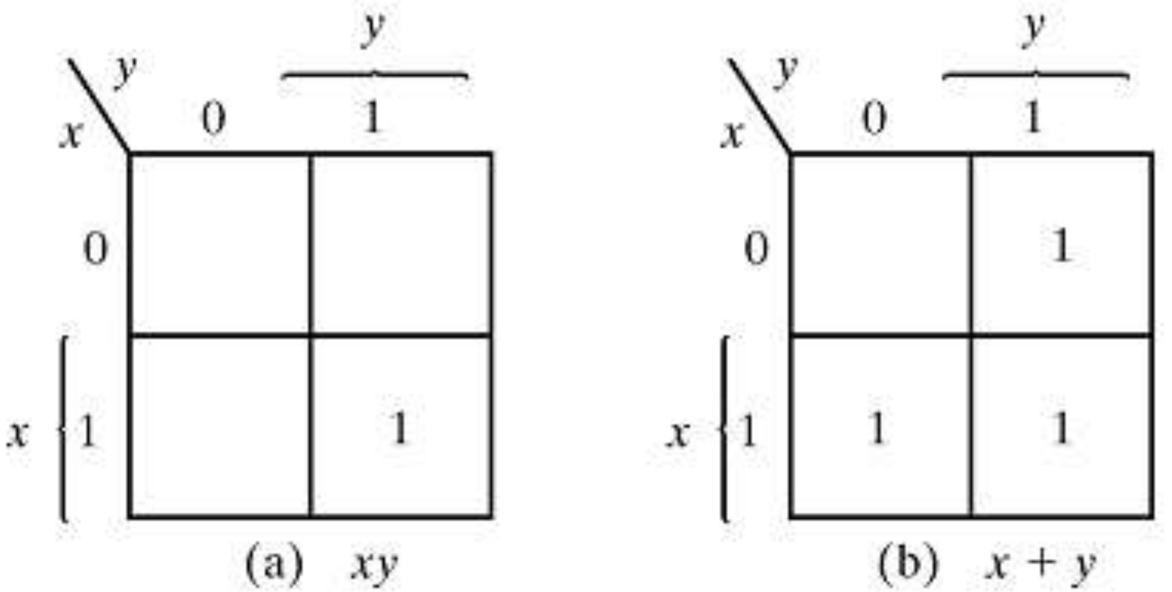


Two-variable Map

2-Variable Map

Representing Function in the map

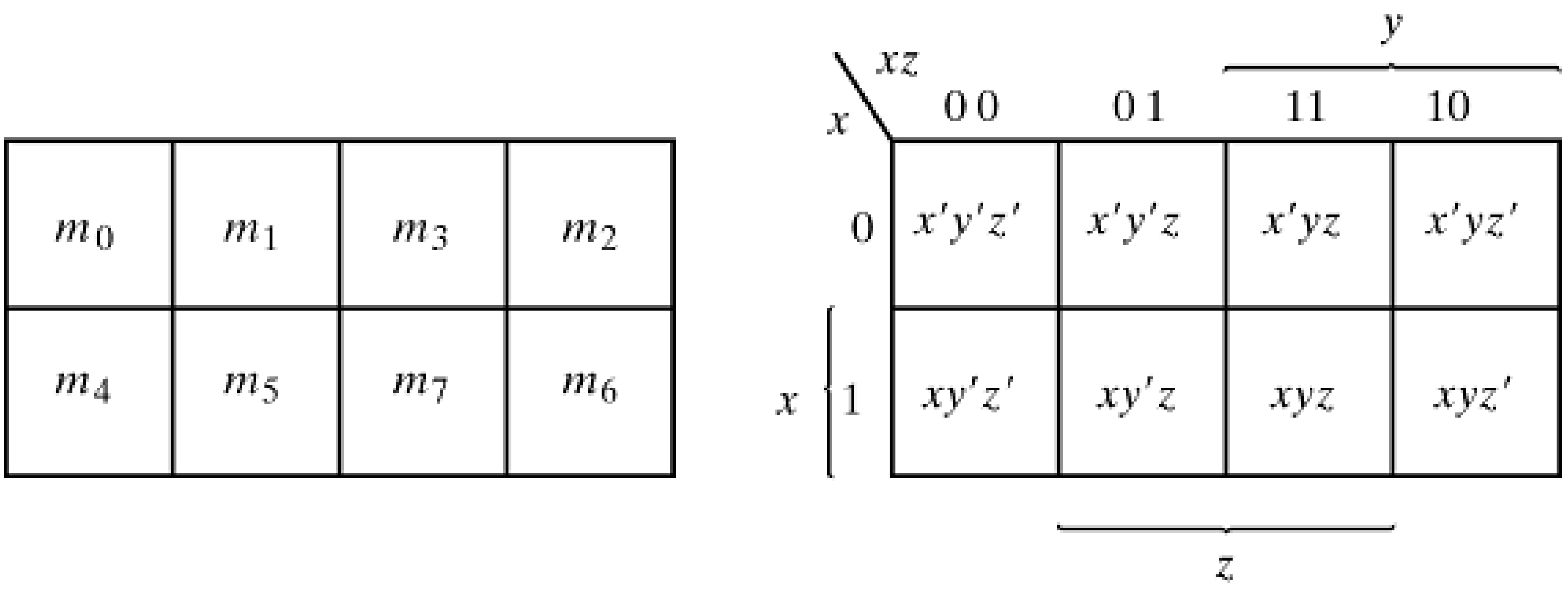
- **F= x.y**
- **F= x+y = x′y + xy′ + xy**



Representation of Functions in the Map

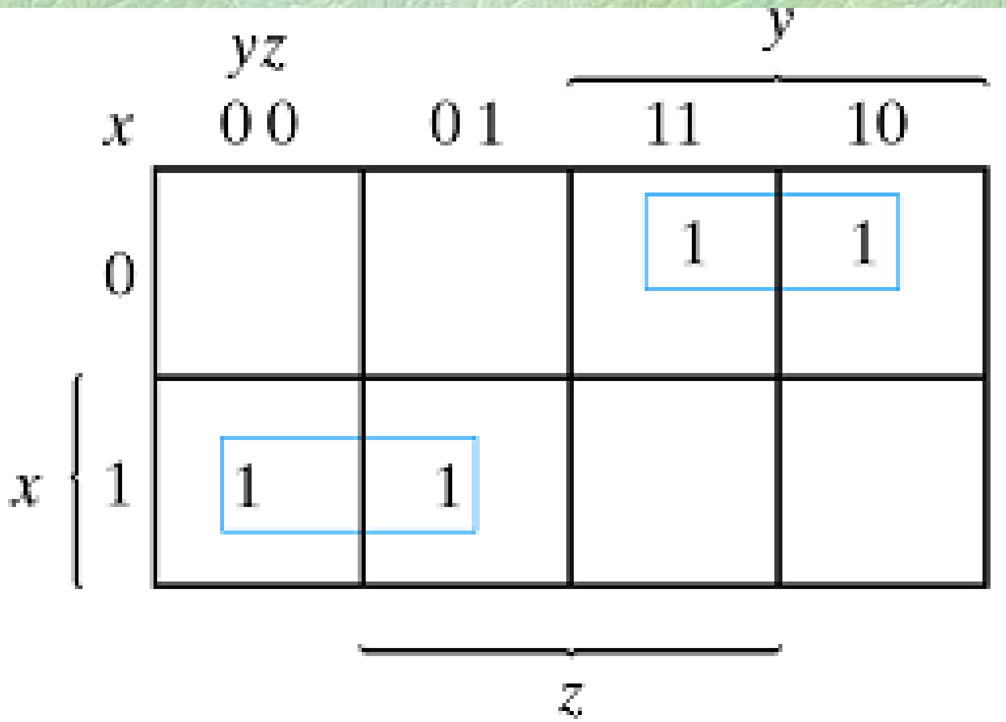
3-Variable Map

- The Map represents a visual diagram of all possible ways a function may be expressed in a standard form



(a) (b)

3-Variable Map : Example F(x,y,z)



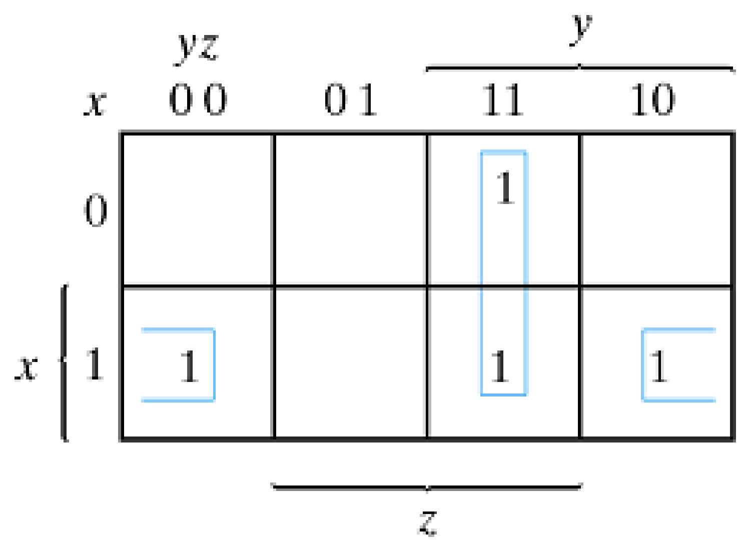
Map for $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

3-Variable Map rules of combination

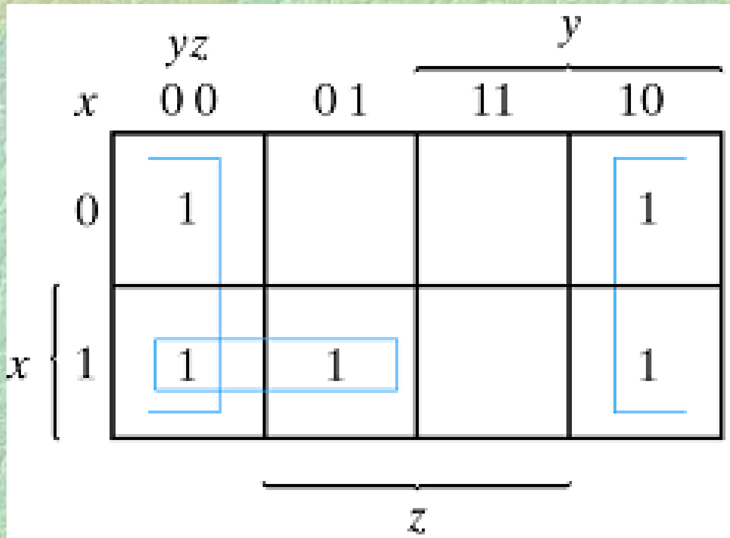
- One square represents one minterm, giving a term of 3 literals.
- Two adjacent squares represent a term of 2 literals
- Four adjacent squares represent a term of 1 literal.
- Eight adjacent squares encompass the entire map and produce a function that always equal to 1.

