Chapter 3

Gate –Level Minimization

Topics

- Karnaugh Maps (K-Maps)
  - K-Maps with 2, 3 and 4 Variables
  - Representing Boolean Function in a K-map
  - Grouping cells in K-map for minimizing SOP
  - Prime Implicants
- NAND/NOR Implementations

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

→ Can be used to determine POS or SOP.

K-Map with Two variables

B \ A	A'	A
B'	A'B'	AB'
B	A'B	AB

B \ A	0	1
0	00	10
1	01	11

B \ A	0	1
0	m_0	m_2
1	m_1	m_3

K-Map with 3 variables

C \ AB	A'B'	A'B	AB	AB'
C	**A'B'C'**	**A'BC'**	**ABC'**	**AB'C'**
C'	**A'B'C**	**A'BC**	**ABC**	**AB'C**

C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5

Kmap With 4 variables

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

A

B

C

D

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

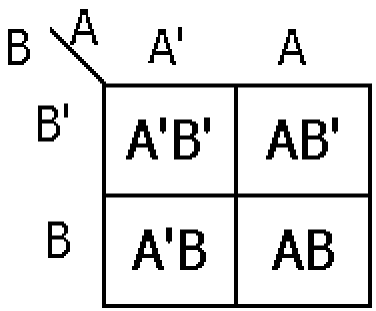
- A group must be as large as possible (i.e. large number of squares)

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

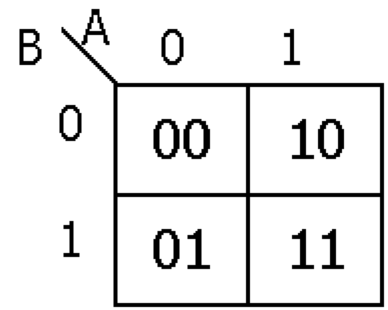
→ the simplified logic expression obtained from a K-map is not always unique. Groupings can be made in different ways.

2 variable Karnaugh Map

$x + x' = 1$

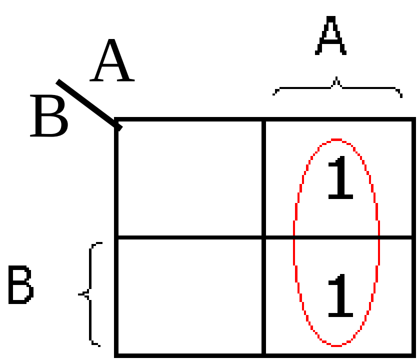


→

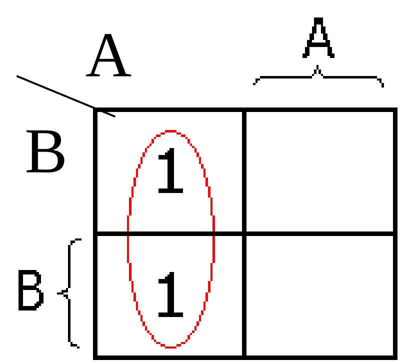


F=AB'+AB

F=A'B'+A'B



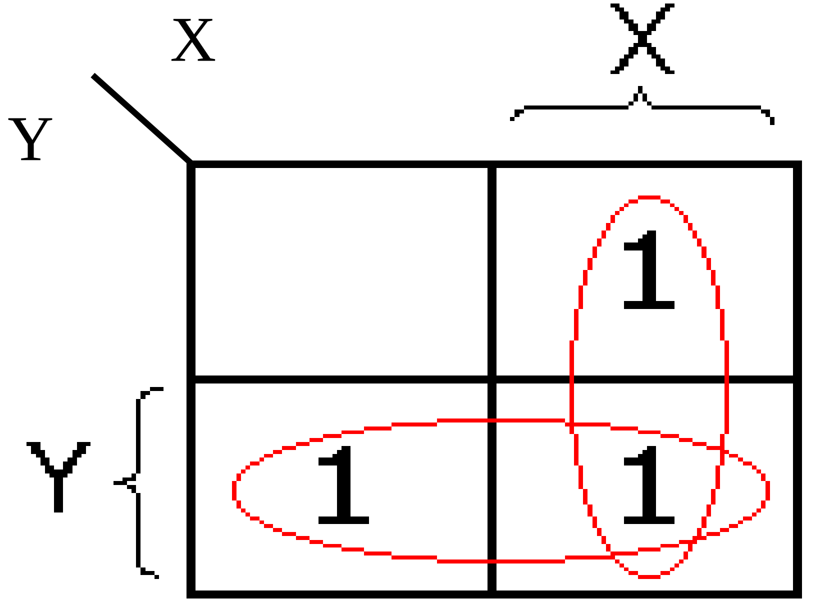
F = A



F = A'

2 variable Karnaugh Map

$F = X'Y + XY + XY'$



$F = X + Y$

3 Variable Karnaugh Map

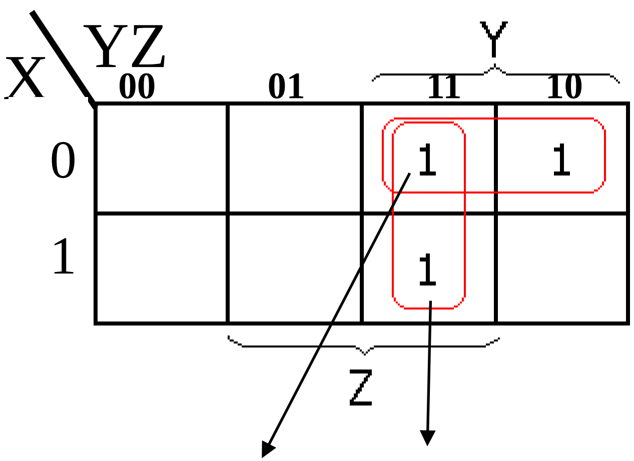
C \ AB				
	A'B'C'	**A'BC'**	**ABC'**	**AB'C'**
	A'B'C	**A'BC**	**ABC**	**AB'C**

⟷

C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5

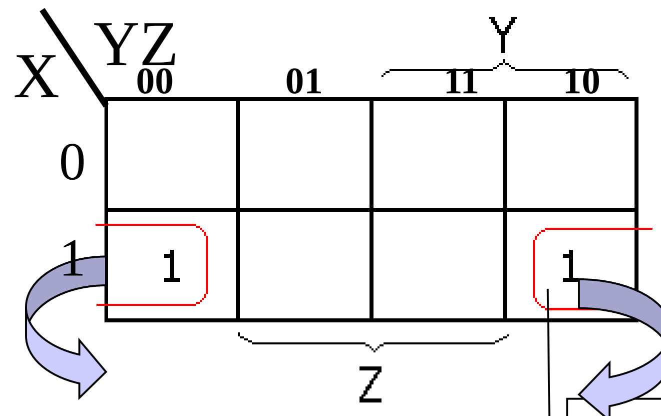
A: columns 11, 10; B: columns 01, 11; C: row 1

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



F = X'Y+YZ

F = XY'Z'+XYZ'

