Chapter 3

Gate –Level Minimization

The Karnaugh MAP

- An alternate approach to representing Boolean functions
- used to minimize Boolean functions
- Easy conversion from truth table to K-map
- Easy to obtain minimized SOP function.
- Simple steps used to perform minimization

→ **Much faster and more efficient than previous minimization techniques with Boolean algebra.**

The Karnaugh MAP

- K-MAP is ideally suited for four or less variables, becoming cumbersome for five or more variables.

→Each square represents a Minterm

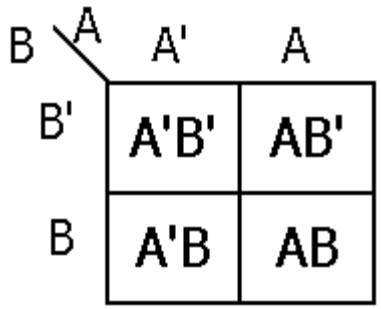
→*Map is arranged such that two neighbors differ in only one variable (e.g. ABC +ABC')*

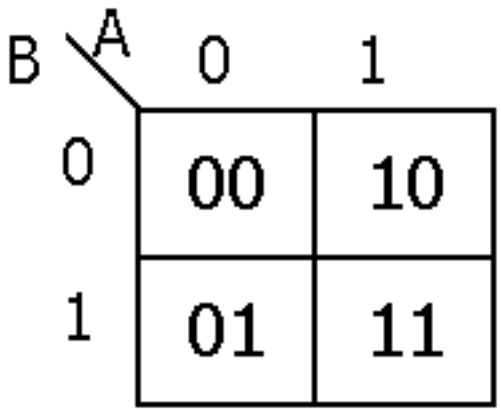
→ *Two terms must be "adjacent" in the map*

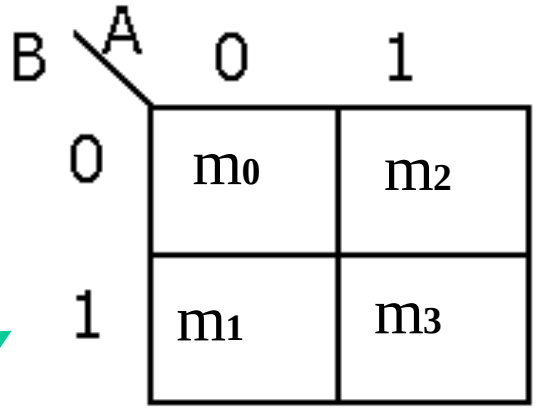
→ A K-map of n variables will have 2^n squares

→ For a Boolean expression, product terms are denoted by 1's, while sum terms are denoted by 0's – or left blank (represented by Minterms in the map)

K-Map with Two variables





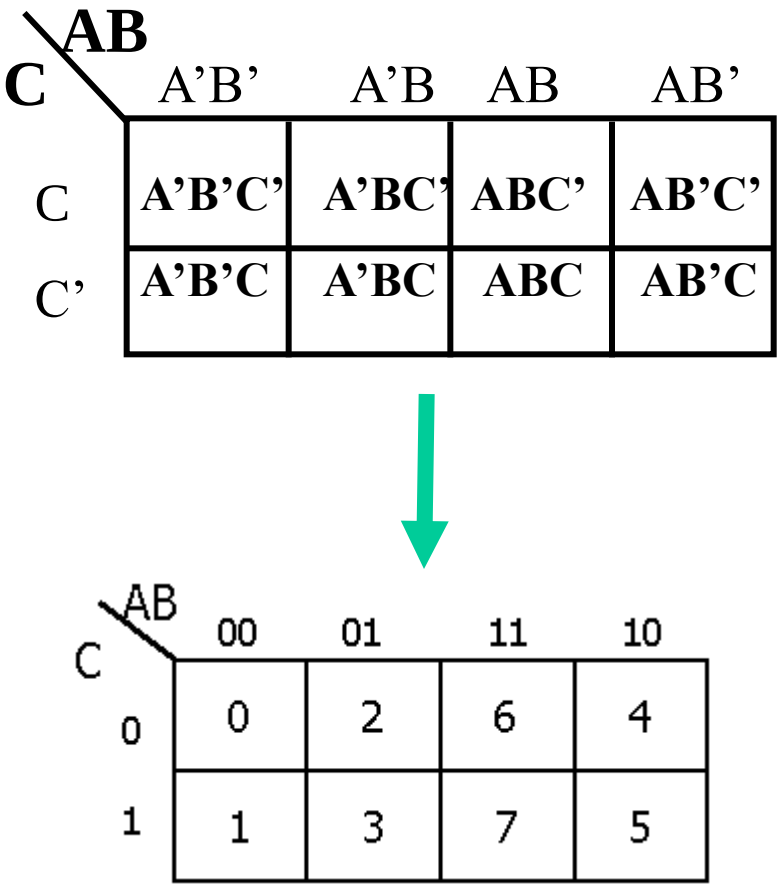


$m_0 = A'B'$

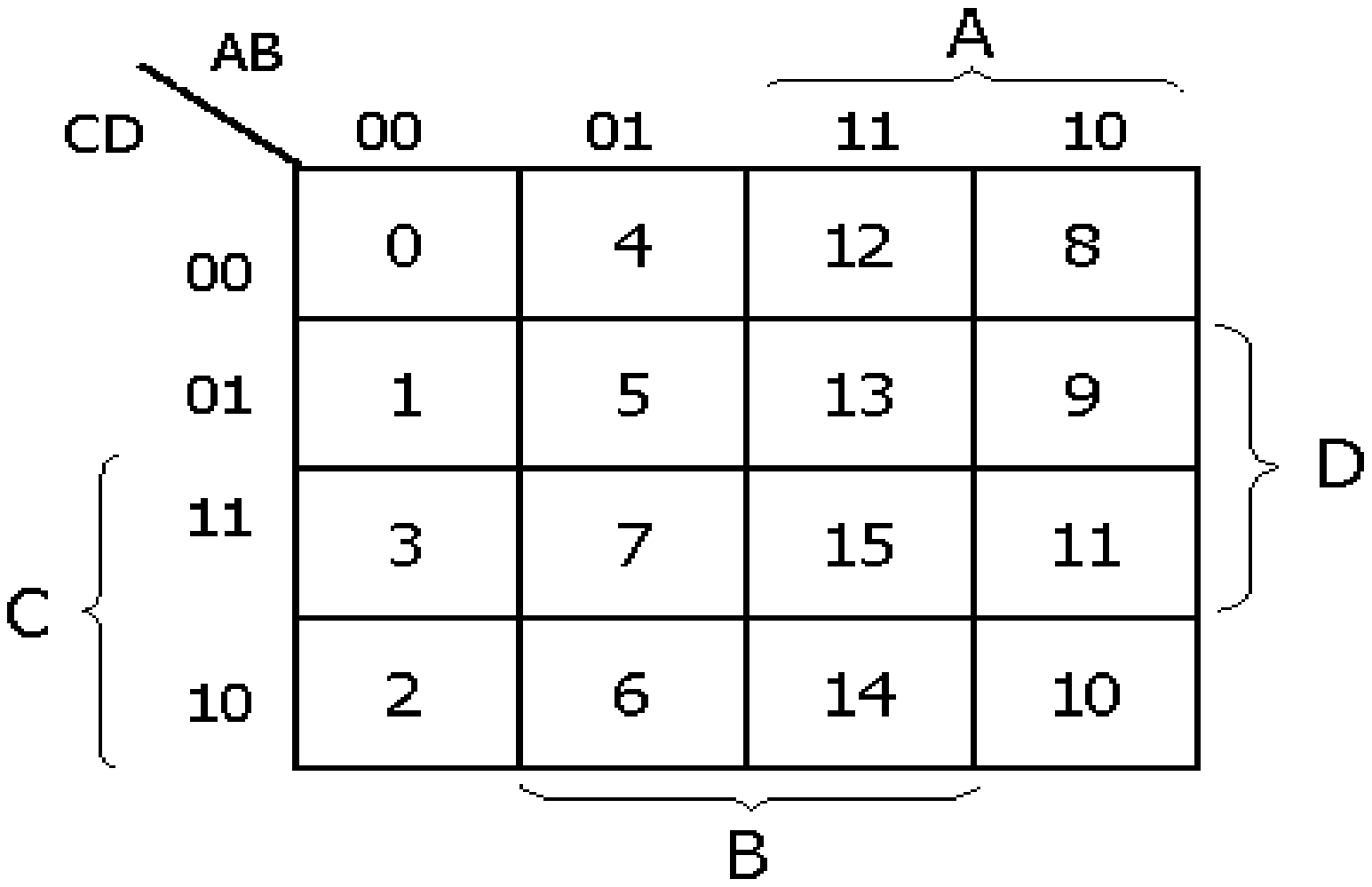
$m_1 = A'B$

$A' (B+B')$

K-Map with 3 variables



Kmap With 4 variables



Assigning 1's and 0's in Kmap

•Assign the value of the outputs to the corresponding Minterms in the K-map

F (A,B,C,D)= A'B'C'D'+A'BC'D'+ AB'C'D' + A'BC'D+ ABC'D + ABCD'+AB'CD'