3-Variable Map :

Other Examples F(x,y,z)



$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$



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

3-Variable Map :

Other Examples F(x,y,z)

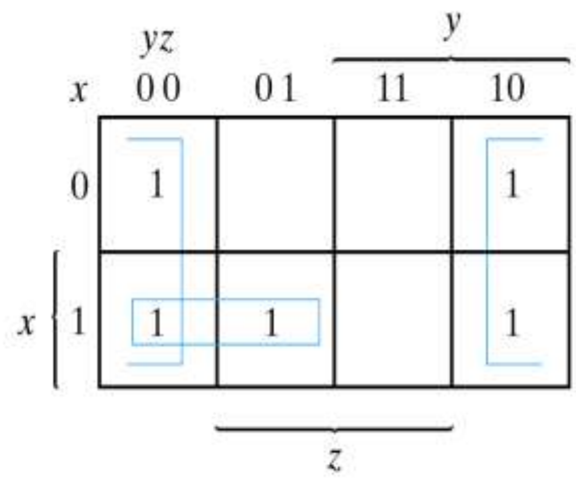
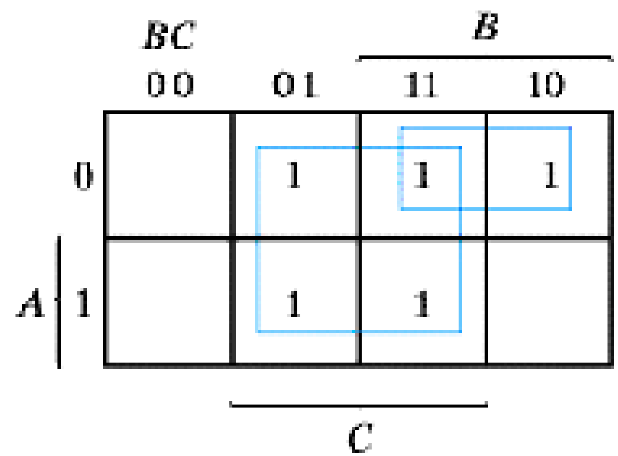


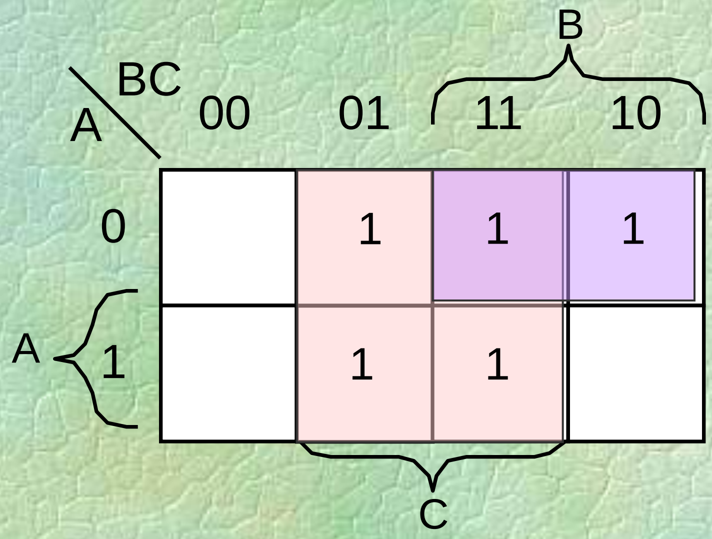
Fig. 3-6 Map for Example 3-3; $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$



$F(A, B, C) = A'C + A'B + AB'C$
$+ BC = C + A'B$

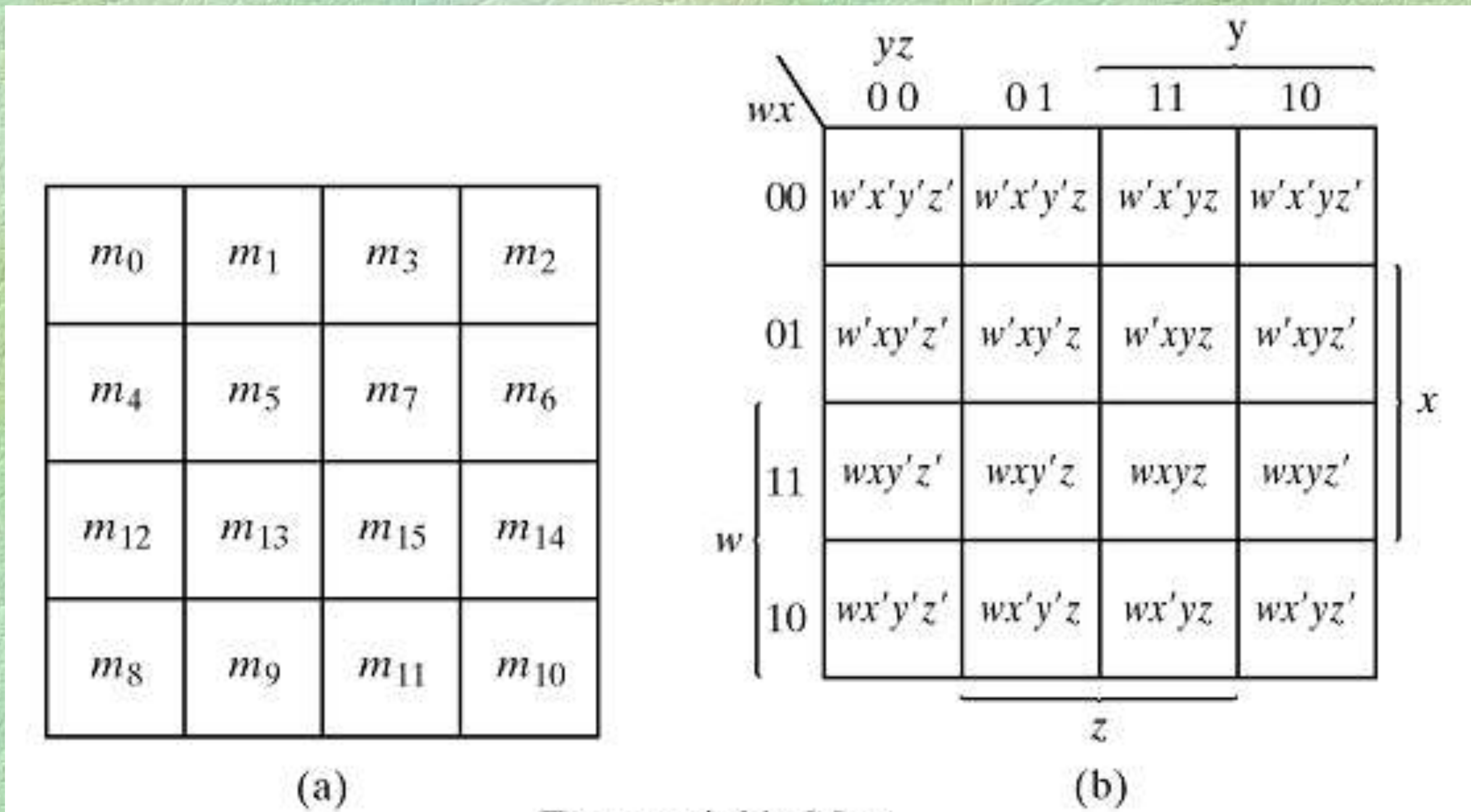
Simplifying using the Map

- $F = A'C + A'B + AB'C + BC$
  - Plot the expression
  - Find minimum adjacent squares
    - Prime Implicant
    - Essential Prime Implicant
  - Draw them
  - Write the expression



$$F = C + A'B$$

4-Variable Map



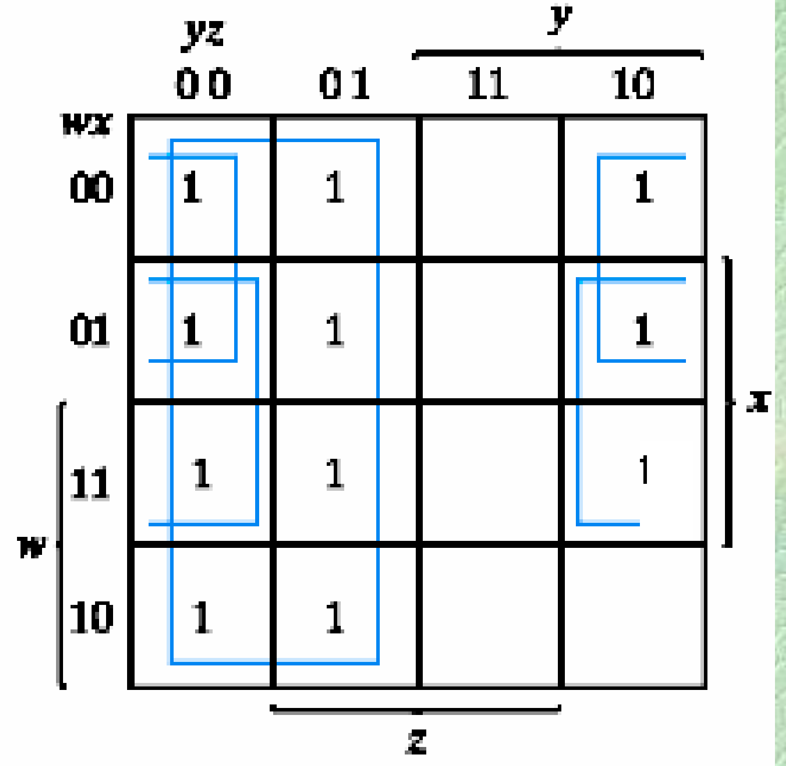
Four-variable Map

4-Variable Map rules of combination

- One square represents one minterm , giving a term of 4 literals.
- Two adjacent squares represent a term of 3 literals.
- Four adjacent squares represent a term of 2 literals.
- Eight adjacent squares represent a term of 1 literal.
- Sixteen adjacent squares represent the function equal to 1.