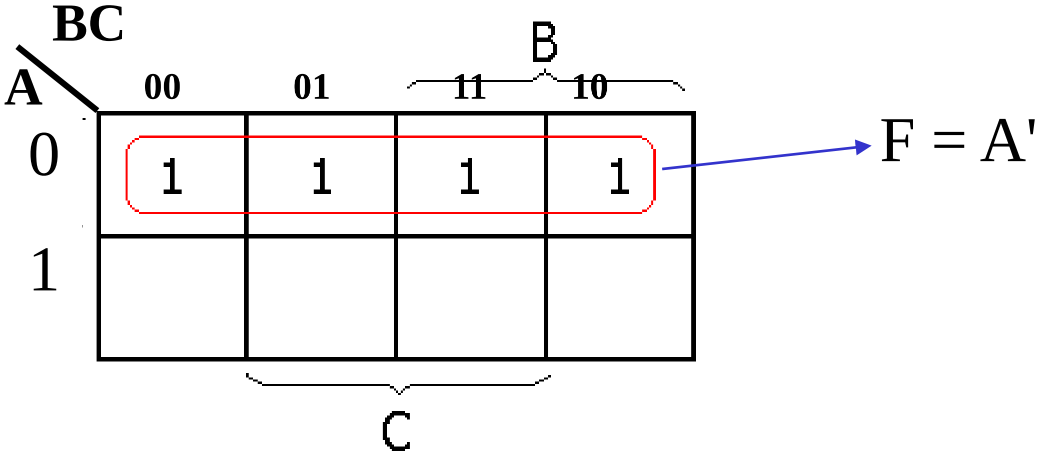


Wrapping around edges

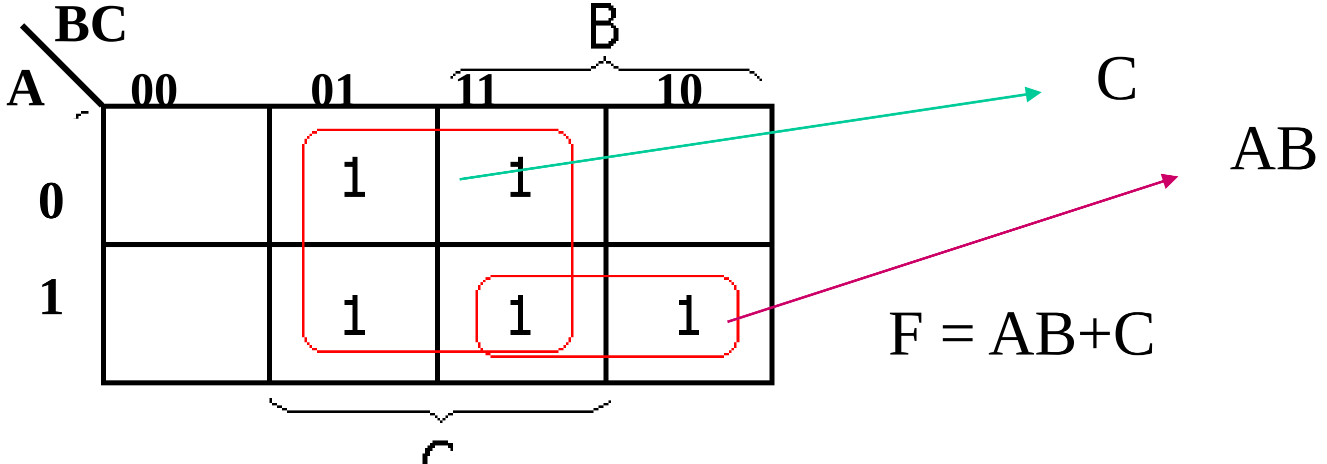
F = XZ'

3 variable Karnaugh Map

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

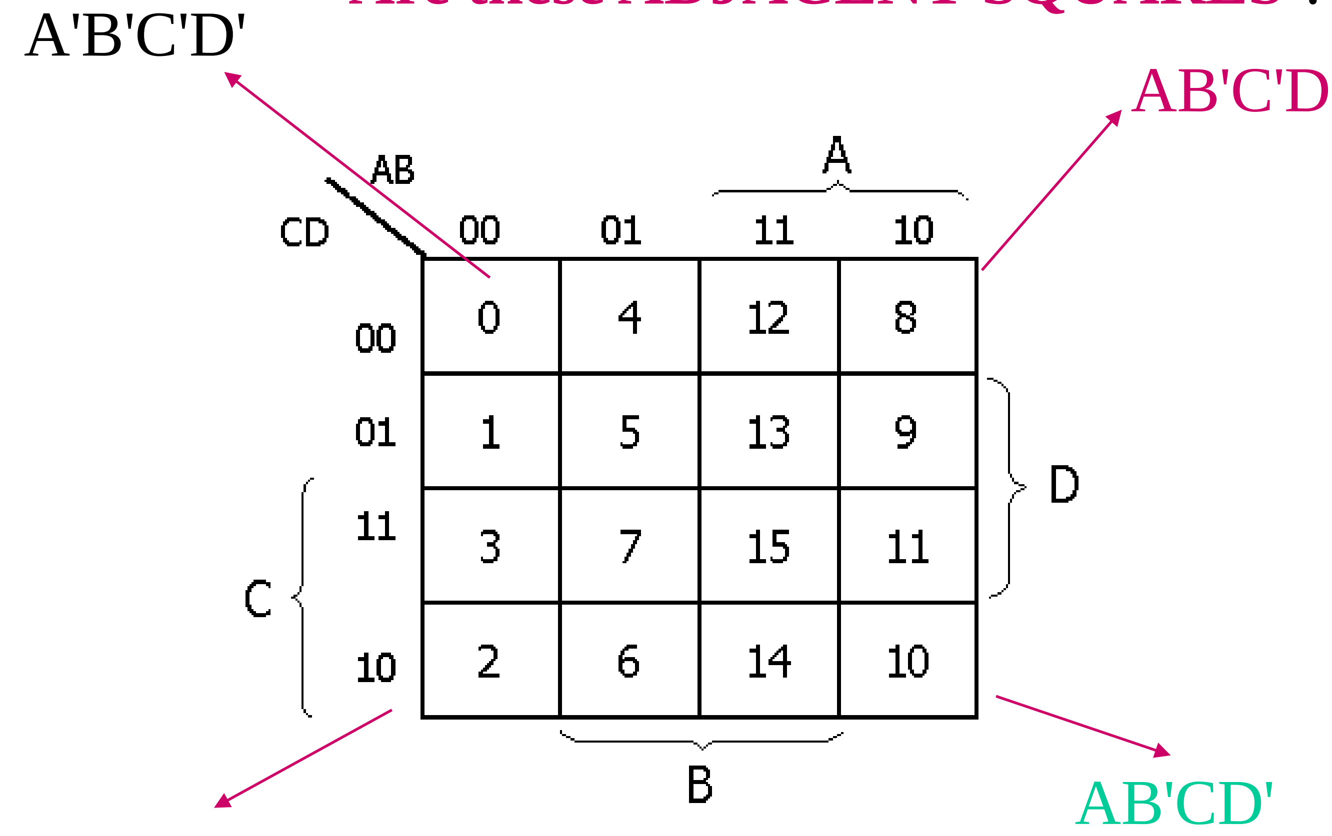


F(A,B,C) = A'BC+A'B'C+AB'C+ABC+ABC'



4 variables K-MAP

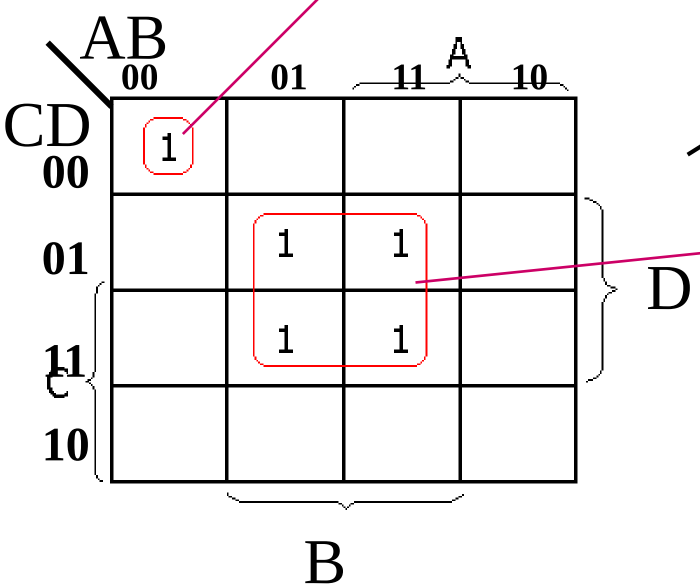
Are these ADJACENT SQUARES ?