CD \ AB	00	01	11	10
00	1	1	0	1
01	0	1	1	0
11	0	0	0	0
10	0	0	1	1

→Consider the squares with 1's to simplify SOP

→ Consider the squares with 0's to simplify POS

Karnaugh Maps - grouping squares

- **Groups of squares are formed in considering the following rules:**

-Every square containing 1 must be considered at least once

- A square containing 1 can be included in as many groups as desired

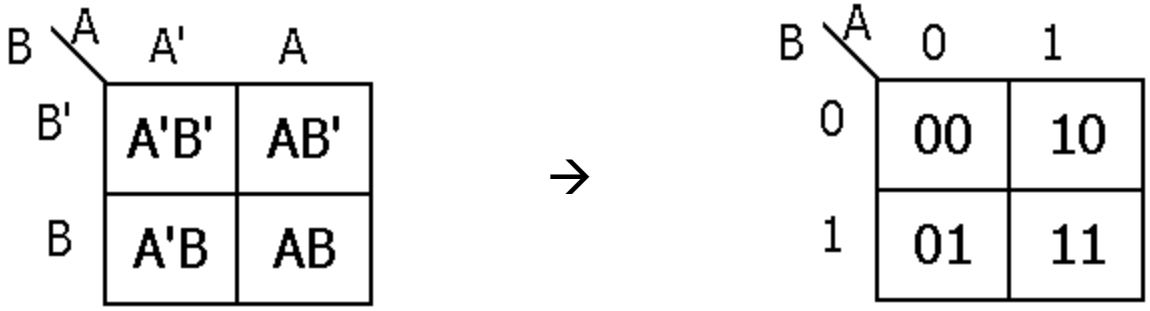
- A group must be as large as possible (i.e. large number of squares)

- The number of squares in a group must be equal to 2^n, i.e. 2,4,8,.

→ the simplified logic expression obtained from a K-map is not always unique. Groupings can be made in different ways.

2 variable Karnaugh Map

$x + x' = 1$



F=AB’+AB

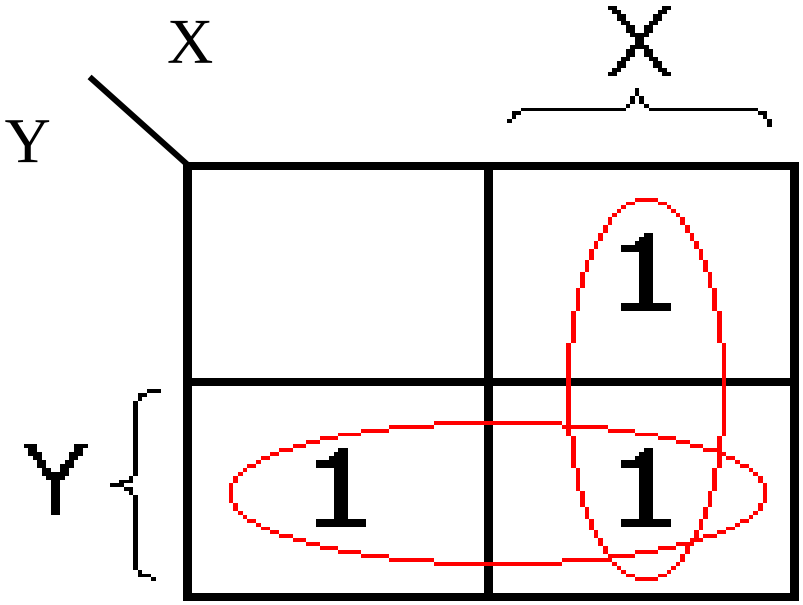
F=A’B’+A’B

F = A

F = A’

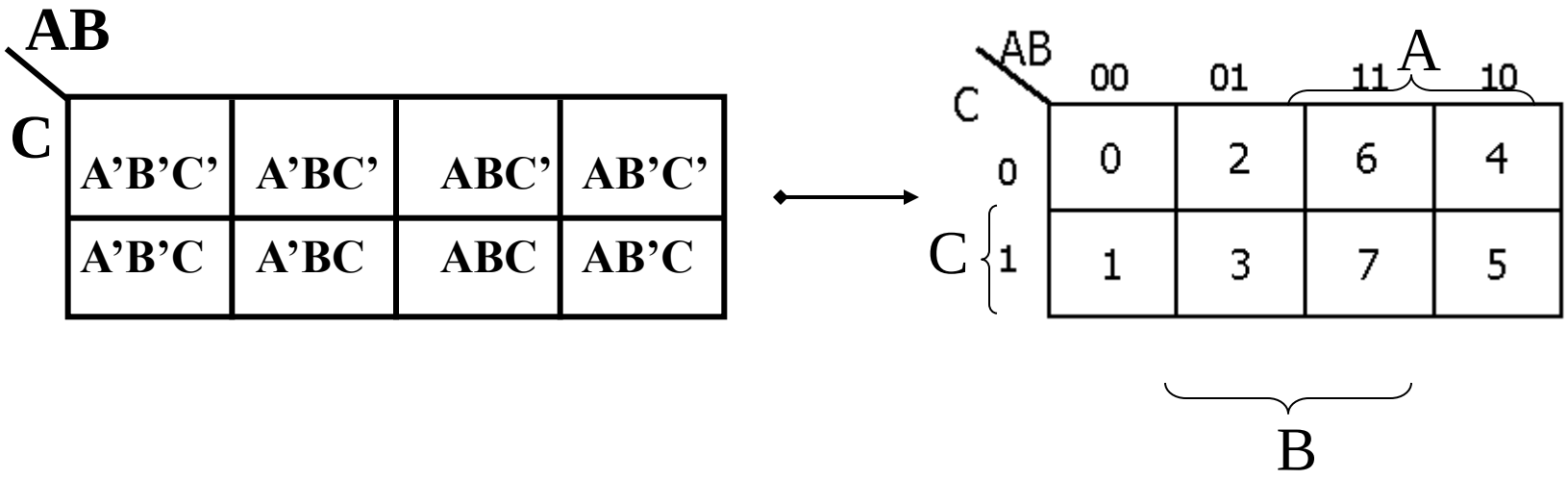
2 variable Karnaugh Map

F =X'Y+XY+XY'

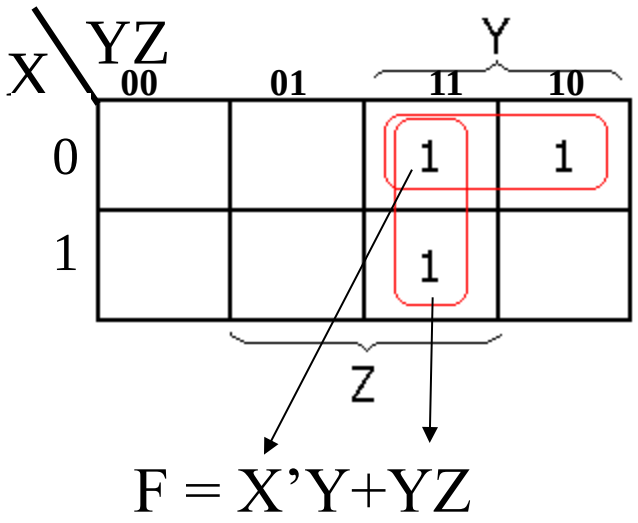


F = X + Y

3 Variable Karnaugh Map

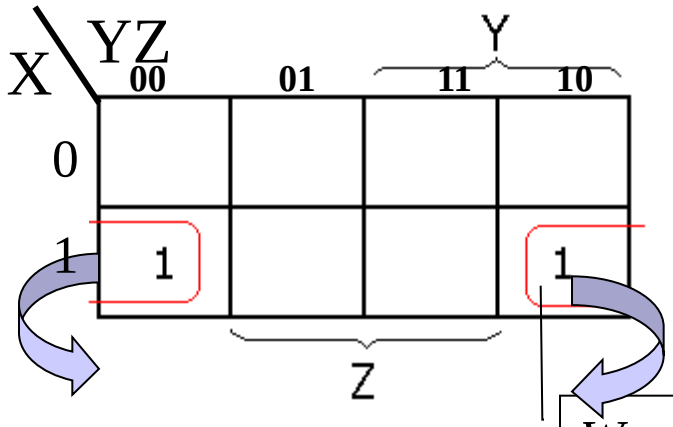


F = X'YZ'+XYZ+X'YZ



F = X'Y+YZ

F = XY'Z'+XYZ'