4-Variable Maps (Example)

- F(w,x,y,z) = ∑(0,1,2,4,5,6,8,9,
- 12,13,14)
- 0000, 0001, 0010, 0100, 0101, 0110, 1000, 1001, 1100, 1101, 1110
- ***f(w,x,y,z)=y'+w'z'+xz'***



Map for Example 3-5; *F*(*w*, *x*, *y*, *z*) = Σ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = *y*' + *w*'*z*' + *xz*'

4-Variable Maps (Example)

- Simplify the Boolean Function:

F= A′B′C′ + B′CD′ + A′BCD′ + AB′C′

Solution:

The simplified function is:

F=B′D′ + B′C′ + A′ CD′

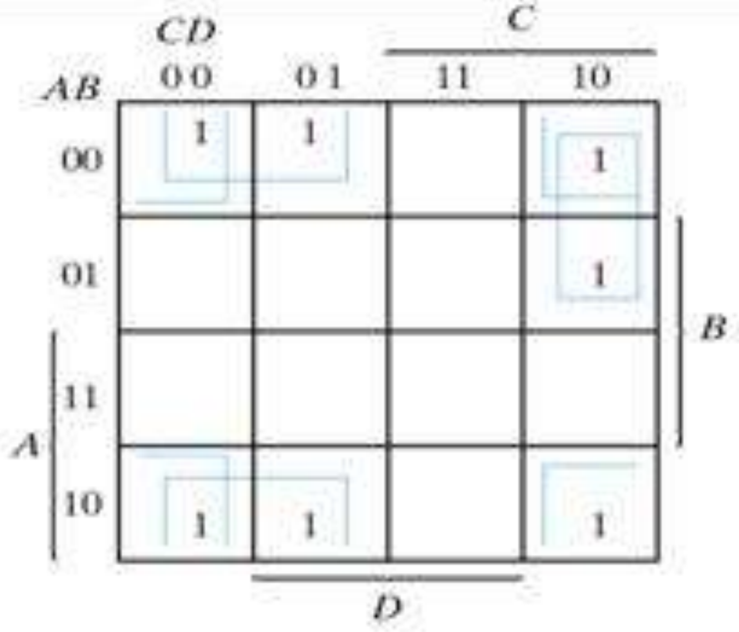


Fig.3-10 Map for Example 3-6; $A'B'C + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

5-Variable Map

$A = 0$

$BC \backslash DE$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$A = 1$

$BC \backslash DE$	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Fig. 3-12 Five-variable Map

5-variable Map

F(w,x,y,z) = ∑(0,2,4,6,9,13, 21, 23, 25, 29,31)

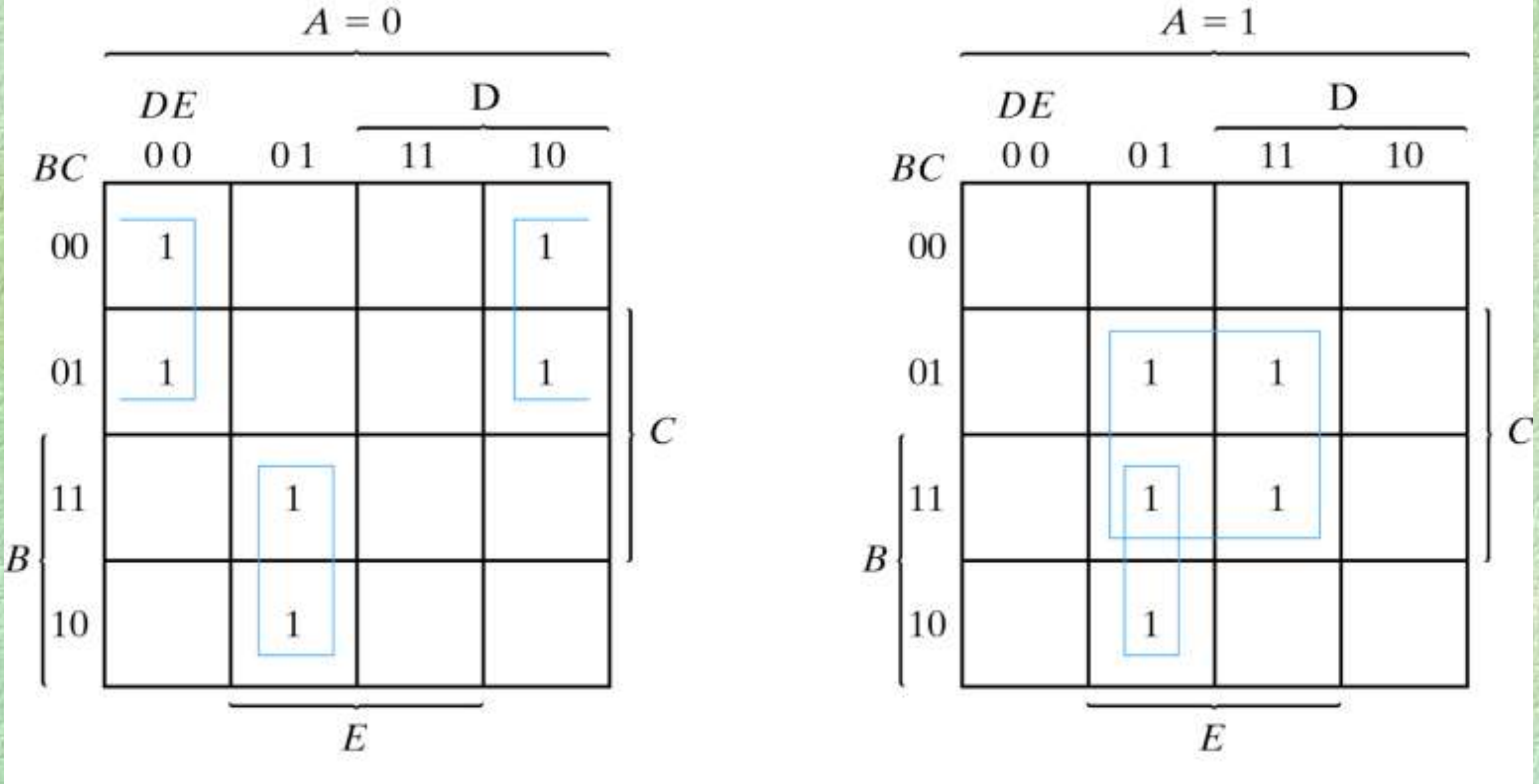


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

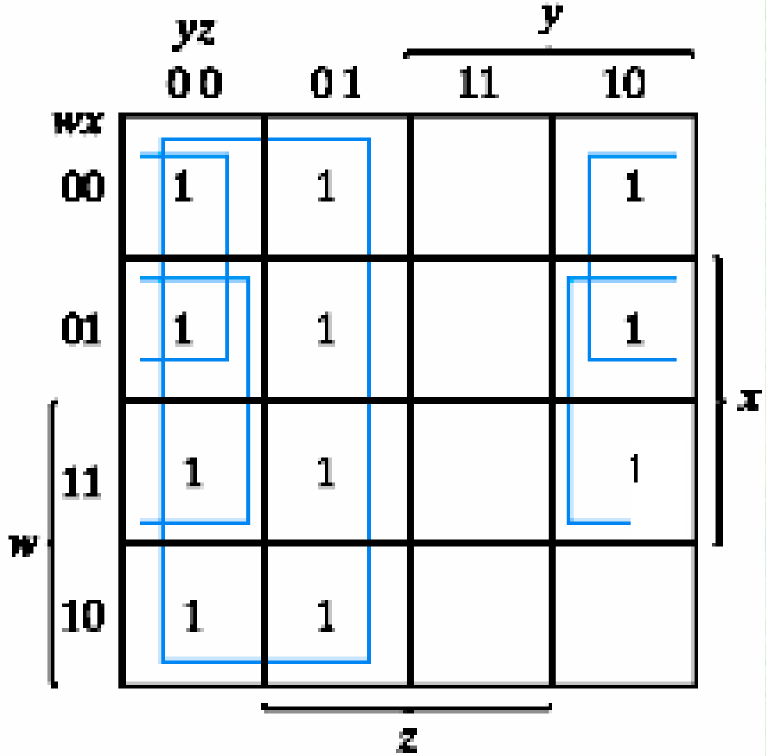
Product of Sum Simplification

- F(w,x,y,z) = ∑(0,1,2,4,5,6,8,9,
- 12,13,14)
- 0000, 0001, 0010, 0100, 0101, 0110, 1000, 1001, 1100, 1101, 1110

wx \ yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

Product of Sum Simplification

- F(w,x,y,z) = ∑(0,1,2,4,5,6,8,9,
- 12,13,14)
- 0000, 0001, 0010, 0100, 0101, 0110, 1000, 1001, 1100, 1101, 1110
- ***f(w,x,y,z)=y'+w'z'+xz'***



Map for Example 3-5; *F*(*w*, *x*, *y*, *z*)
= Σ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)
= *y*′ + *w*′*z*′ + *xz*′

Product of Sum Simplification

$F' = yz + wx'y$

$F = (F')'$

$F = (yz + wx'y)'$

$F = (yz)'(wx'y)'$

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

wx \ yz	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

Are they the Same?

- $F = y' + w'z' + xz'$ Normal Simplification (Sum of Product)
- $F' = yz + wx'y$
  - $(F')'$
  - $(yz + wx'y)'$
  - $(yz)'(wx'y)'$
  - $(y'+z')(w'+x+y')$ Product of Sum Simplification
  - $y'w' + y'x + y'y' + z'w' + z'x + z'y'$
  - $y'(w' + x + z' + y') + z'w' + z'x$
  - $y' + z'w' + z'x$