Variables A,B,C,D

4 variable K-MAP

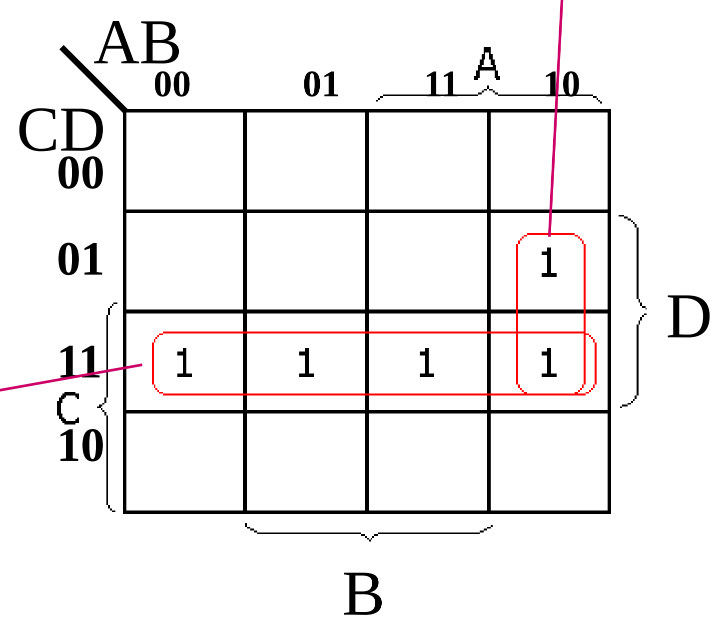
A'B'C'D'



F = BD+A'B'C'D'

BD

AB'D



F = CD+AB'D

CD

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

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

K-map with Don't Care outputs

- Don't care outputs can be either 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)

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

- •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 = A'B'+CD or F= A'D+CD

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

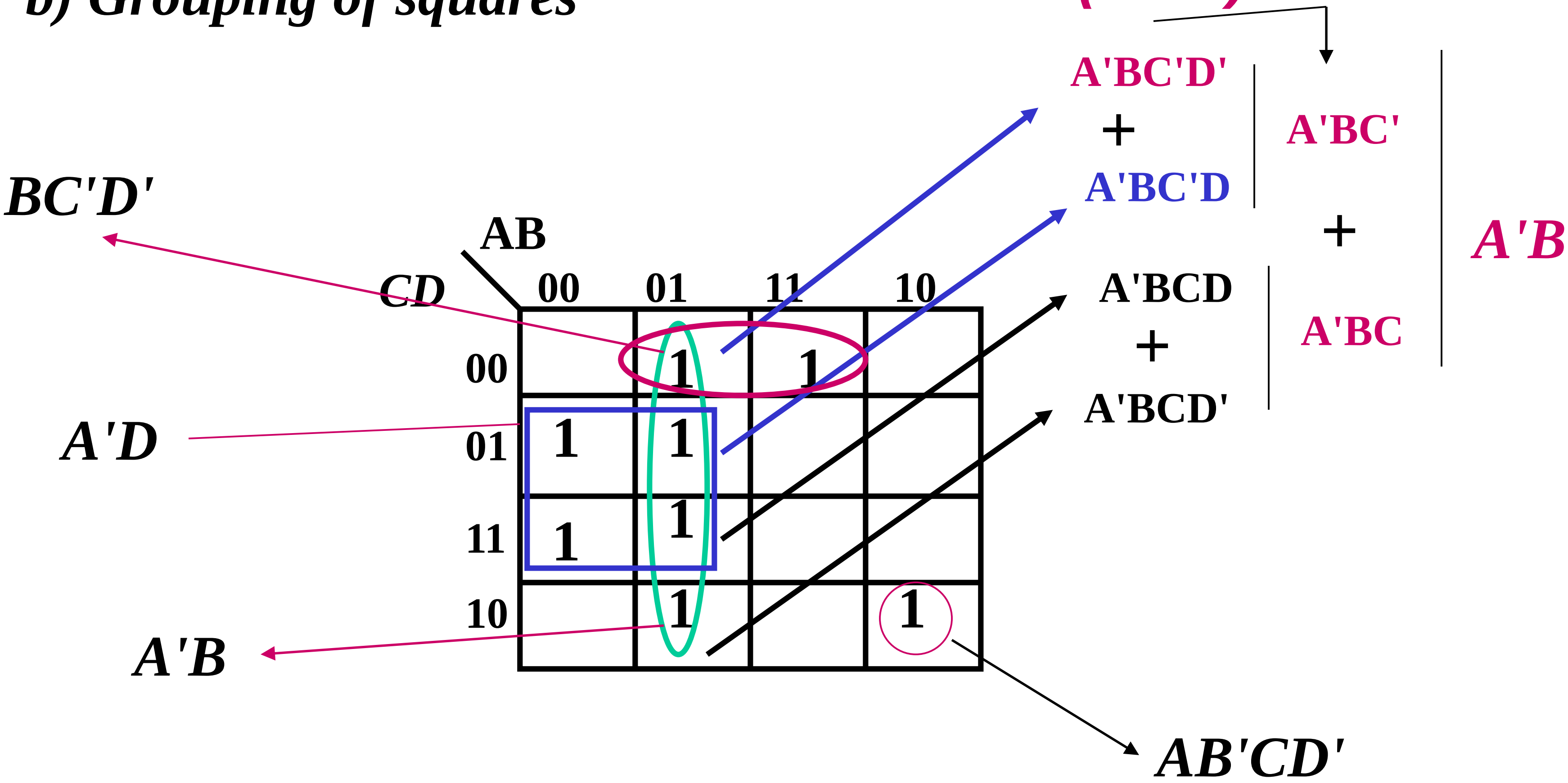
$$F(A,B,C,D) = \Sigma\, m\,(1, 3, 4, 5, 6, 7, 10,12)$$

a) Building K-Map for F

CD \ AB	00	01	11	10
00	0	4 **1**	12 **1**	8
01	**1** 1	5 **1**	13	9
11	**1** 3	7 **1**	15	11
10	2	6 **1**	14	**1** 10

b) Grouping of squares

$A'BC'(D+D') = A'BC'$



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

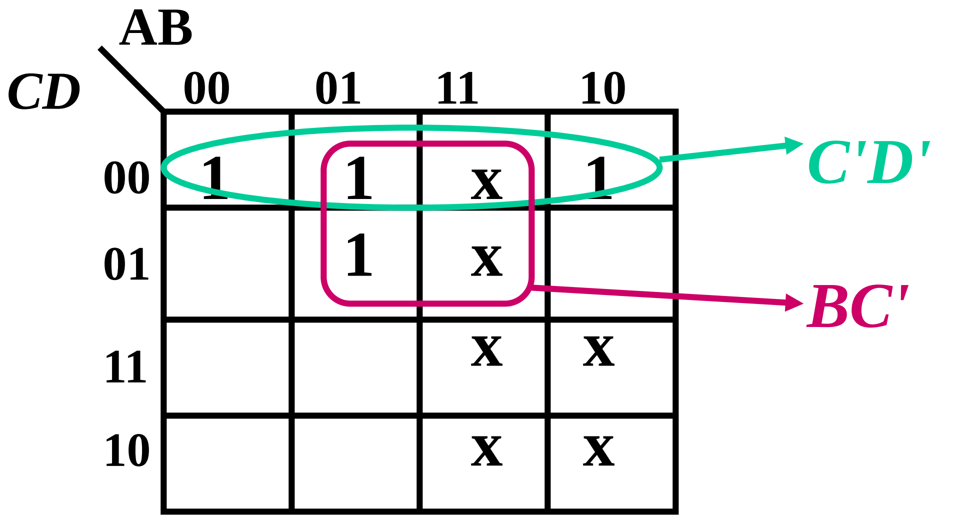
CD \ AB	00	01	11	10
00	1	1	x	1
01	0	1	x	0
11	0	0	x	x
10	0	0	x	x