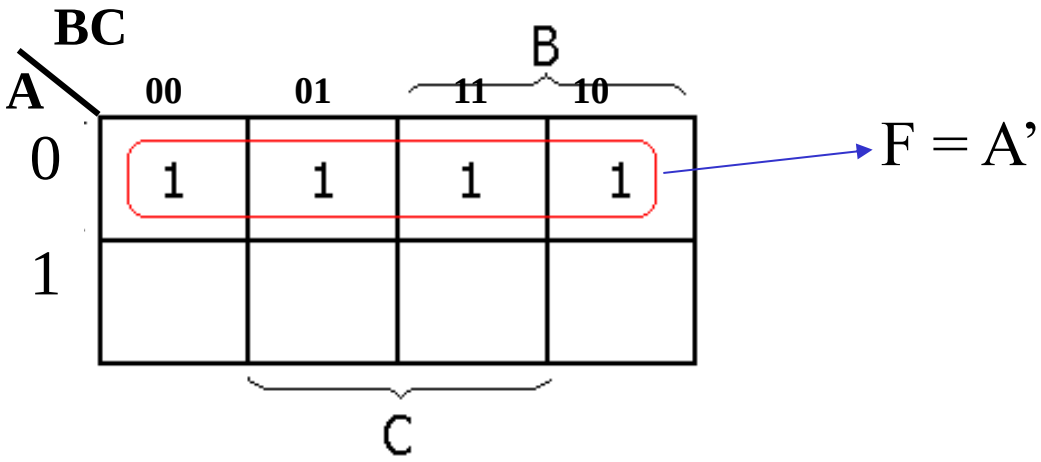


F = XZ'

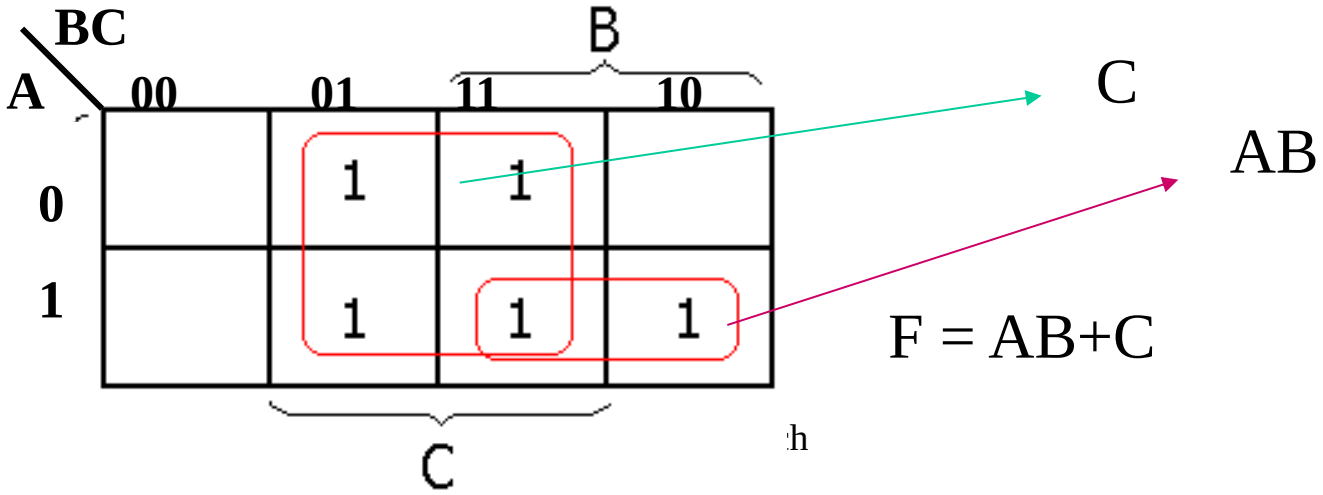
Wrapping around edges

3 variable Karnaugh Map

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

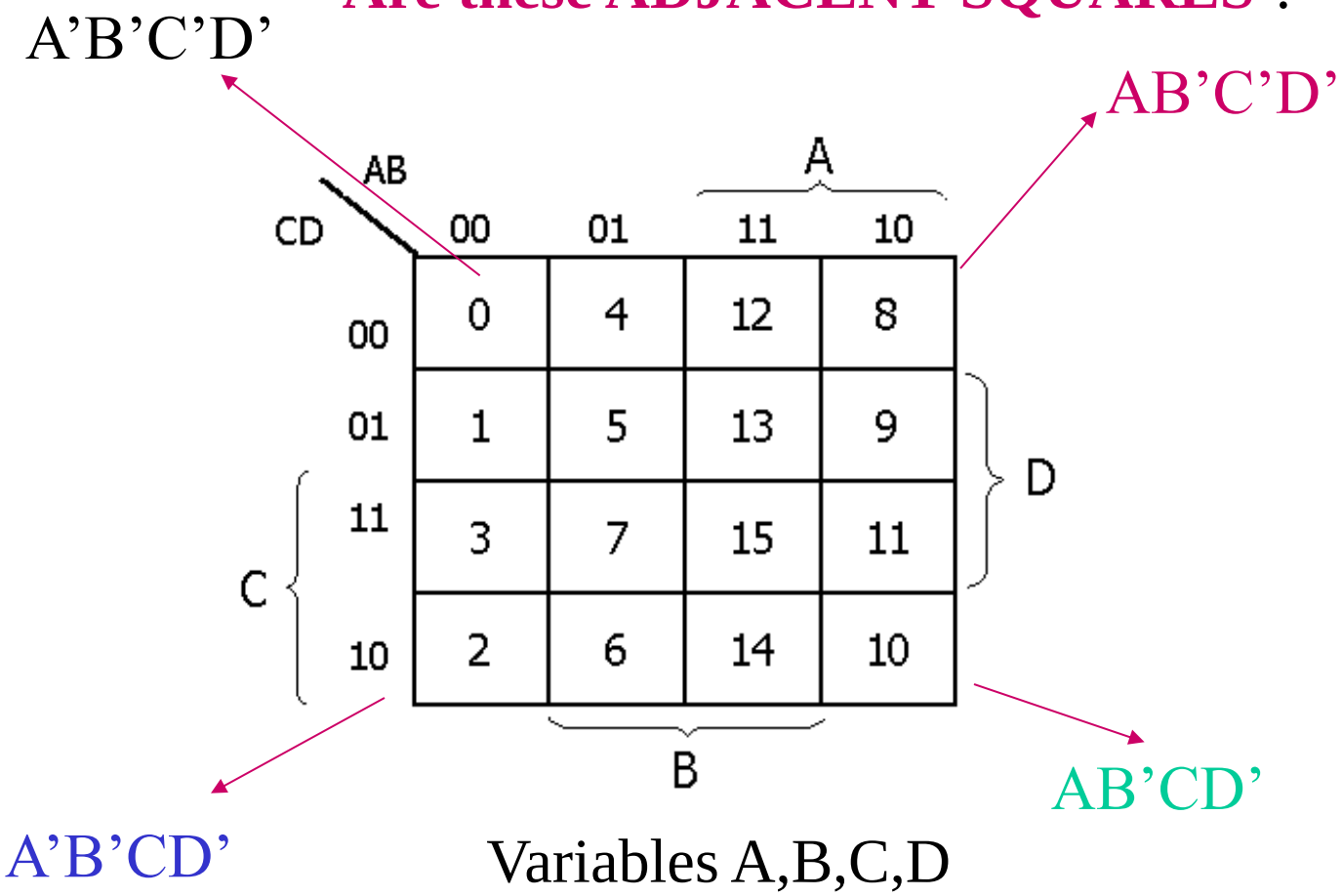


F(A,B,C) = A'BC+A'B'C+AB'C+ABC+ABC'



4 variables K-MAP

Are these ADJACENT SQUARES ?



Variables A,B,C,D

Function with "don't care" Outputs

- Example

A purely binary number is converted into a 5-4-2-1 BCD number recall that BCD is often used to represent numbers in computers. The truth table is as below.