Product of sums simplification

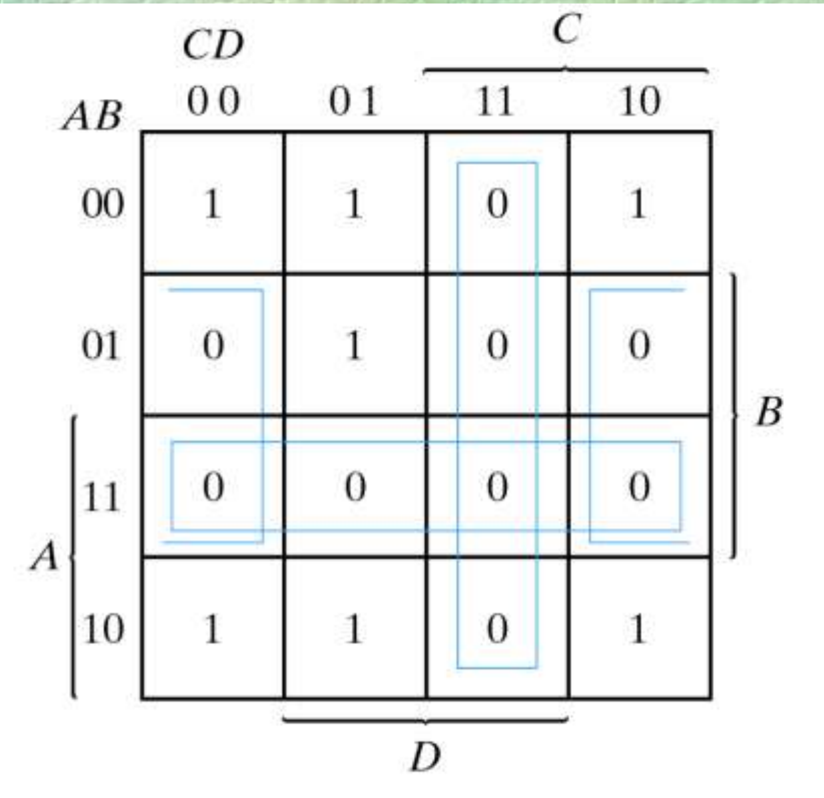
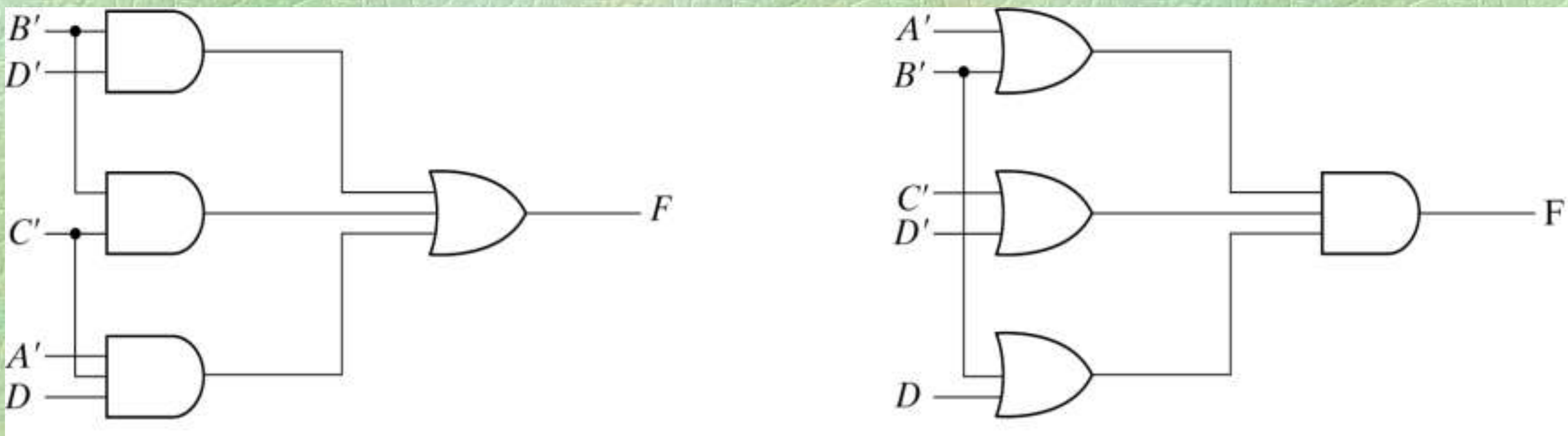


Fig. 3-14 Map for Example 3-8; $F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$

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

Gates Implementation : example



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

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

Fig. 3-15 Gate Implementation of the Function of Example 3-8

Don’t Care Conditions :

- Sometimes a certain combination of inputs will never be evaluated by your digital system, thus a “Don’t care” is placed for those valuation
  - E.g. consider a BCD (**Binary Coded Decimal**) number, there are 4 binary variables b_3, b_2, b_1, b_0 that represents decimal 0 to 9. design a system that detect if the BCD input given is divisible with 3
    - 4 bits has 16 combinations, but only 10 are used to represent decimal 0 to 9, the remaining combinations are not used.
    - System will produce 1 if the BCD is divisible by 3.

Don't Care Example

b_3b_2 \ b_1b_0	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	d	d	d	d
10	0	1	d	d

Decimal Representation	Binary				f
	b_3	b_2	b_1	b_0	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
Unused	1	0	1	0	d
Unused	1	0	1	1	d
Unused	1	1	0	0	d
Unused	1	1	0	1	d
Unused	1	1	1	0	d
unused	1	1	1	1	d

Simplifying With Don't Cares

$$b_2b_1b_0' + b_2'b_1b_0 + b_3b_0$$

You can either use or not use the don't care cell
(it can be treated like a "1" if it can produce more efficient result)

b_3b_2 \ b_1b_0	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	d	d	d	d
10	0	1	d	d

So What Does Don't Care Means?

- We simply don't care what the function values are for the unused input valuation Denote by "d" or "x"

- Keep in mind to use as minimum amount of terms as possible

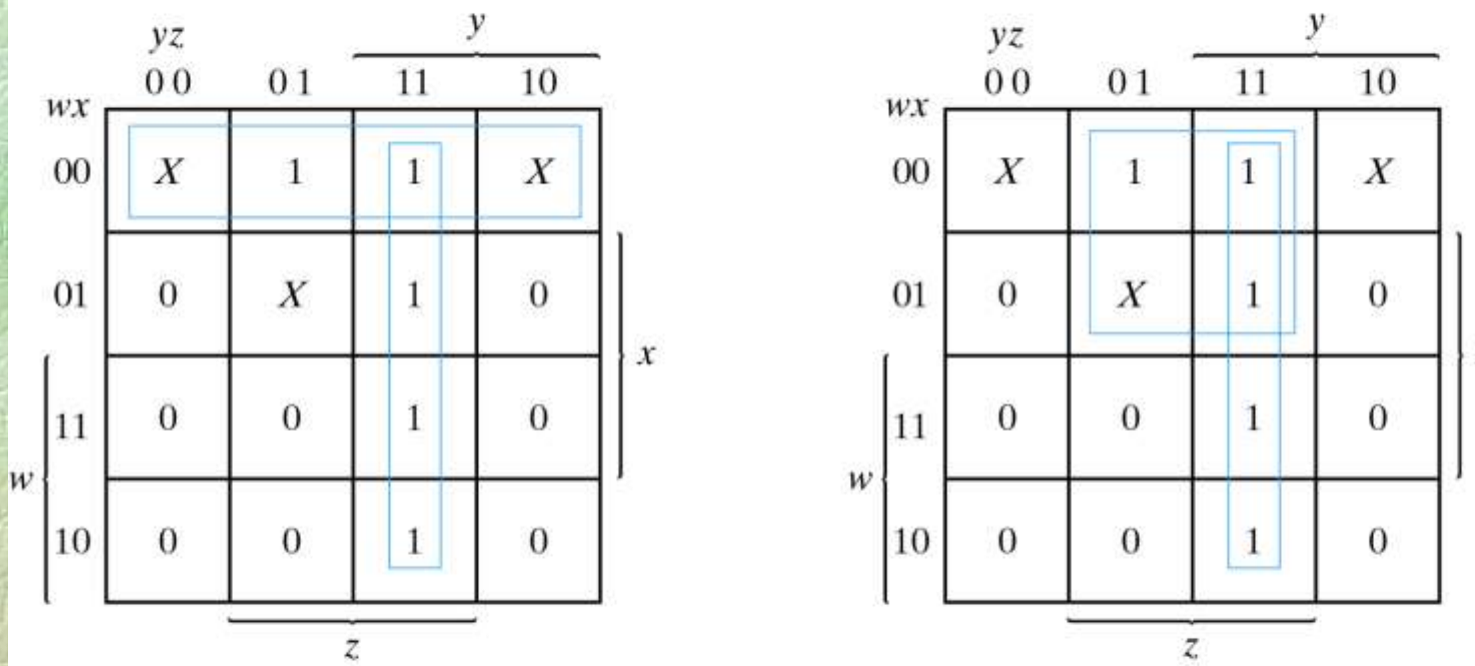
Example with don’t Care condition

Simplify :

F(W,X,Y,Z)= ∑(1,3,7,11,15)

With the Don’t care conditions of:

d(w,x,y,z)= ∑(0,2,5)



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

Fig. 3-17 Example with don’t-care Conditions

***F(w,x,y,z)*= *yz*+*w*'*x*'= ∑(0,1,2,3,7,11,15)**

***F(w,x,y,z)*= *yz*+*w*'*z*= ∑(1,3,5,7,11,15)**

***F*'=z'+wy`**

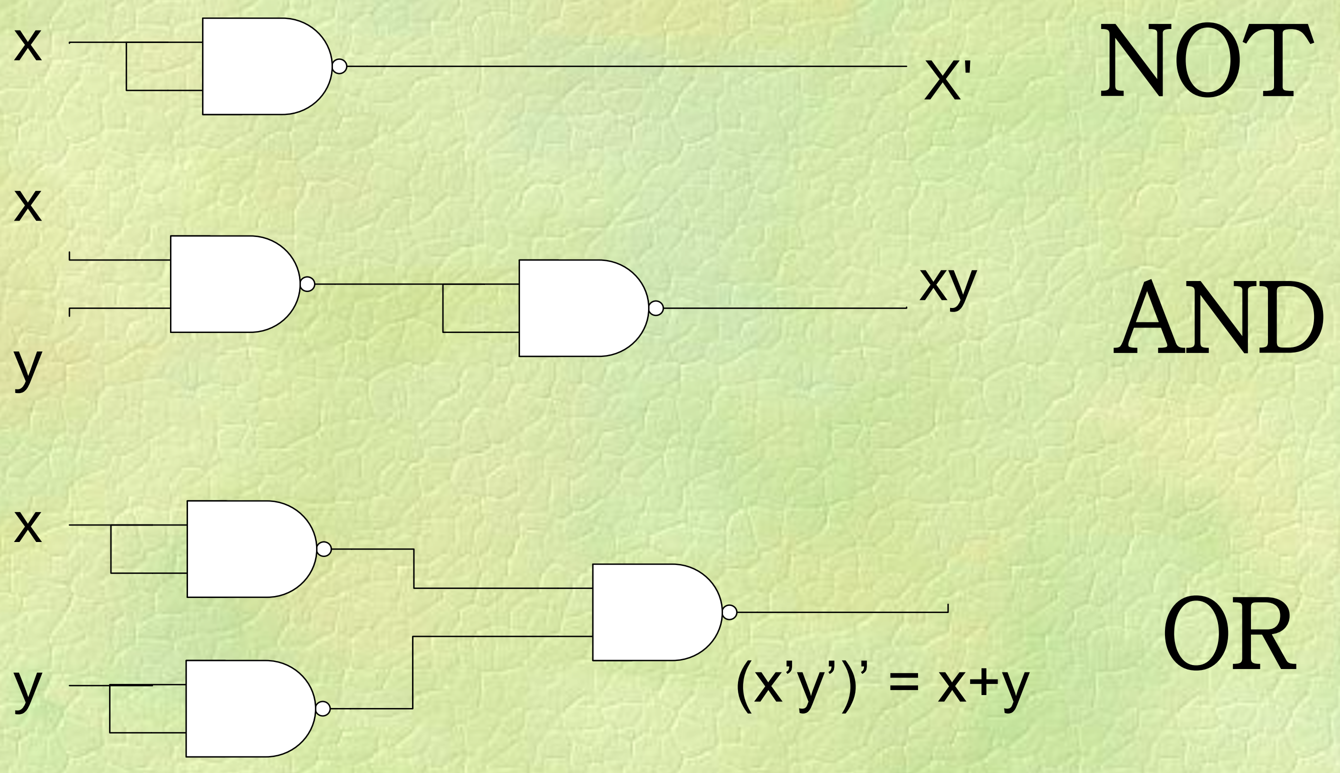
***F(w,x,y,z)* = z(w'+y)= ∑(0,2,4,6,8,9,10,12,13,14)**