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

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

b) Obtain Sum of Products for F



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

Prime implicants

When grouping square:

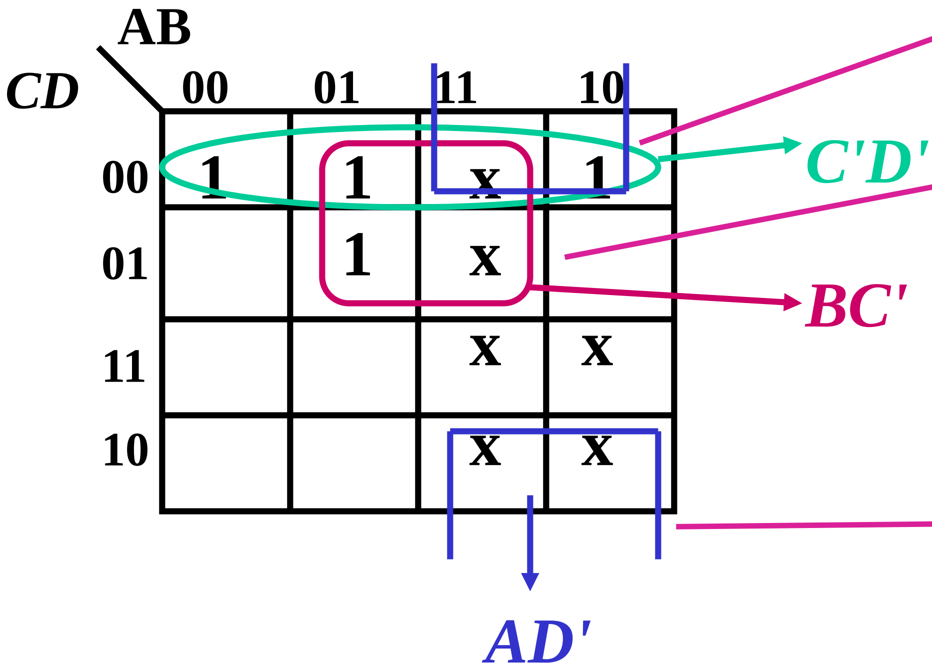
- A group should contain a maximum of adjacent cells
  - → Known as *PRIME IMPLICANT*
  - → Only valid if the group is not contained in a larger group

- Each group represents one product term in the function

- *Essential Prime Implicant*
  - Has at least one square that is not covered by any other group
- *Optional prime implicant*
  - All of its squares covered by other groups

- A function should contain a minimum set of product terms, when selecting groups:
  - Include all *Essential Prime Implicants*
  - Select among the *Optional prime implicants,* so that all cells with a 1 have been covered.

b) Obtain Sum of Products for F



Essential Prime Implicant

Essential Prime Implicant

Optional Prime Implicant

$F = BC' + C'D'$

Function made up of only essential prime implicants

b) Obtain Sum of Products for F

ab \ cd	00	01	11	10
00	m_0	m_1 1	m_3 1	m_2
01	m_4 1	m_5 1	m_7	m_6
11	m_{12} 1	m_{13} 1	m_{15} 1	m_{14} 1
10	m_8	m_9	m_{11} 1	m_{10} 1

$$F = ac + bc' + a'b'd$$

- Rules to select optional PIs:
  select optional PI that covers the most "1" cells not already covered; ie, select minimum number of optional PIs to cover all "1" cells not already covered. There may be multiple ways.

- Groups circled in red are essential prime implicants
  - bc', ac
- Groups circled in blue are optional prime implicants
  - ab, a'c'd, a'b'd, b'cd
- To create minimized function, only one optional prime implicant is added.
  - Any other combination would lead to 4 terms

b) Obtain Sum of Products for F

ab \ cd	00	01	11	10
00	m_0 1	m_1 1	m_3 1	m_2
01	m_4	m_5	m_7 1	m_6
11	m_{12}	m_{13} 1	m_{15} 1	m_{14}
10	m_8 1	m_9 1	m_{11}	m_{10} 1

- Groups circled in red are essential prime implicants
  - b'c', ab'd'
- Groups circled in other colors are optional prime implicants
  - a'b'd, a'cd, bcd, abd,
- Note that any other combination leads to more terms.

$$F = b'c' + ab'd' + a'cd + a'b'd + abd$$

b) Obtain Sum of Products for F

ab \ cd	00	01	11	10
00	m_0 1	m_1 1	m_3	m_2 1
01	m_4 1	m_5 1	m_7 1	m_6 1
11	m_{12}	m_{13}	m_{15} 1	m_{14}
10	m_8	m_9	m_{11}	m_{10} 1

- Groups circled in red are essential prime implicants
  - a'd', a'c', bcd, b'cd'
- Groups circled in blue is an optional prime implicant
  - a‘b,

$$F = a'd' + a'c' + bcd + b'cd'$$

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

$$F'' = (B'D + C)' = (B''+D')\bullet C'$$

$$F = (B + D')\bullet C'$$

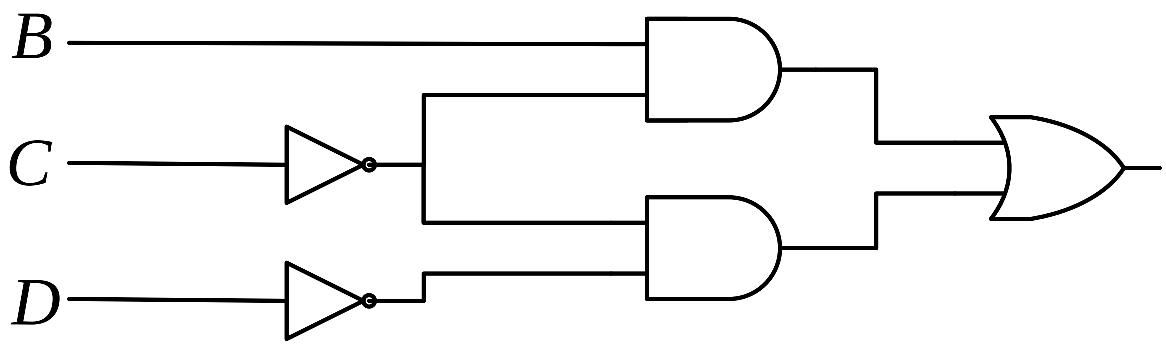
CD \ AB	00	01	11	10
00			x	
01	0		x	0
11	0	0	x	x
10	0	0	x	x

B'D

C

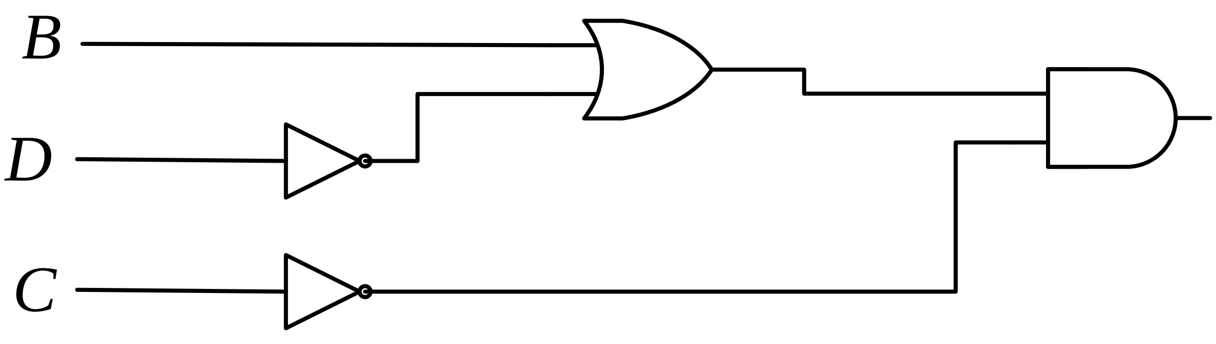
Two Level Implementations

SOP : Two Level Implementation



F = BC' + C'D'