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
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				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Function with "don't care" Outputs

- Example

A purely binary number is converted into a 5-4-2-1 BCD number recall that BCD is often used to represent numbers in computers. The truth table is as below.

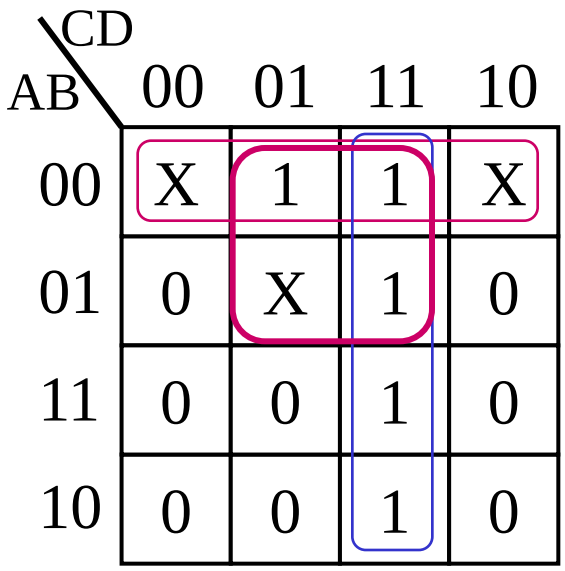
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Don't care terms

Σd(10,11,12,13,14,15) are don't care outputs for W, X, Y,Z

K-map with Don't Care outputs

- Don't care outputs can be used as 0 or 1.
- This can be used to help simplify logic functions.
- Example: F(A,B,C,D)= Σ m(1,3,7,11,15 with Σ d(0,2,5)



•X denotes a "don't care" term.

• X are used as 1's or 0's to increase the number of squares during the grouping

F = CD + A'B' or F= CD + A'D

Solution to the 5-4-2-1 BCD example

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
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				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Don't care terms

Using K-maps for the 4 variable we obtain:

W=A+BD+BC

X=BC'D'+AD

Y=CD+B'C+AD'

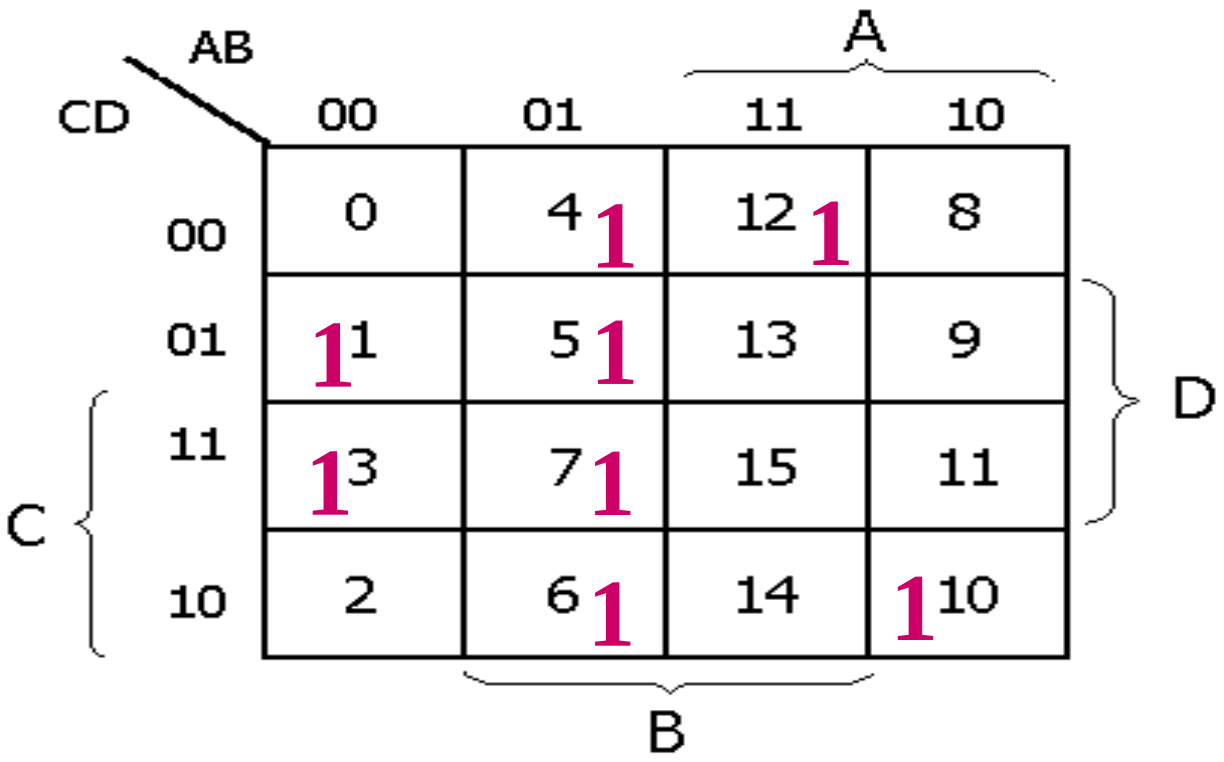
Z=AD'+A'B'D+BCD'

K-Maps- Examples

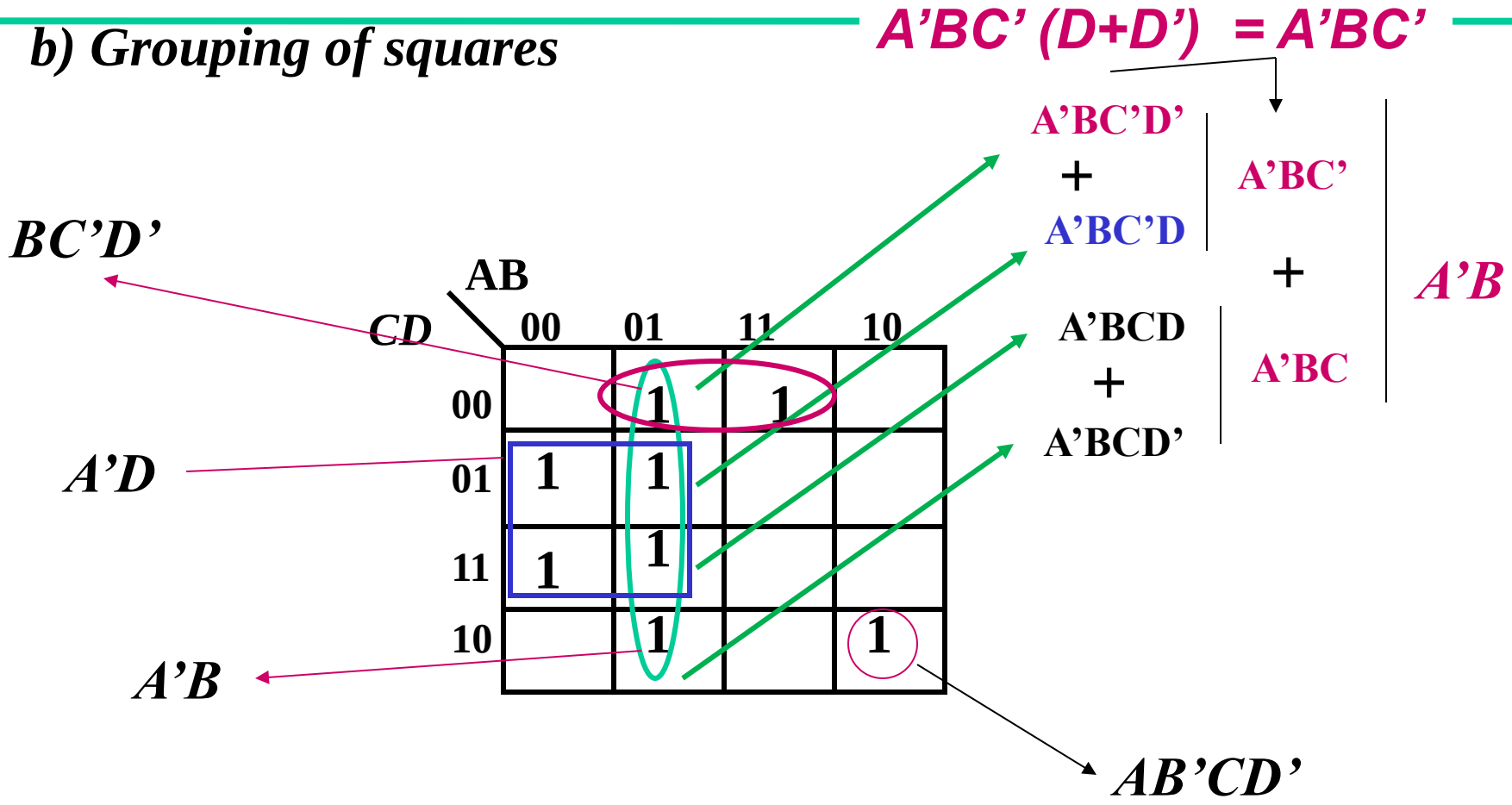
1- simplify the following expression using K-Maps