Implementation of Logic Gates

- Inverter
- NOR
- NAND

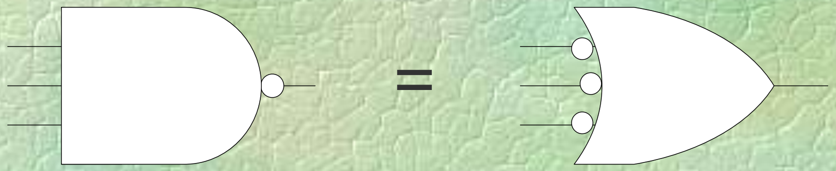
- In the market, logic gates are more commonly implemented using NAND and NOR gates rather than AND & OR
  - Because It is easier to manufactured

NOT, AND & OR Gates
implementation using NAND



NAND Gate's Symbols

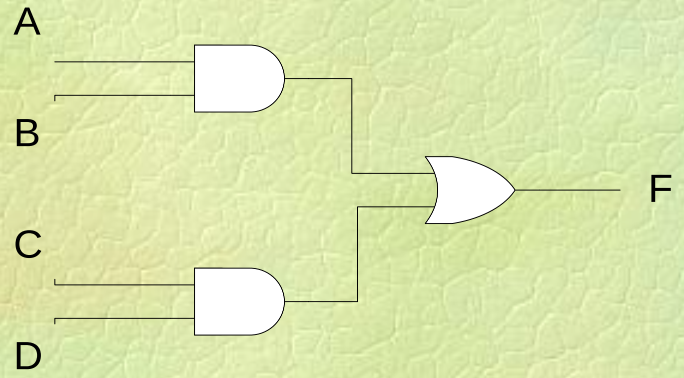
- NAND Gate as Universal Gate
  - Any gate can be represented using NAND
- Implemented as if AND-Invert or Invert-OR
  - $(xyz)' = x' + y' + z'$

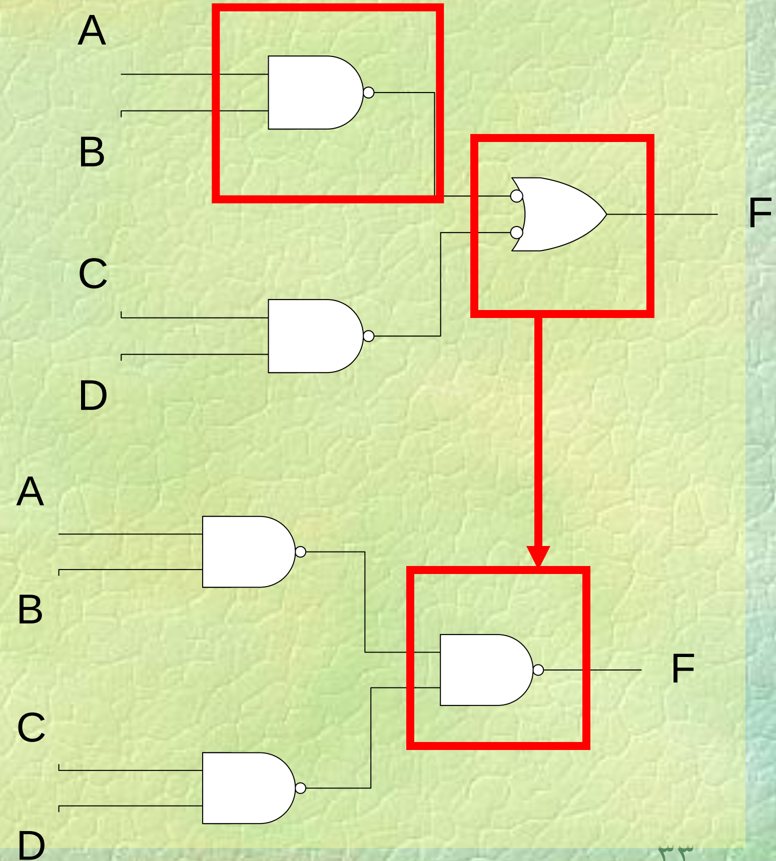


Two-Level Implementation

$F=[(AB)'' + (CD)''] = AB + CD$

- F = AB + CD



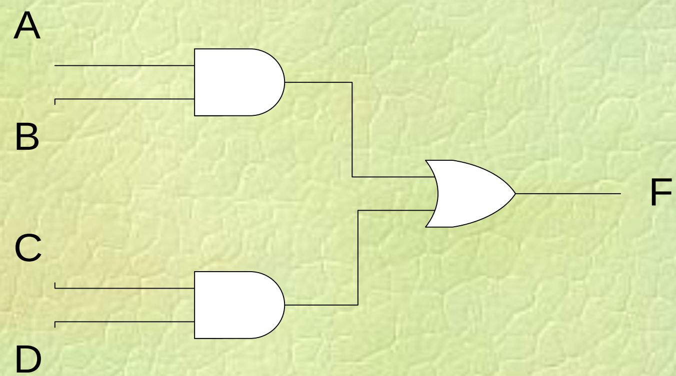


$F=[(AB)' . (CD)']' = [(A+B) . (C+D)]'$

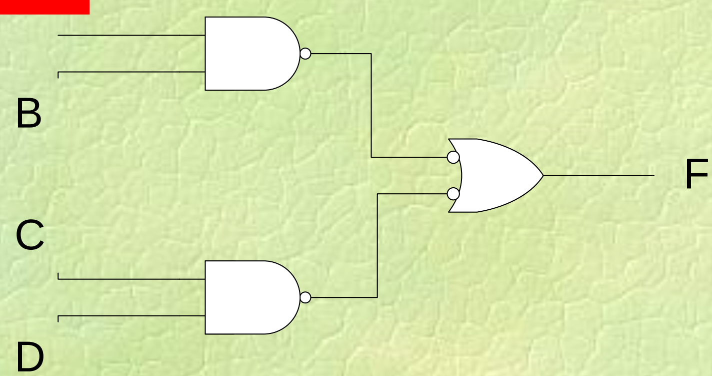
$= AB + CD$

Two-Level Implementation

- F = AB + CD

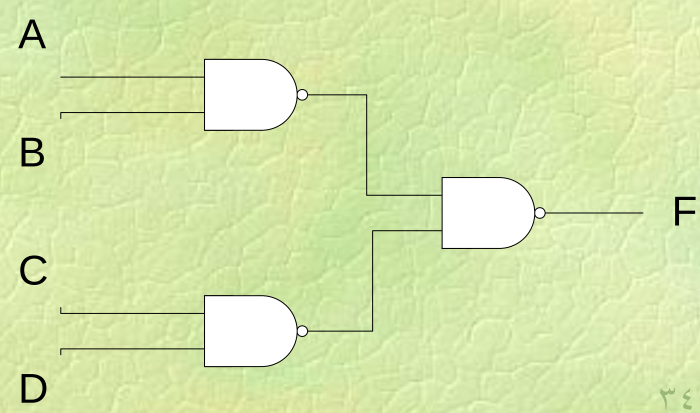


Level-1



Read the summary of procedure in Page 85 (top)

Level-2



Example

Implement the following Boolean function with NAND gates:
F(x,y,z) = (1,2,3,4,5,7)

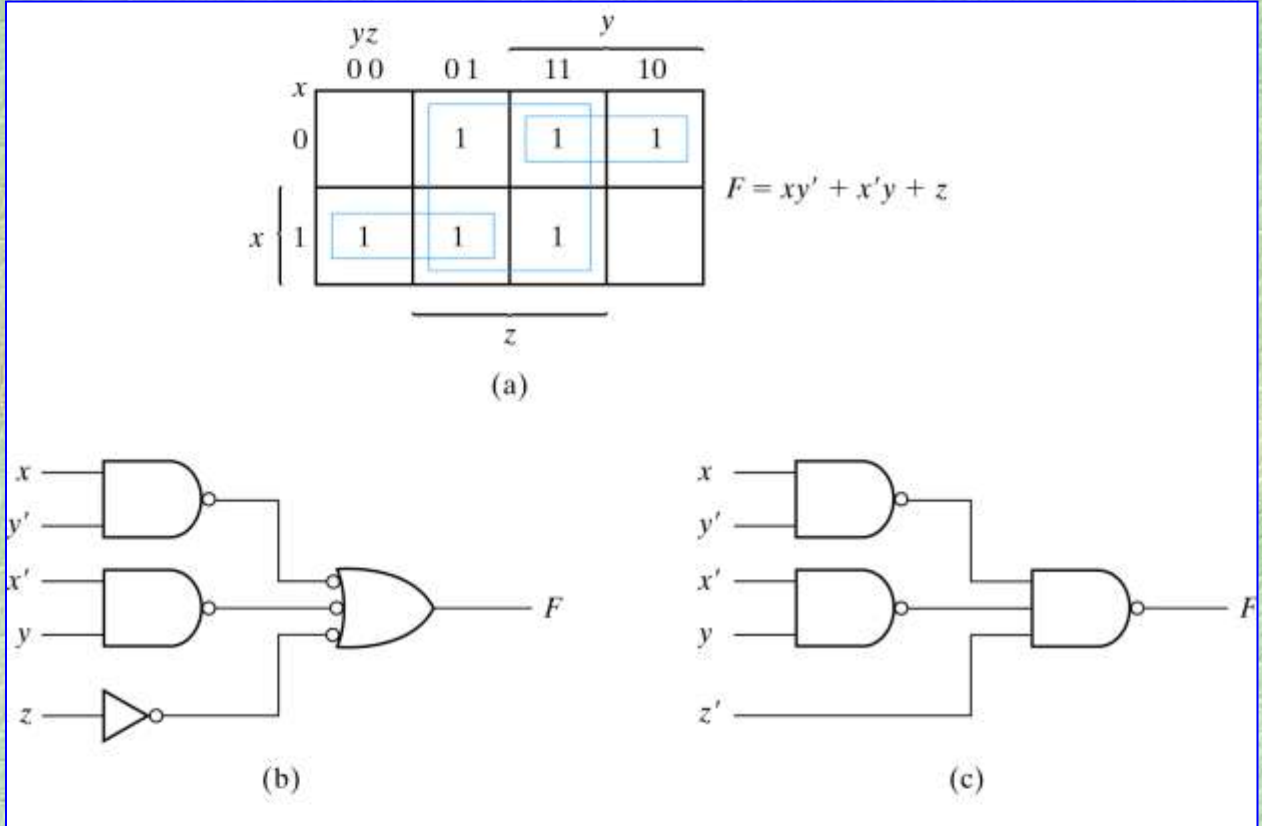


Fig. 3-21 Solution to Example 3-10

Implementation with NAND gates procedure:

1- Simplify the function and express it in sum of products.