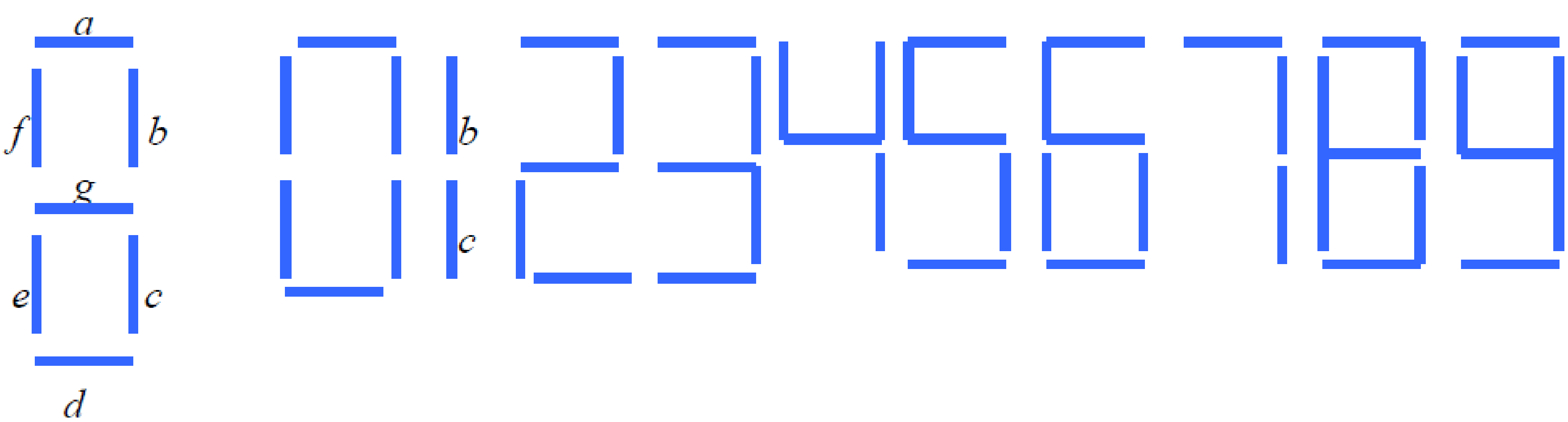
POS : Two Level Implementation



F = (B +D')•C'

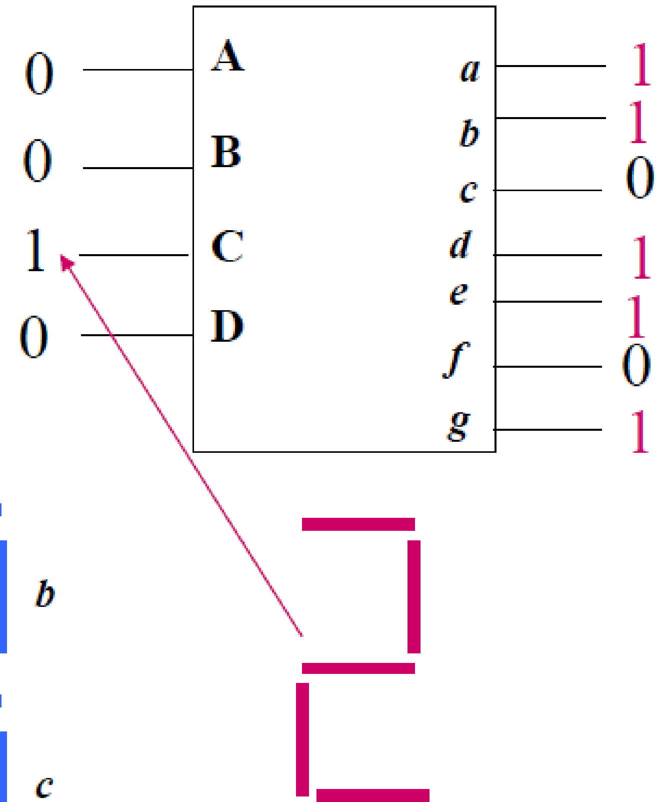
Seven Segment Decoder -Example

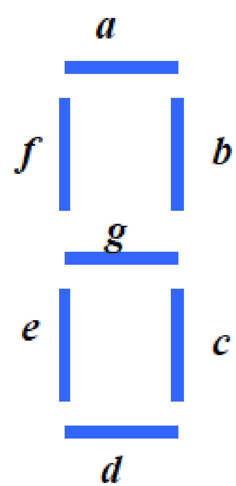
a BCD to Seven Segment Decoder inputs data in BCD form and converts it to a seven segment output



(a) Segment designation (b) Numerical designation for display

A- BCD to Seven Segment Decoder





A	B	C	D	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

Don't care terms

K-MAP

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

$a =$

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

$b =$

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

c =

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

d =

a

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

a = A + BD + B'D'+ C

b

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

b = CD + C'D' + B'

c

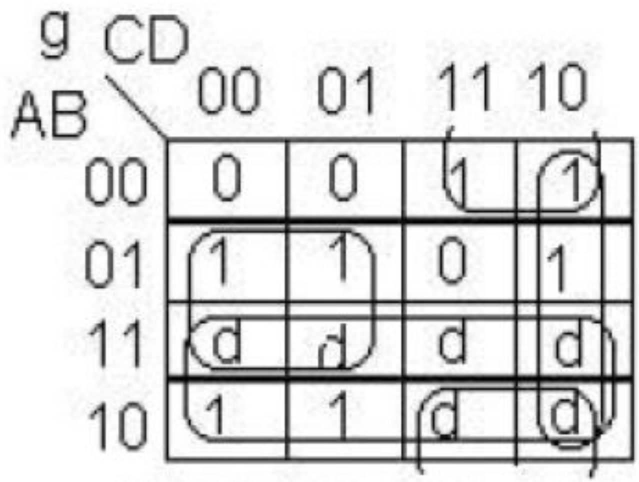
AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	d	d	d	d
10	1	1	d	d

c= B + C'+ D

d

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

d=BC'D+B'C+B'D'+CD'



g=BC'+B'C+CD'+A

Implementations using NAND & NOR Gates

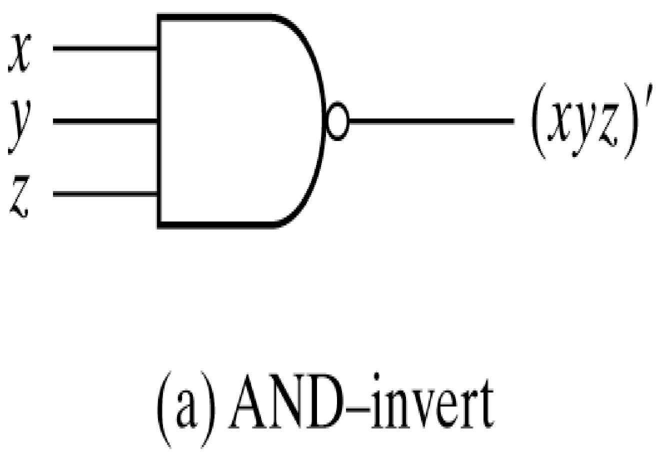
• Digital circuits are frequently constructed with NAND or NOR gates rather with AND and OR gates.