$$F(A,B,C,D) = \sum m\,(1, 3, 4, 5, 6, 7, 10, 12)$$

a) Building K-Map for F



b) Grouping of squares



c) Write the Simplified Expression

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

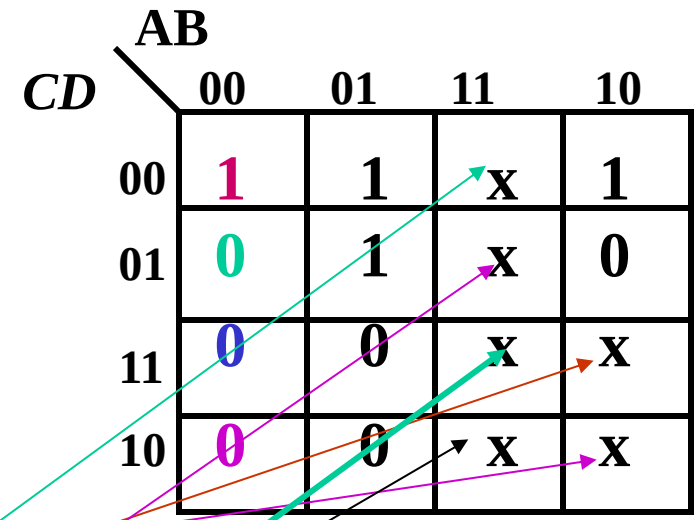
A	*B*	*C*	*D*	*F*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

a) Building K-map from the truth table

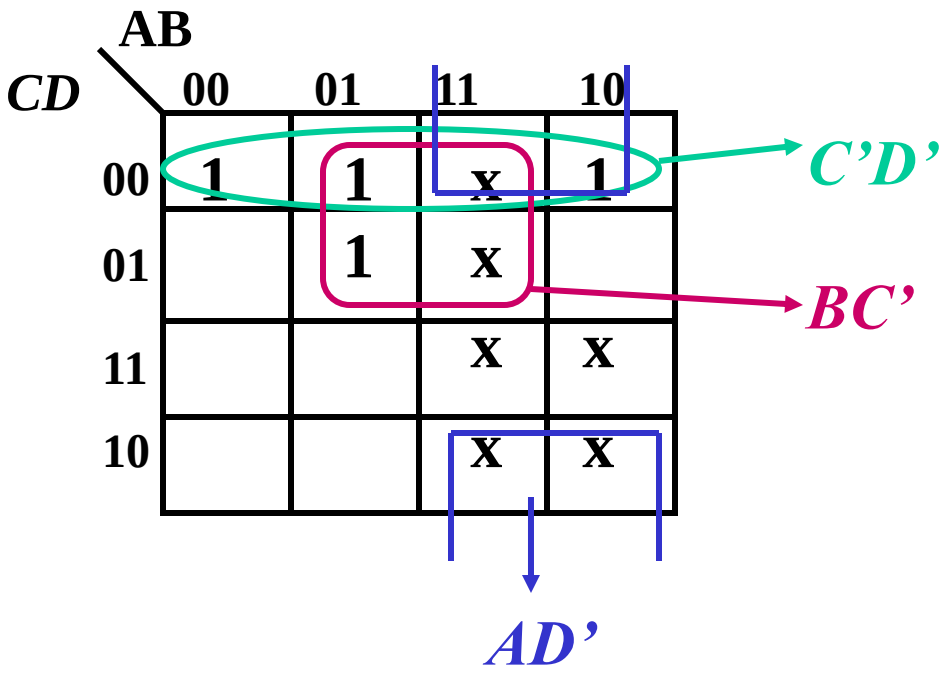
CD \ AB	00	01	11	10
00	1	1	x	1
01	0	1	x	0
11	0	0	x	x
10	0	0	x	x

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

a) Building K-map from the truth table



b) Obtain Sum of Products for F



$$F = BC' + C'D'$$

Prime implicants

When grouping square:

• a group should contain a maximum of adjacent squares

→Known as *PRIME IMPLICANT*

• each group represents one term in the function

• a function should contain a minimum set of terms

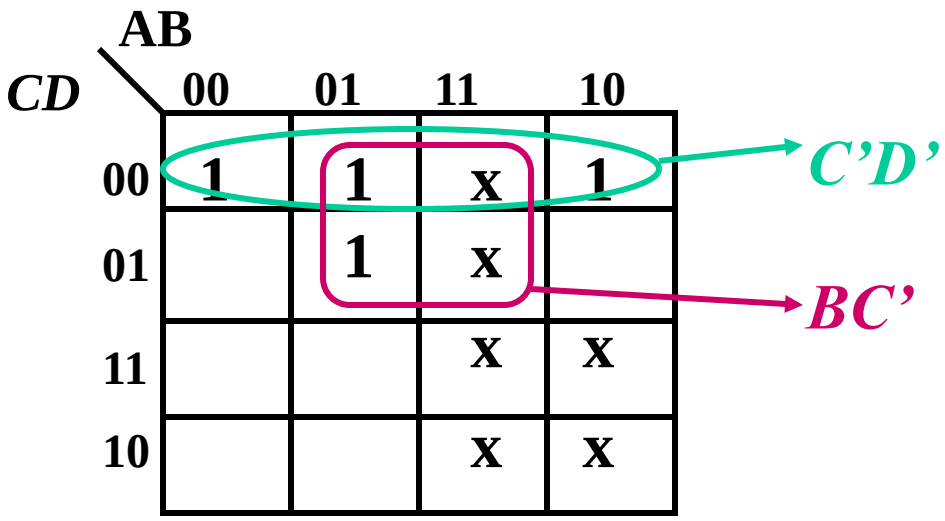
when selecting groups :

- Select only those groups that have at least one square that is not covered by another group:

→ Known as *Essential Prime Implicant*

- Do not select a group that has all its squares covered by other groups: →Known as *Optional prime implicant*

b) Obtain Sum of Products for F



$$F = BC' + C'D'$$

c) Obtain product of Sums : 2 STEPS

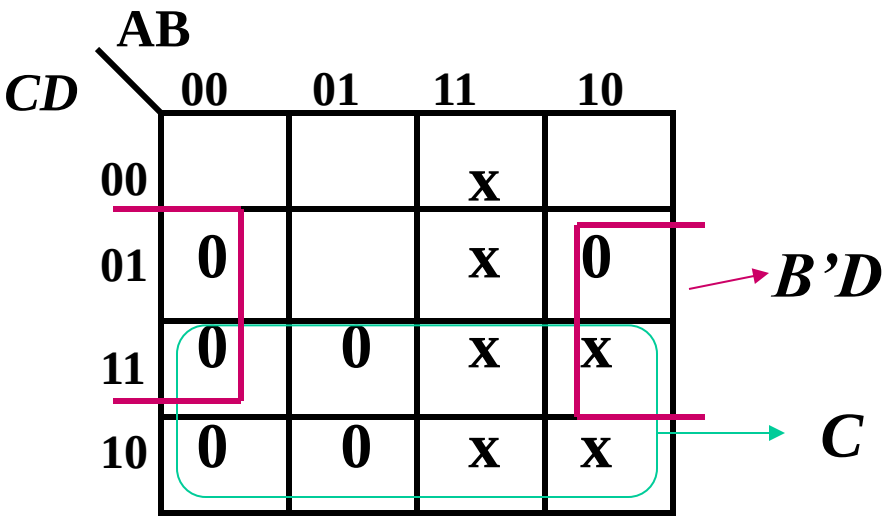
1 - use Minterms to simplify and obtain F'