2- Draw a NAND gate for each product term of the expression that has at least two literals.

3- Draw a single gate using the AND-invert or the invert-OR in the second level.

4- A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate

Multilevel Logic Circuit #1

- To obtain a multilevel NAND diagram from a Boolean Expression:
- Draw the Logic Diagram
- F = A (CD + B)+ BC'

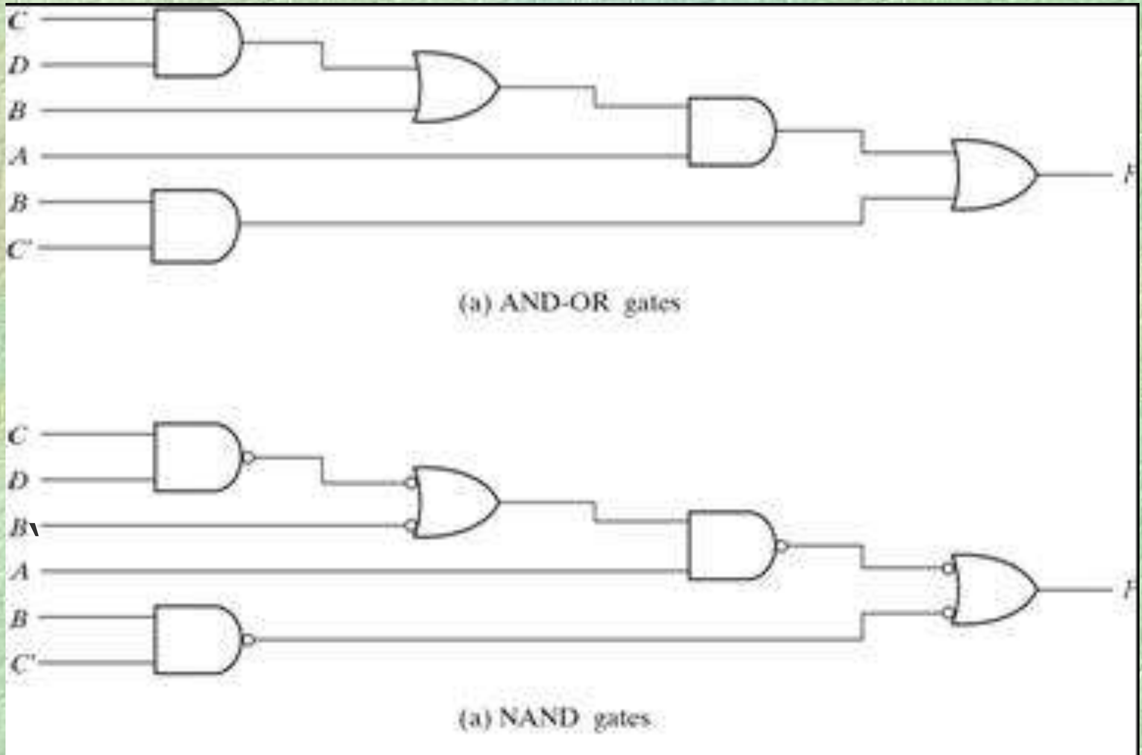


Fig. 3-22 Implementing $F = A(CD + B) + BC$

Multilevel Logic Circuit # 2

- Convert all AND gates to NAND gates with AND invert graphic symbol
- Convert all OR gates to NAND gates with Invert OR graphic symbol.
- Check all the bubbles in the diagram. For every bubble that is not compensated by an other small circle along the same line, insert an inverter (one input NAND gate) or complement the input literal.

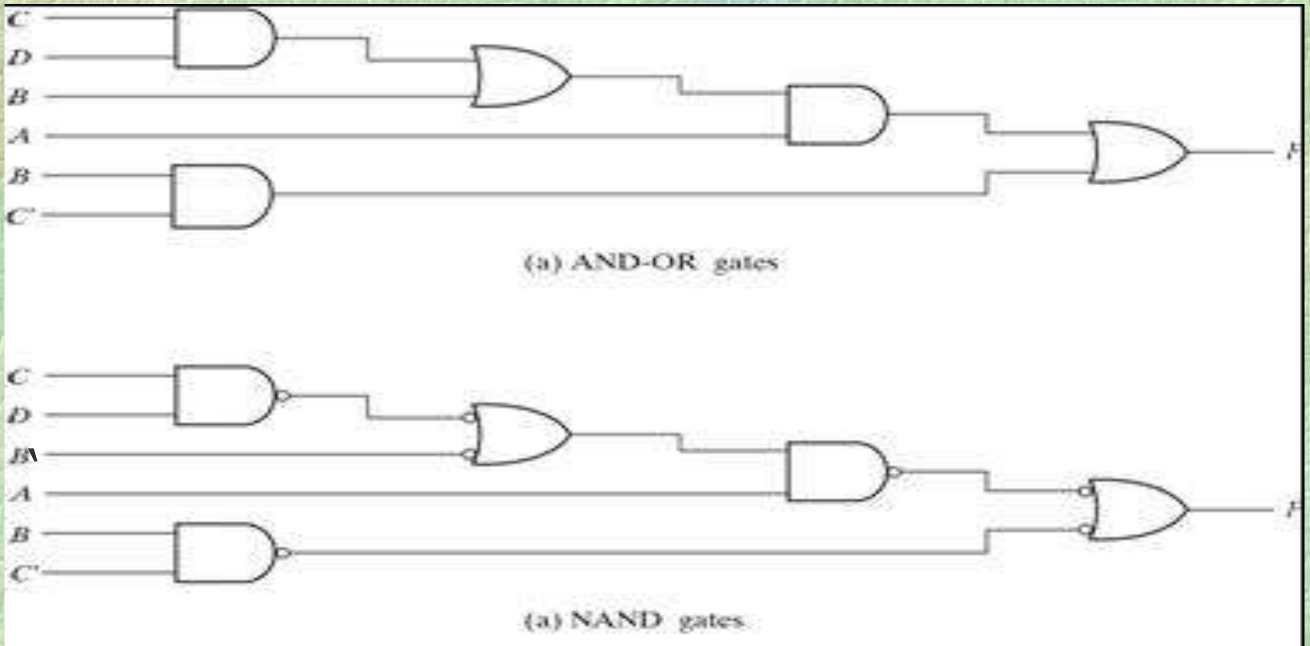


Fig. 3-22 Implementing $F = A(CD + B) + BC$

Multilevel Logic Circuit #3

- Consider the multilevel Boolean function:

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

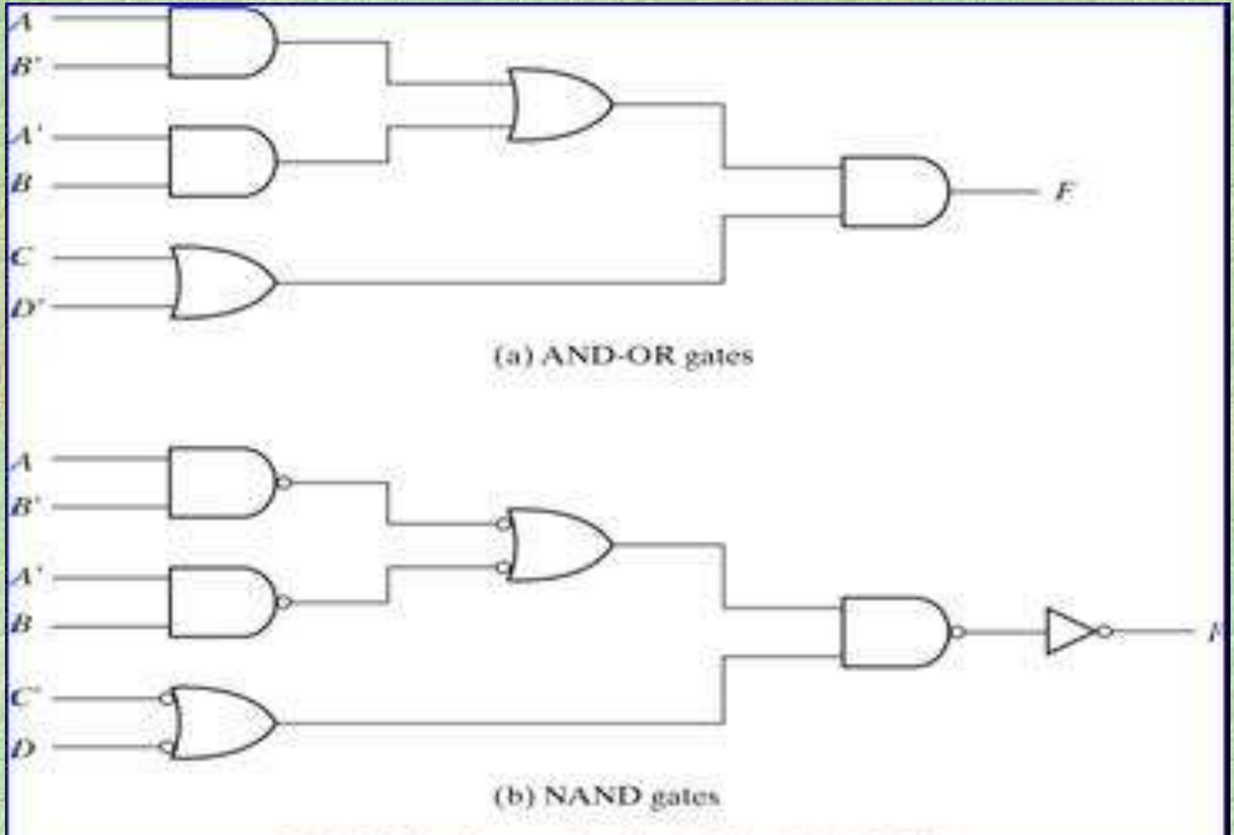


Fig. 3-23 Implementing $F = (AB' + A'B)(C + D')$

NOR Implementation

- ***Universal Gate***: The NOR gate is said to be a universal gate because any digital system can be implemented with it.