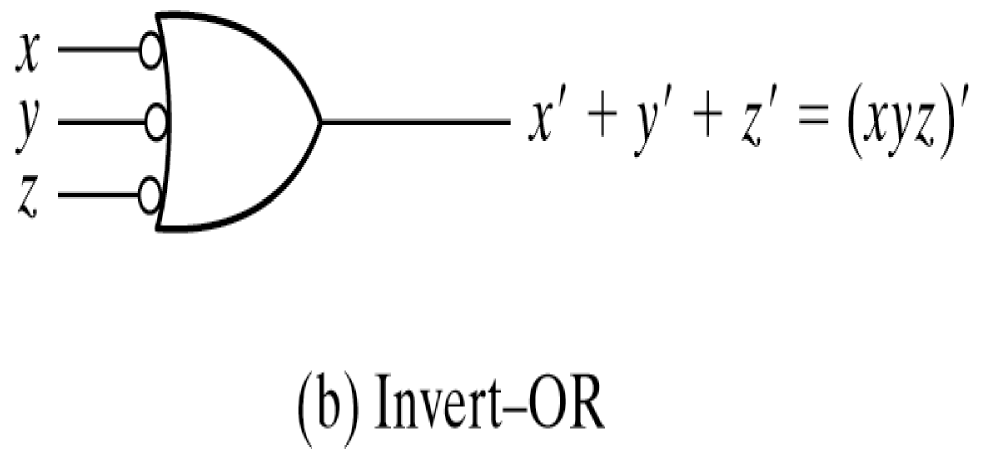
→ Both NAND and NOR gates are very valuable as any design can be realized using either one.

•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.

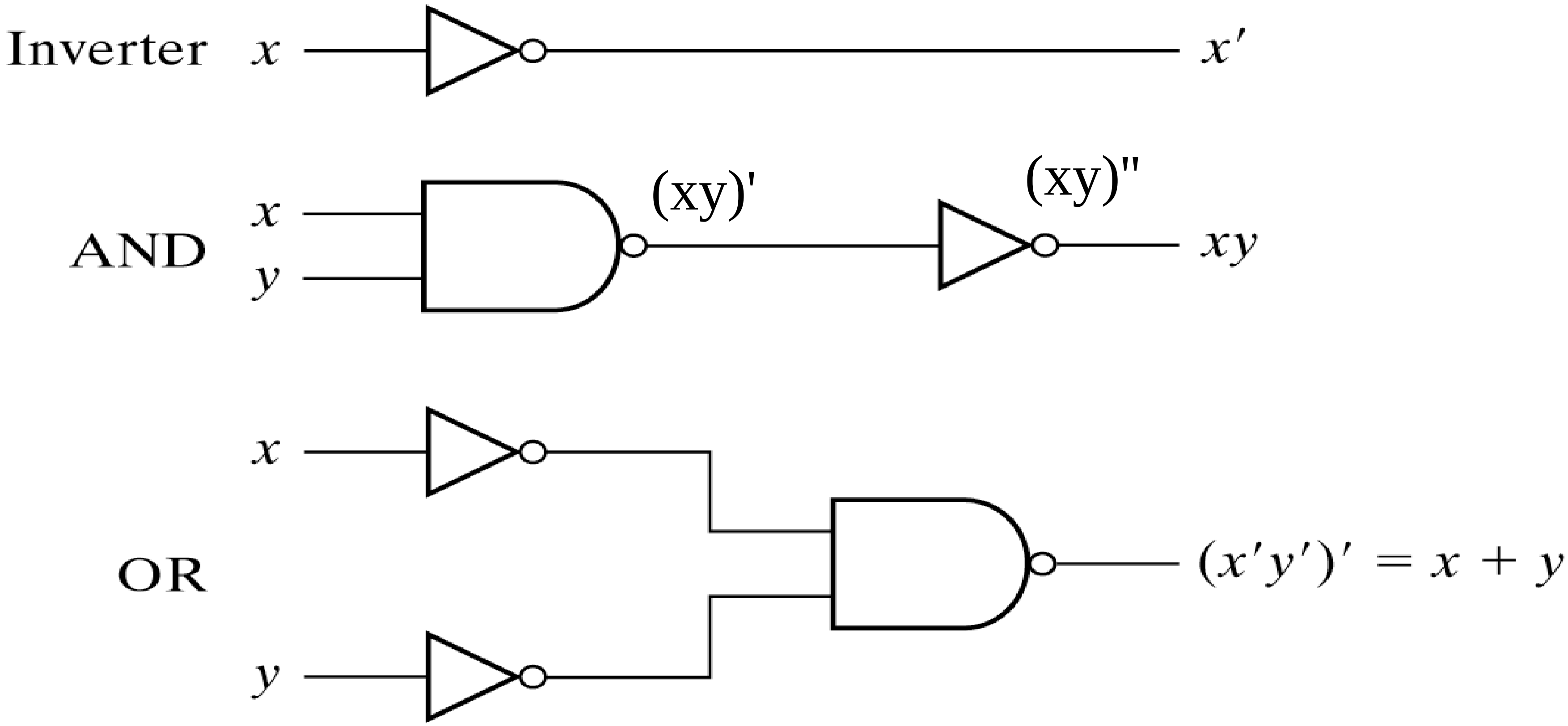
Logic Operations with NAND Gates





Two Graphic Symbols for NAND Gate

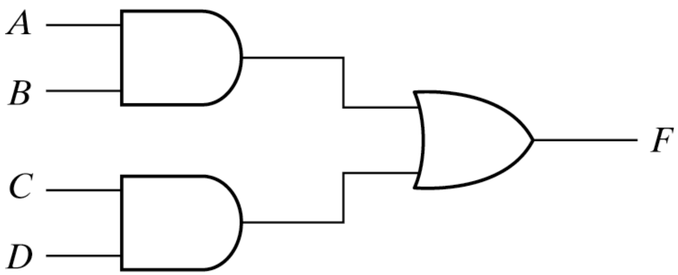
Logic Operations with NAND Gates



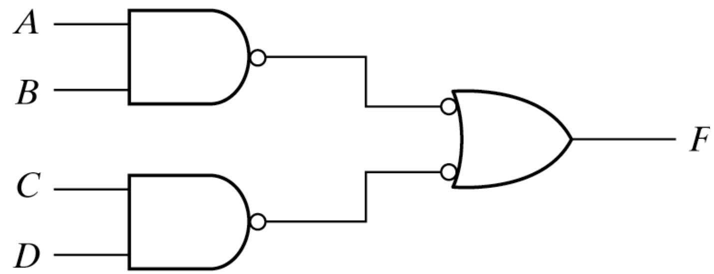
Logic Operations with NAND Gates

NAND gates Implementations

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

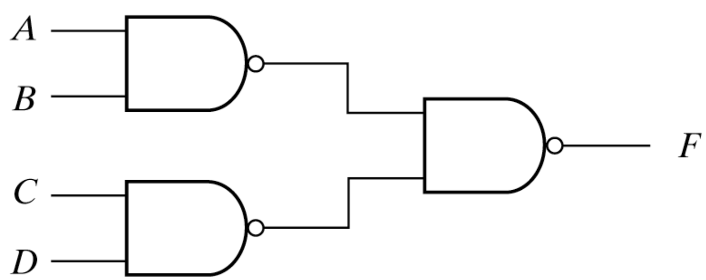


a) Two level with AND-OR



b) Two level with NAND& Invert- OR

SOP



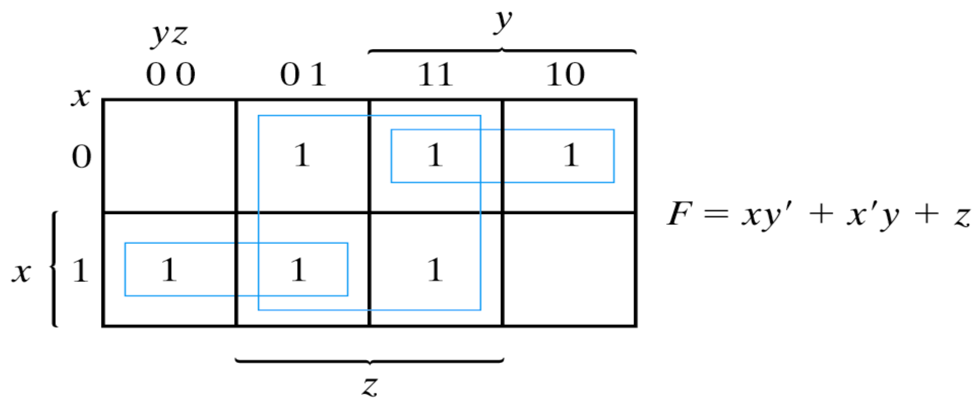
c) Two level with NAND gates
(use this in exam)