$$F' = B'D + C$$

2 - complement F' to get the Product of Sum form

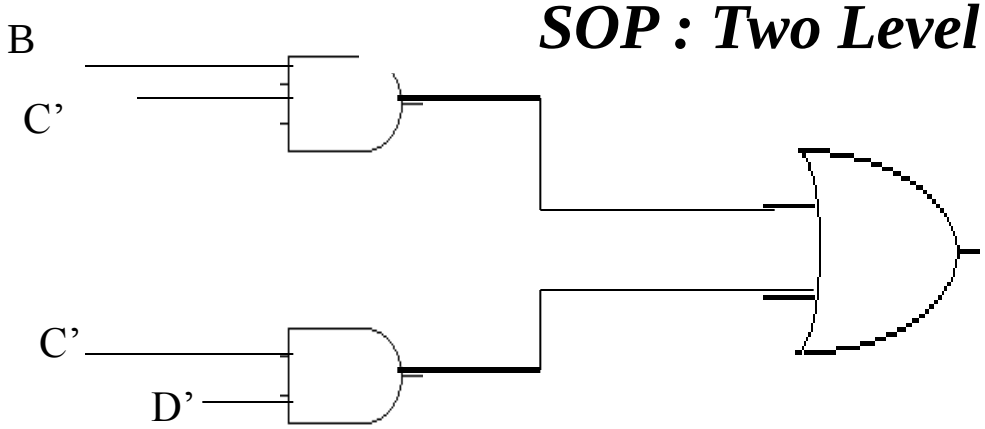
$$F'' = (B'D + C)' = (B''+D').C'$$

$$F = (B +D').C'$$



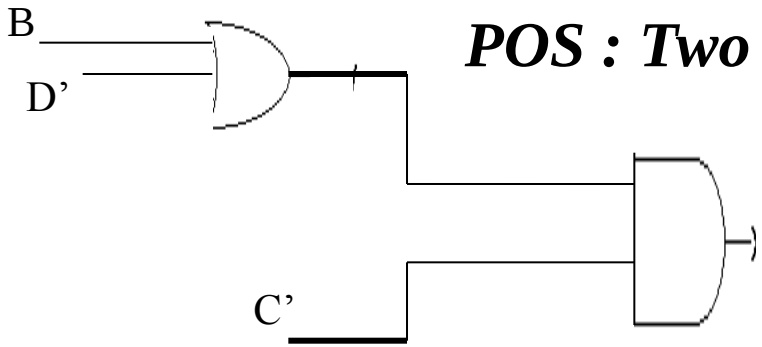
Two Level Implementations

SOP : Two Level Implementation



$$F = BC' + C'D'$$

POS : Two Level Implementation



$$F = (B + D').C'$$

Implementations using NAND & NOR Gates

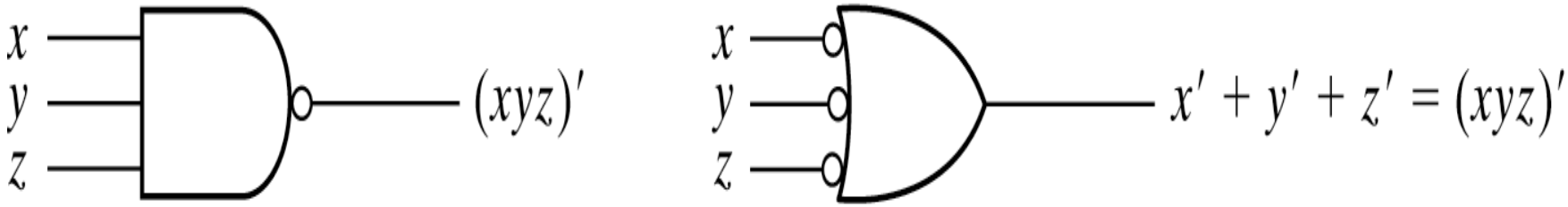
• Digital circuits are frequently constructed with NAND or NOR gates rather with AND and OR gates.

→ Both NAND and NOR gates are very valuable as any design can be realized using either one.

•It is easier to build digital circuits using all NAND or NOR gates than to combine AND,OR, and NOT gates.

•NAND/NOR gates are typically faster and cheaper to produce.

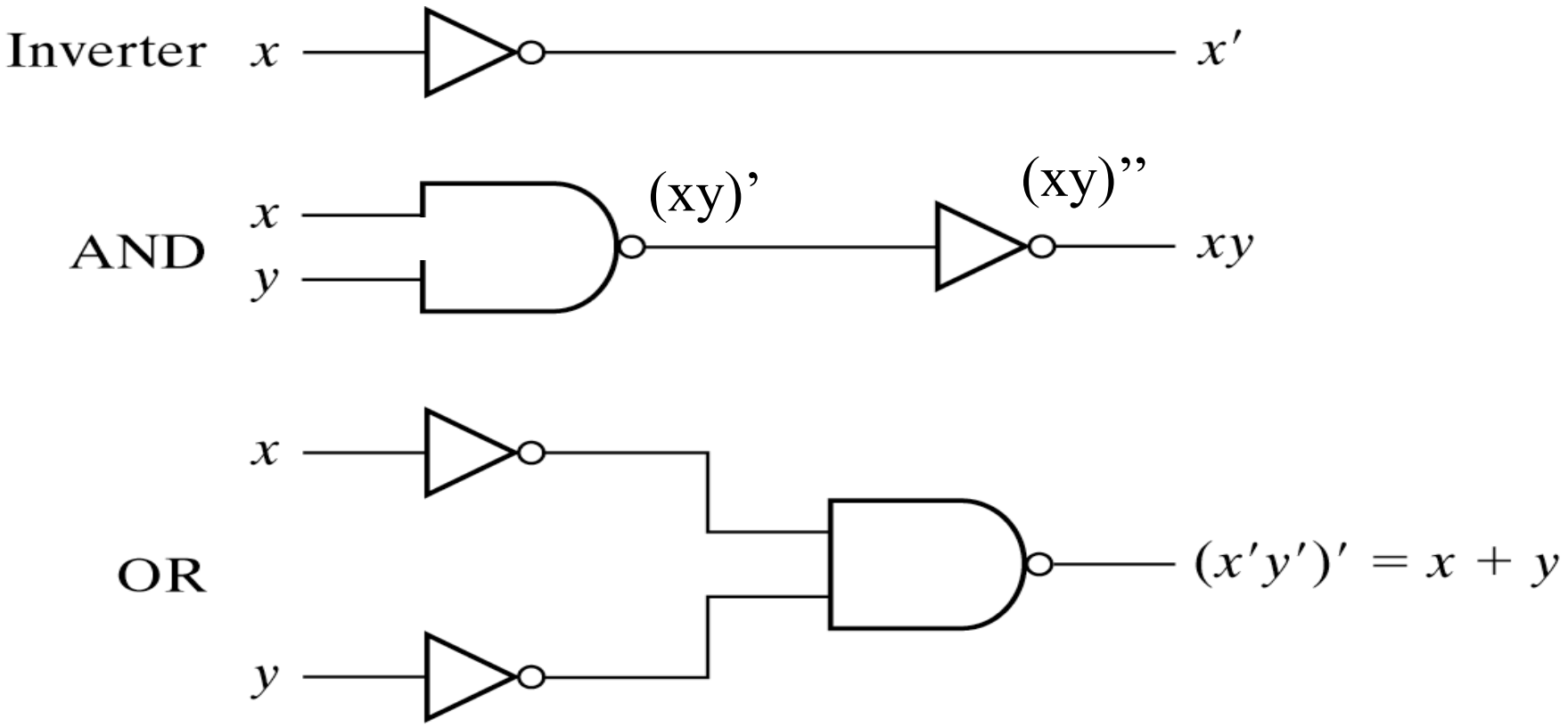
Logic Operations with NAND Gates



(a) AND–invert (b) Invert–OR

Two Graphic Symbols for NAND Gate

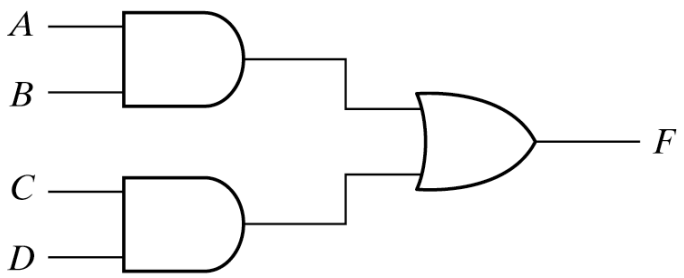
Logic Operations with NAND Gates



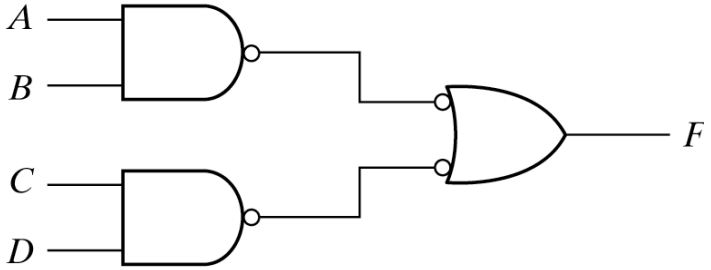
Logic Operations with NAND Gates

NAND gates Implementations

(AB)'' + (CD)''= ((AB)'(CD)')'