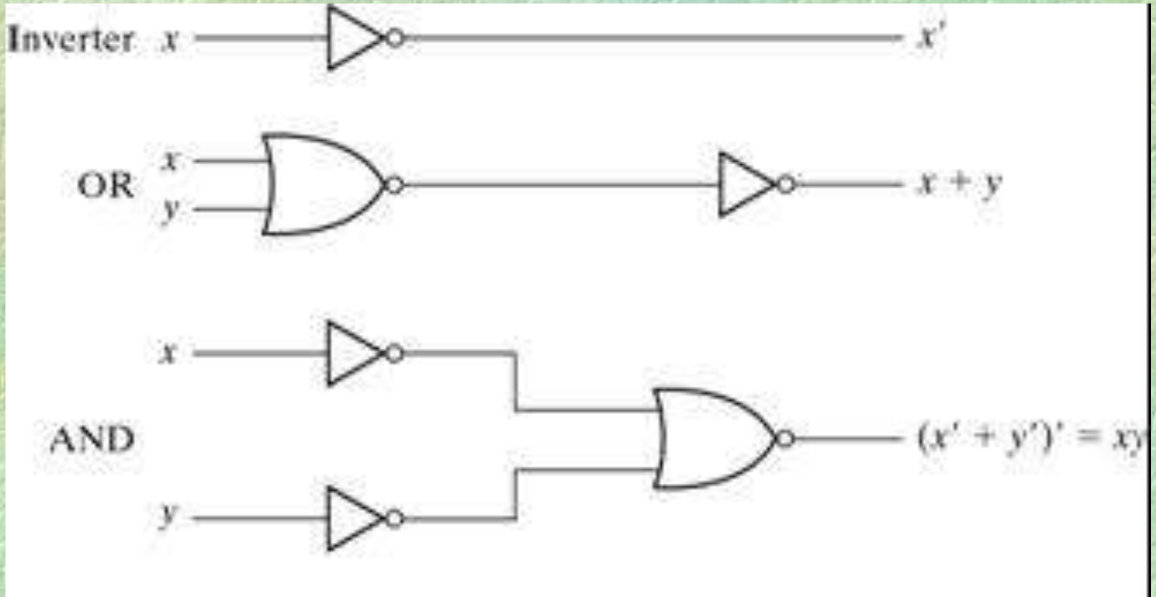


Fig. 3-24 Logic Operations with NOR Gates

NOR Gate Symbol

- Implemented as if OR-Invert or Invert-AND
  - $(x' y' z') = (x + y + z)'$

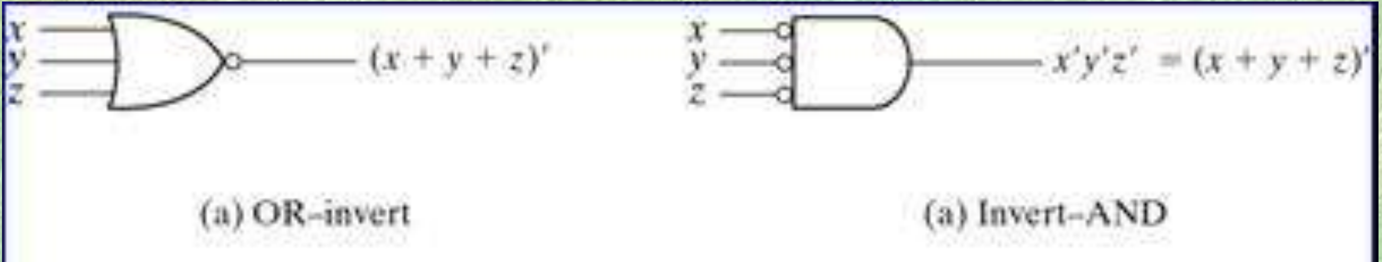


Fig. 3-25 Two Graphic Symbols for NOR Gate

Example

- **Implement the following Boolean function with NOR gates:**

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

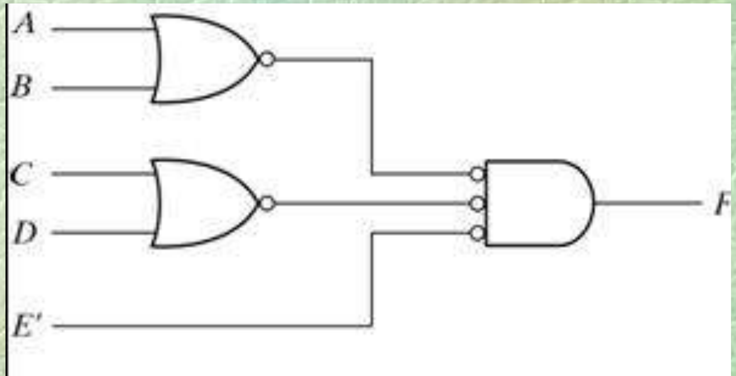


Fig. 3-26 Implementing $F = (A + B)(C + D)E$

Multilevel Logic Circuit with NOR Implementation

- Give the NOR multilevel implementation for the Boolean function:

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

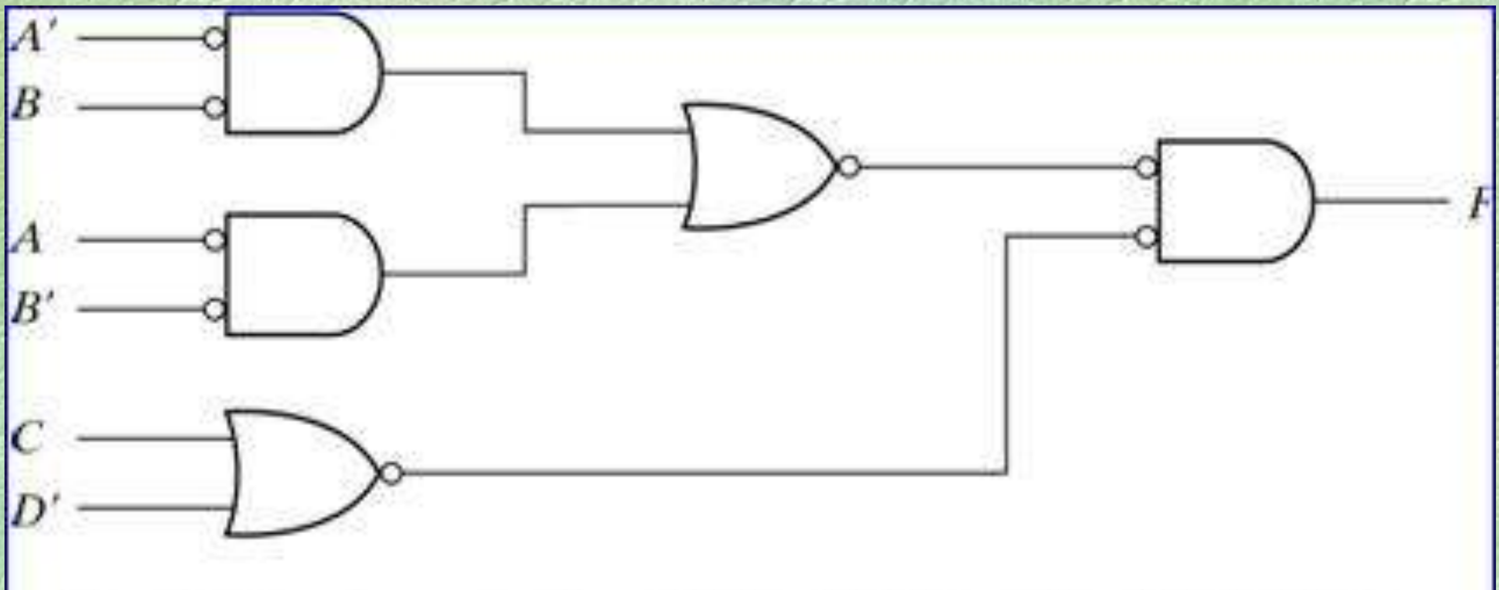


Fig. 3-27 Implementing $F = (AB' + A'B)(C + D')$ with NOR Gates

Exclusive OR Function

$$x \oplus y = xy' + x'y$$

XNOR: Inverted XOR

$$(x \oplus y)' = xy + x'y'$$

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

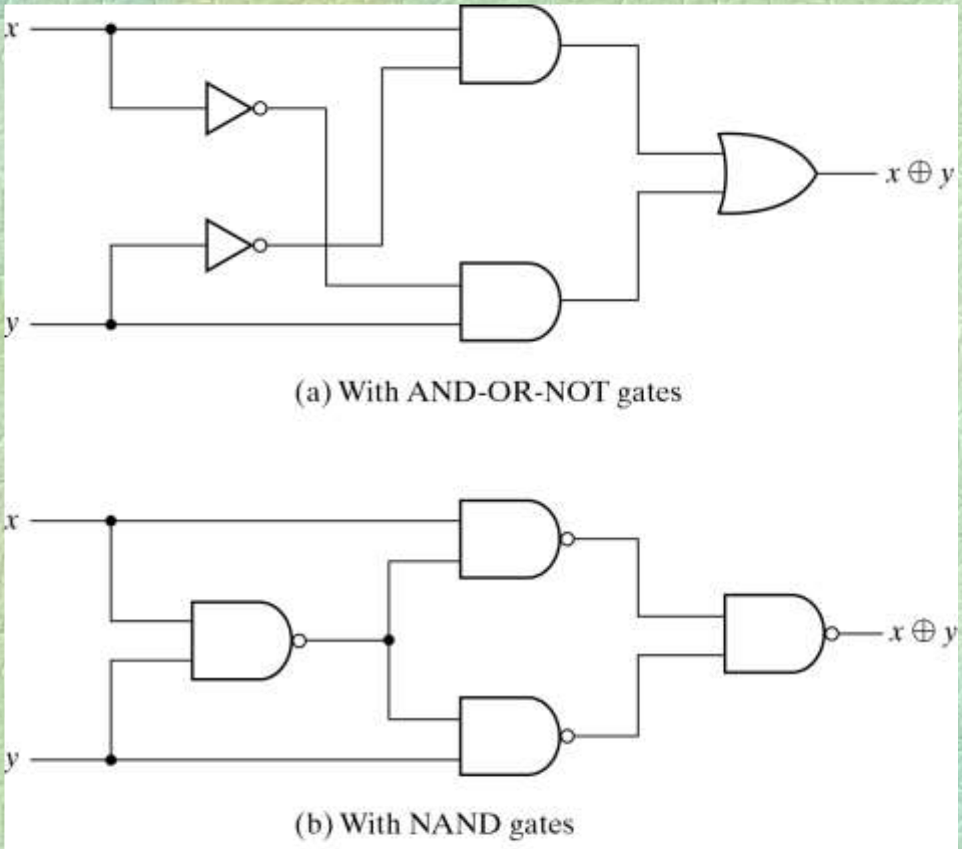


Fig. 3-32 Exclusive-OR Implementations

Exclusive OR Implementation

The first NAND gate perform the operation $(xy)' = (x'+y')$

Then

$x \oplus y = (x'+y')x+(x'+y')y$

$= xy' + x'y = x \oplus y$

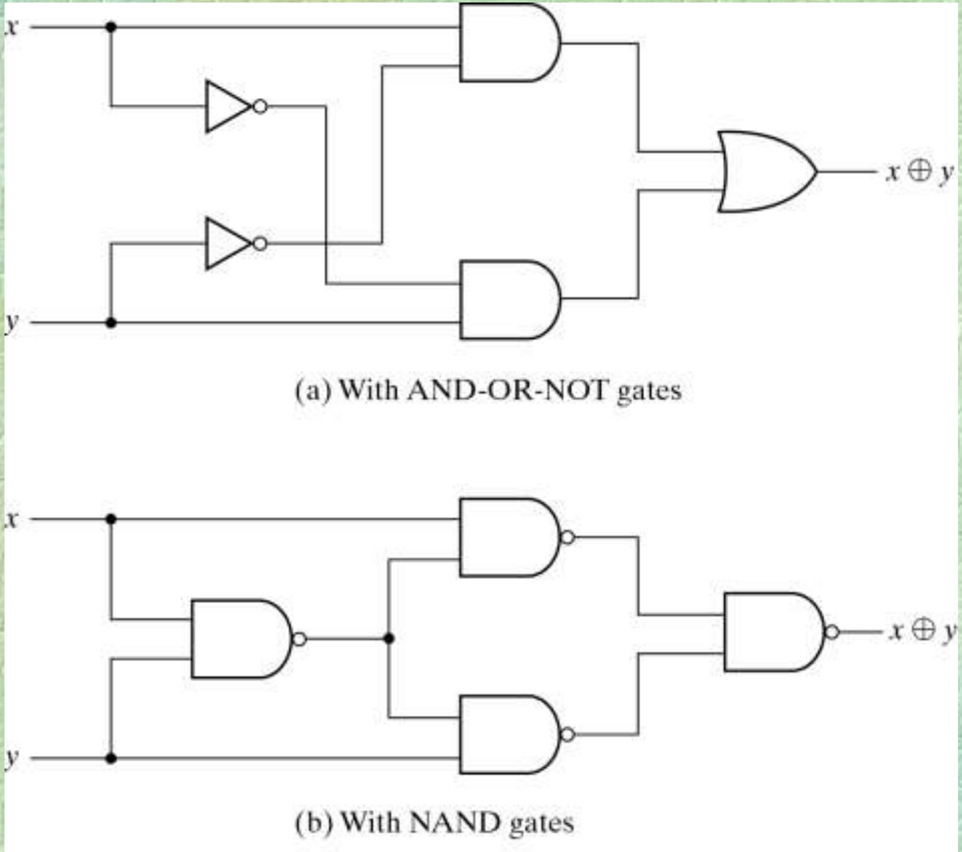


Fig. 3-32 Exclusive-OR Implementations

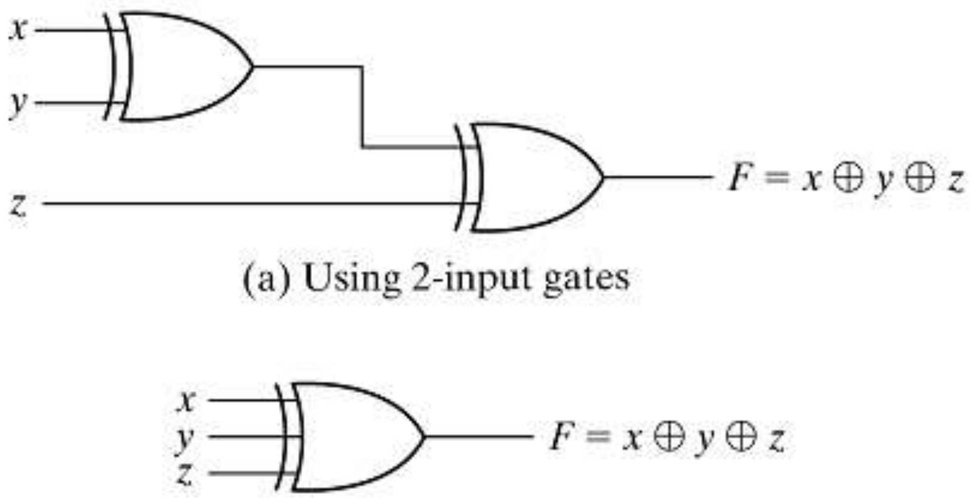
Odd Function

$A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C$

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

$= \sum(1,2,4,7)$

This means that in the 3 or more variable case the requirement of XOR function to be equal to 1 is that an odd number of variables be equal to 1



(a) Using 2-input gates

(b) 3-input gate

x	y	z	F
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

(c) Truth table

Three Variable XOR Odd and Even Function

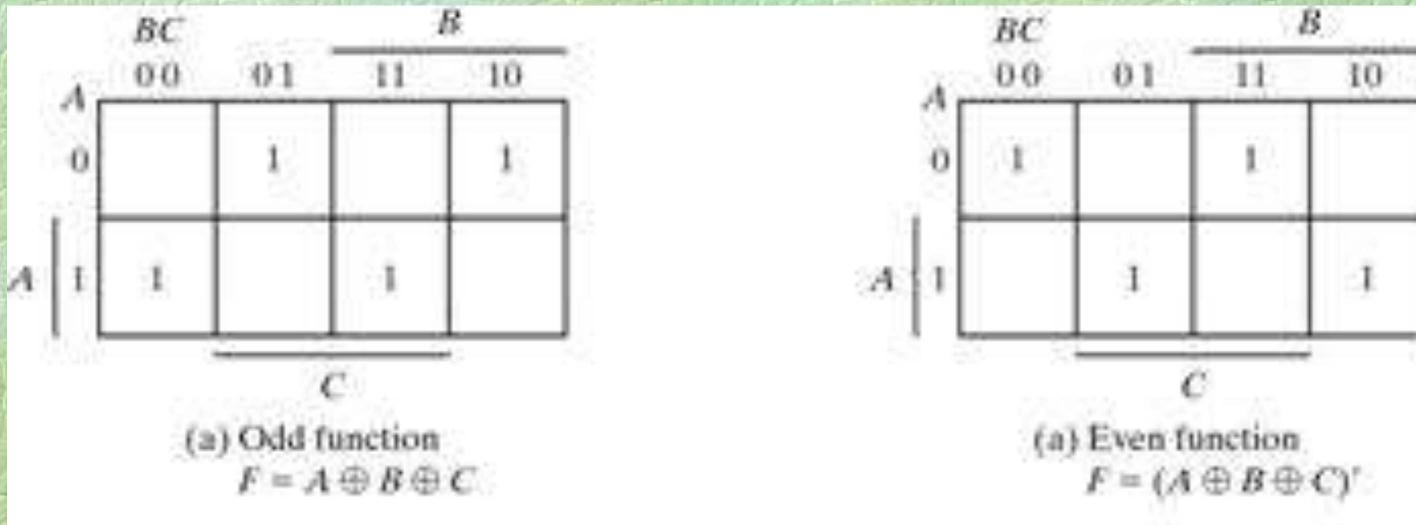


Fig. 3-33 Map for a Three-variable Exclusive-OR Function

Fig. 3-34 Logic Diagram of Odd and Even Functions

Four Variable XOR Odd and Even Functions

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

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

$$= \sum(1,2,4,7,8,11,13,14)$$

AB \ CD	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

Odd Function

$$F = A \oplus B \oplus C \oplus D$$

AB \ CD	00	01	11	10
00	1		1	
01		1		1
11	1		1	
10		1		1

Even Function

$$F = (A \oplus B \oplus C \oplus D)'$$

Parity Generation and Checking

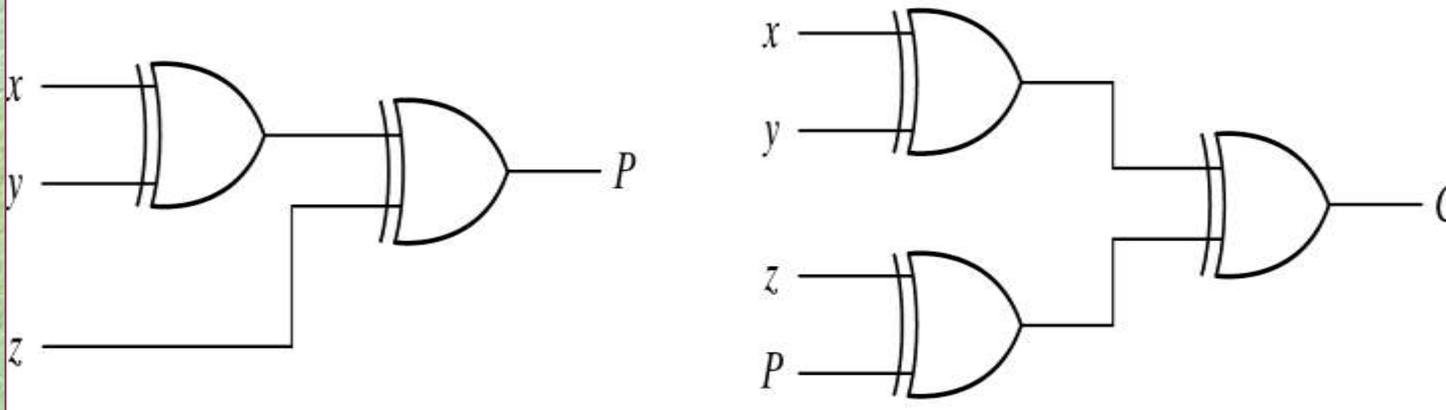
Three-Bit Message			*Parity Bit*
x	*y*	*Z*	*P*
0	*0*	*0*	0
0	*0*	*1*	1
0	*1*	*0*	1
0	*1*	*1*	0
1	*0*	*0*	1
1	*0*	*1*	0
1	*1*	*0*	0
1	*1*	*1*	1

Even Parity Generator 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

Even Parity checker Truth Table

Logic Diagram of Parity Generator and Checker



(a) 3-bit even parity generator

(a) 4-bit even parity checker

Fig. 3-36 Logic Diagram of a Parity Generator and Checker