Three Ways to Implement F = AB + CD

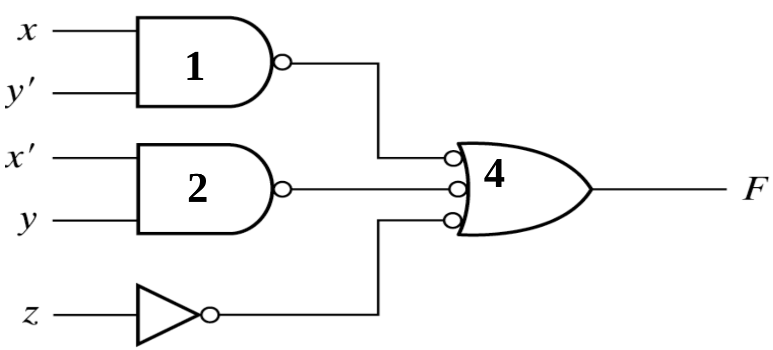
NAND gates implementations -Examples

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

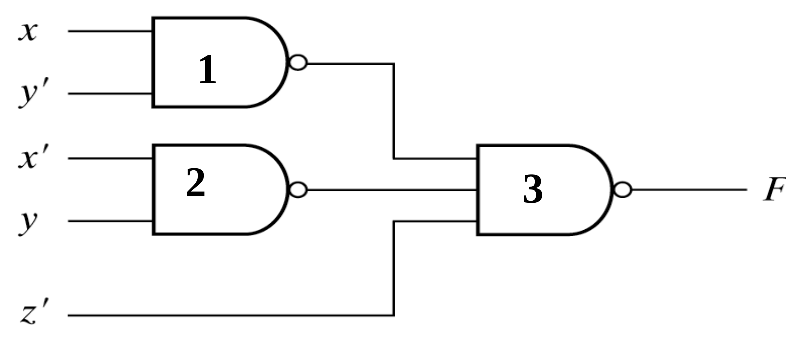
3: $((xy')'(x'y)'(z'))' = (xy')'' + (x'y)'' + (z')' = xy' + x'y + z$



(a)



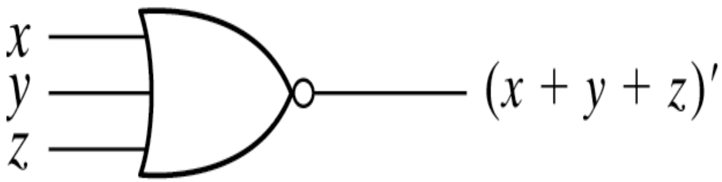
(b)



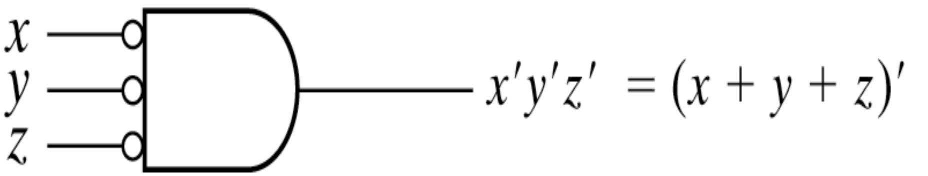
(c)