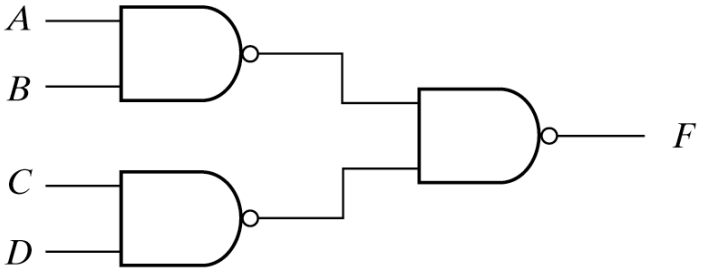


a) Two level with AND-OR



b) Two level with NAND& Invert- OR

SOP



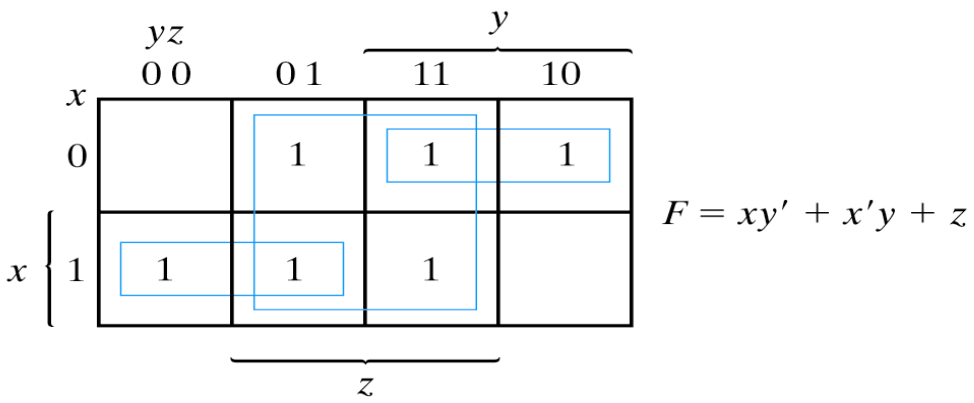
c) Two level with NAND gates
(use this in exam)

Three Ways to Implement F = AB + CD

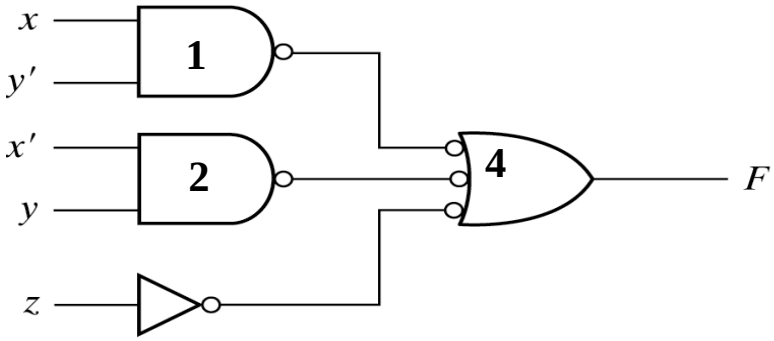
NAND gates implementations -Examples

1: (xy')' 2: (x'y)' 4: ((xy')')' + ((x'y)')' + (z')'= xy' + x'y +z

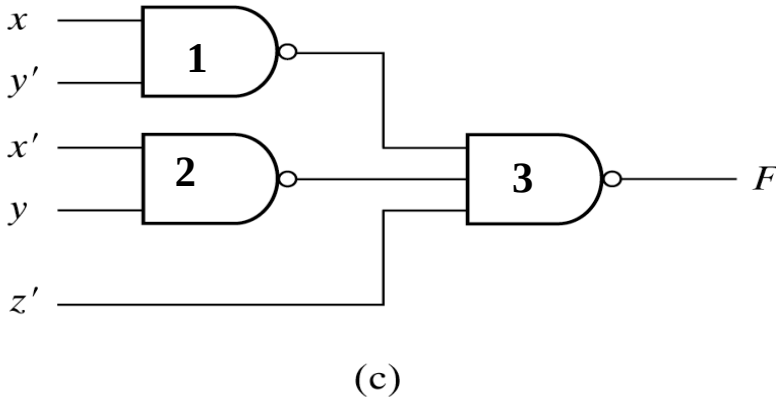
3: ((xy')'(x'y)'(z'))'= (xy')" + (x'y)"+ (z')' = xy' + x'y +z



(a)



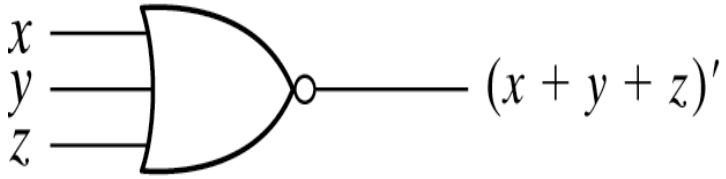
(b)



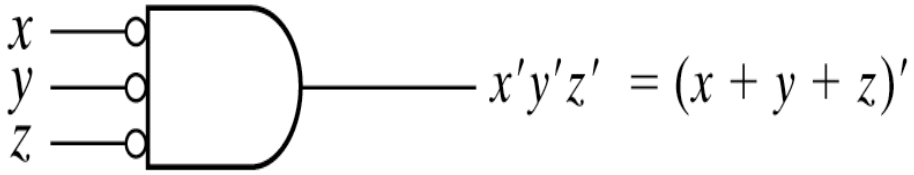
(c)