Using NAND gates to implement SOP

Logic Operations with NOR Gates



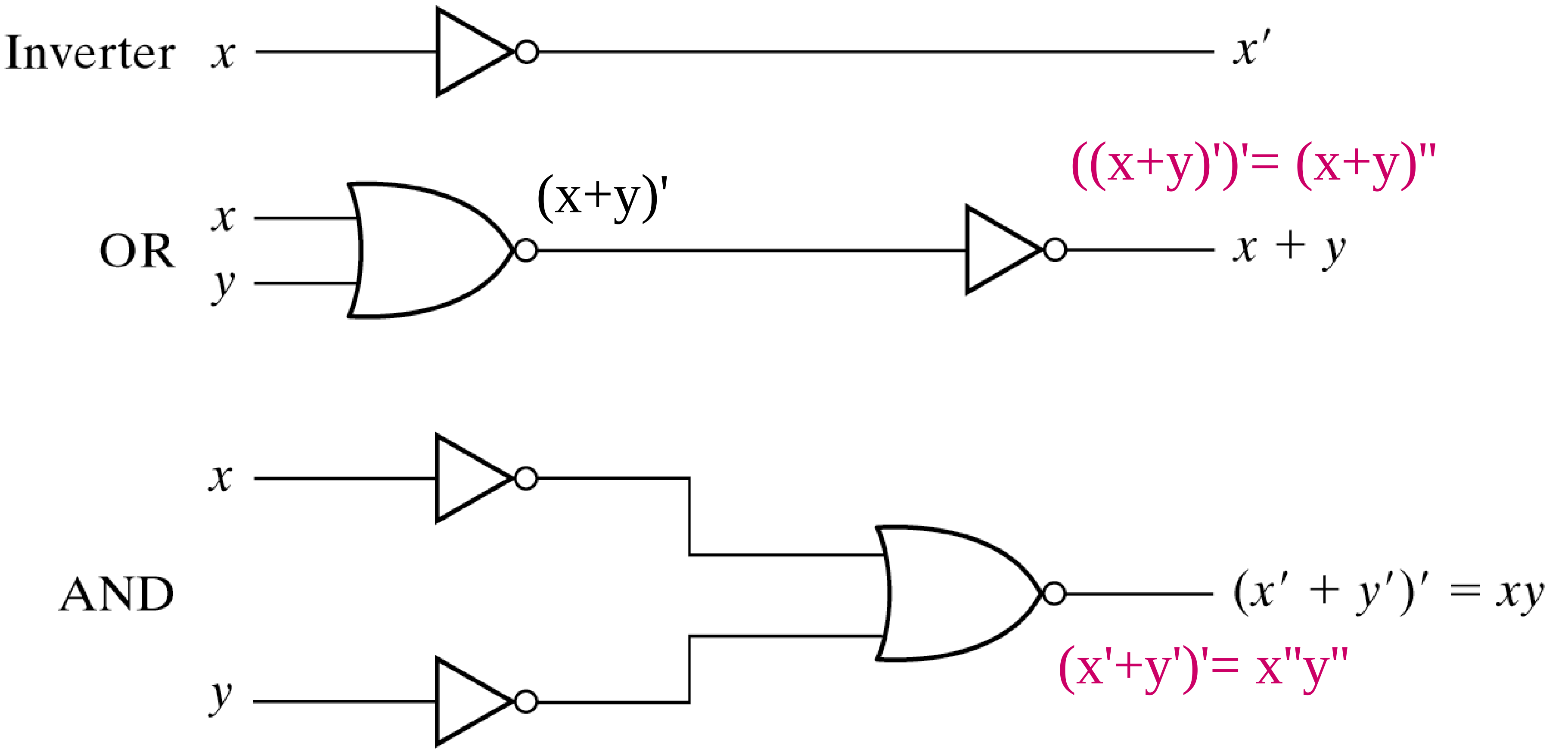
(a) OR–invert



(a) Invert–AND

Two Graphic Symbols for NOR Gate

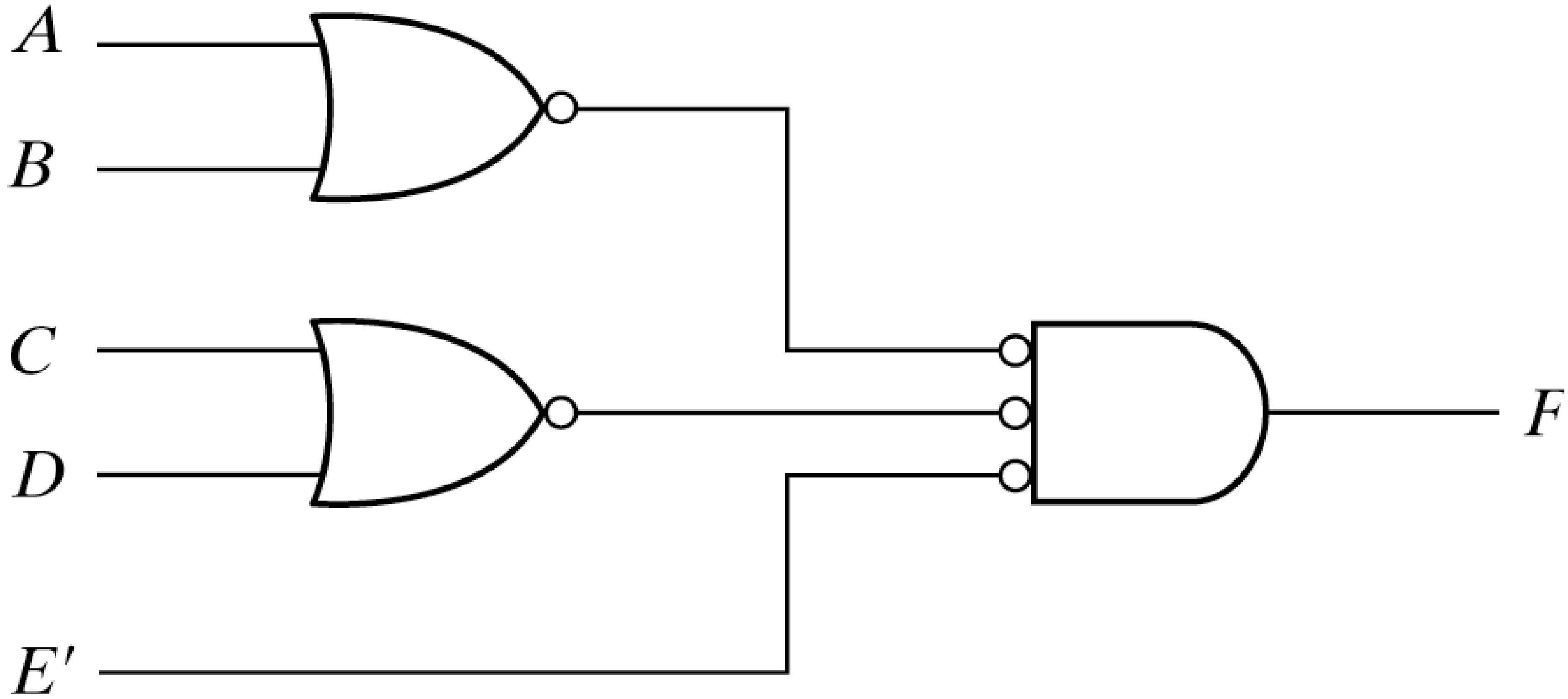
Logic Operations with NOR Gates



Logic Operations with NOR Gates

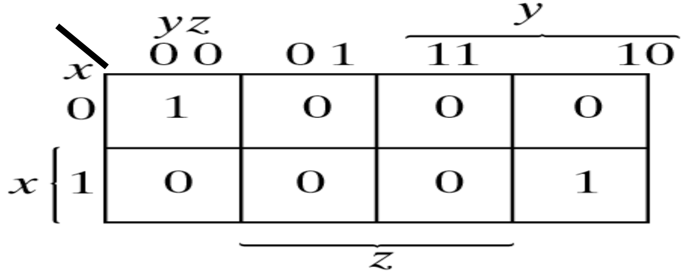
NOR gates Implementation -Examples

POS with NOR



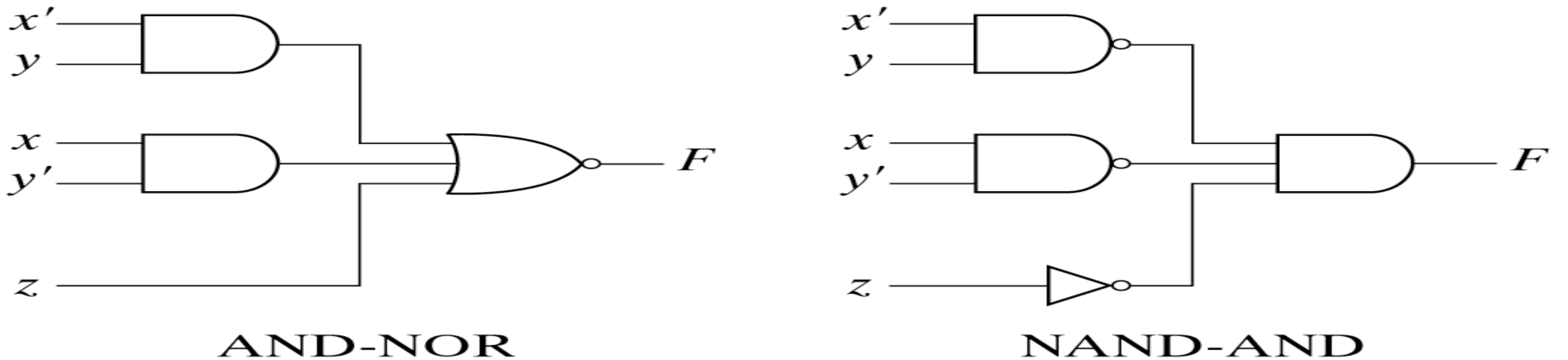
Implementing $F = (A + B)(C + D)E$

Other implementation examples

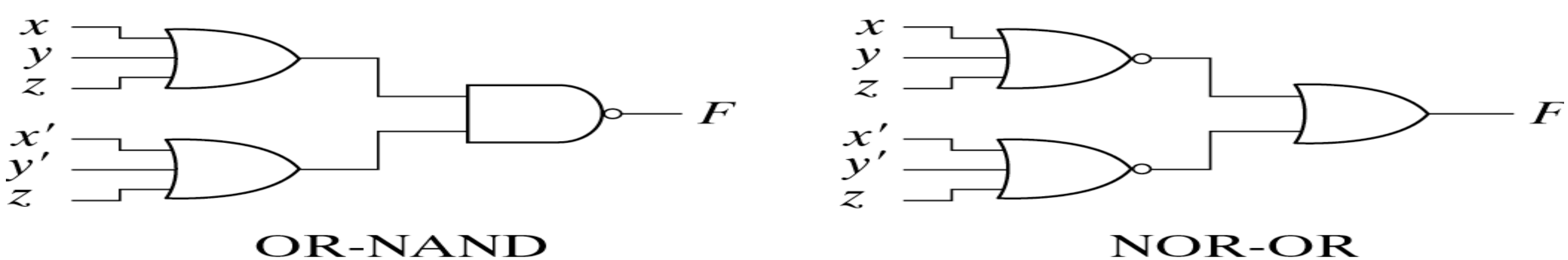


$F = x'y'z' + xyz'$
$F' = x'y + xy' + z$

(a) Map simplification in sum of products.



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



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

Other Two-level Implementations