Using NAND gates to implement SOP

Logic Operations with NOR Gates



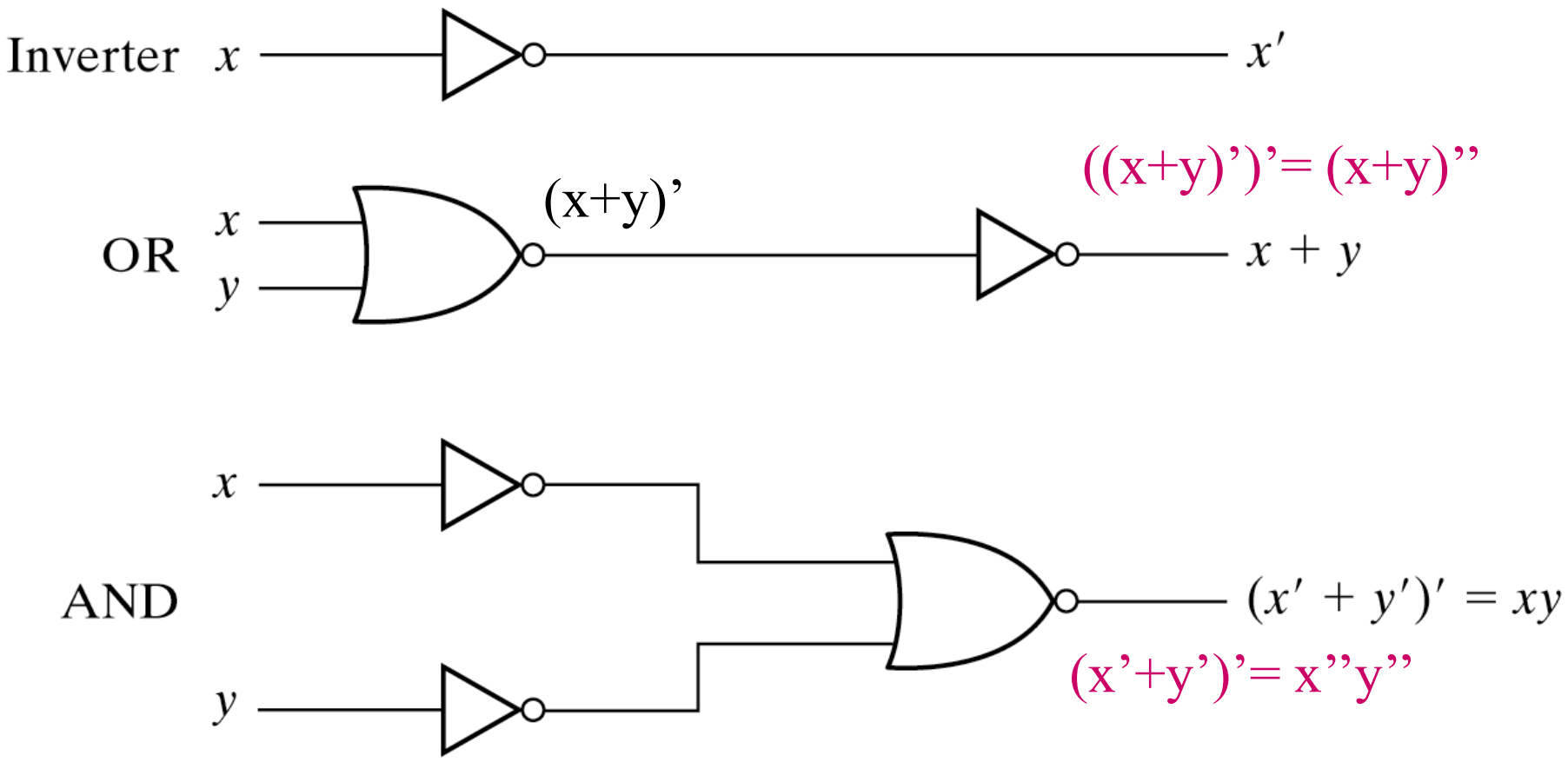
(a) OR–invert



(a) Invert–AND

Two Graphic Symbols for NOR Gate

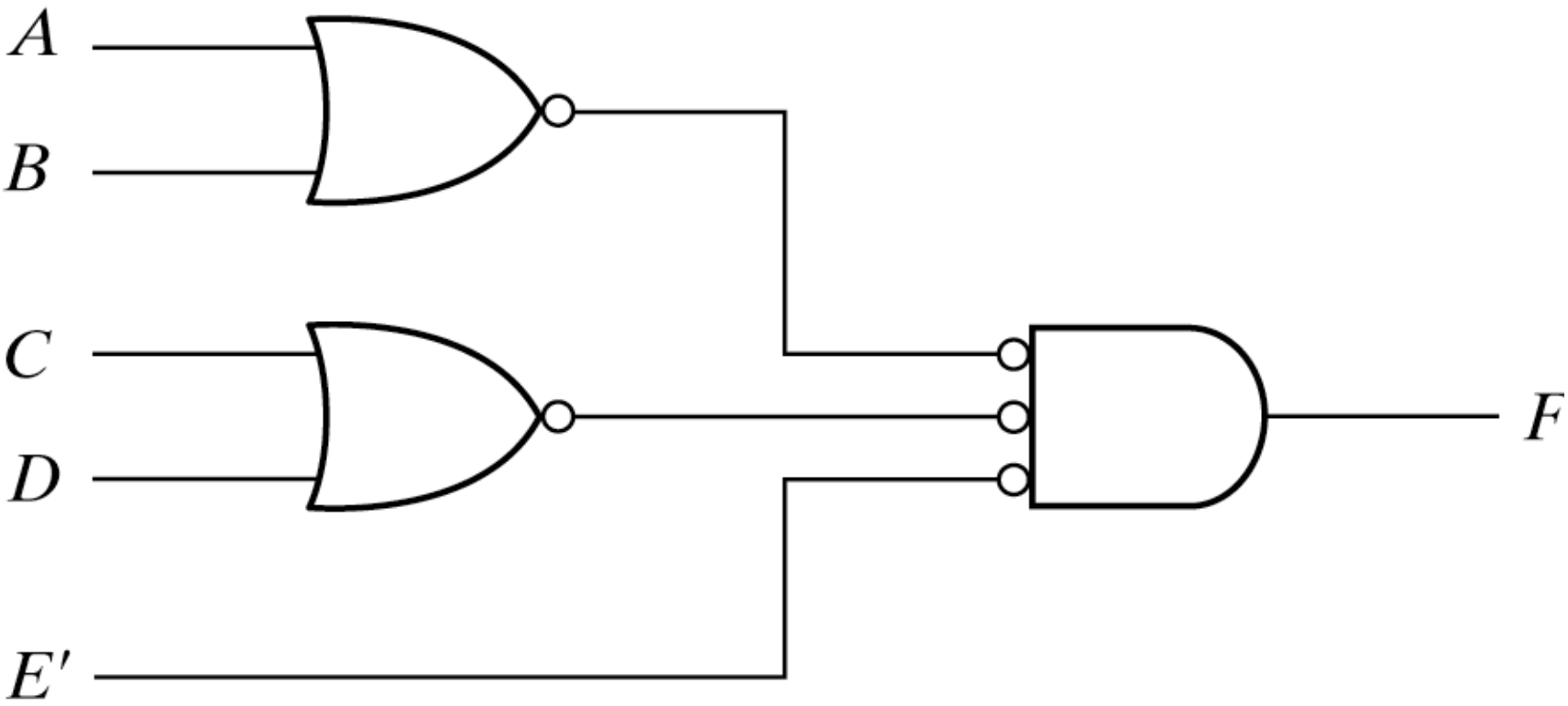
Logic Operations with NOR Gates



Logic Operations with NOR Gates

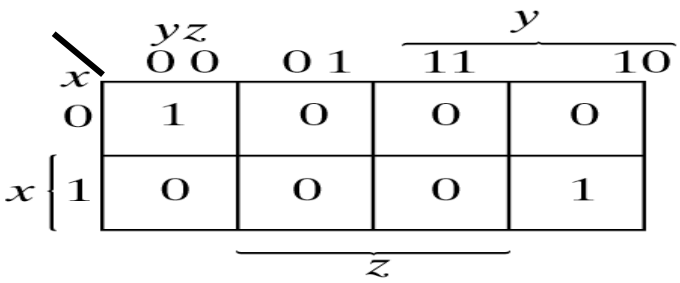
NOR gates Implementation -Examples

POS with NOR



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

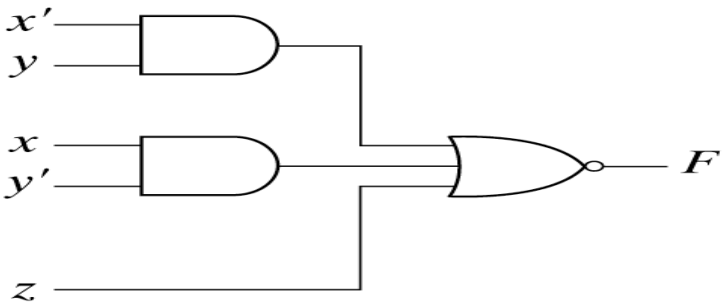
Other implementation examples



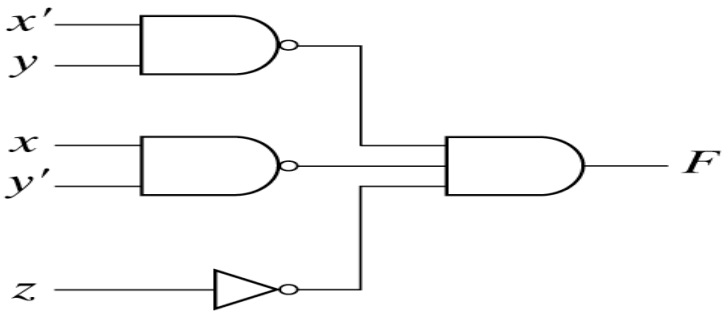
$F = x'y'z' + xyz'$

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

(a) Map simplification in sum of products.

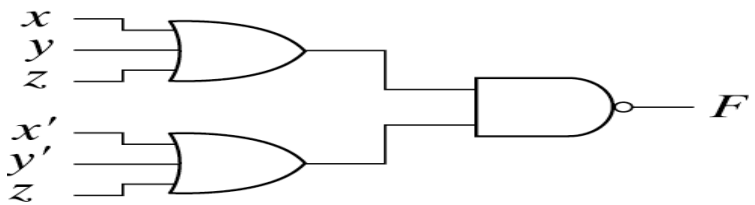


AND-NOR

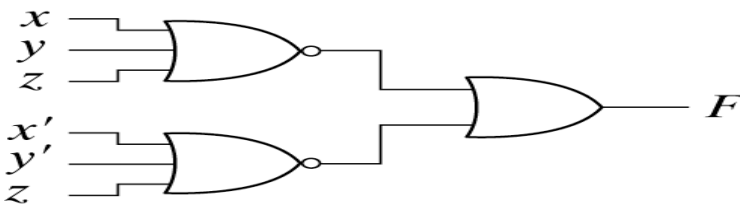


NAND-AND

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



OR-NAND



NOR-OR

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

Other Two-level Implementations