

---

# 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

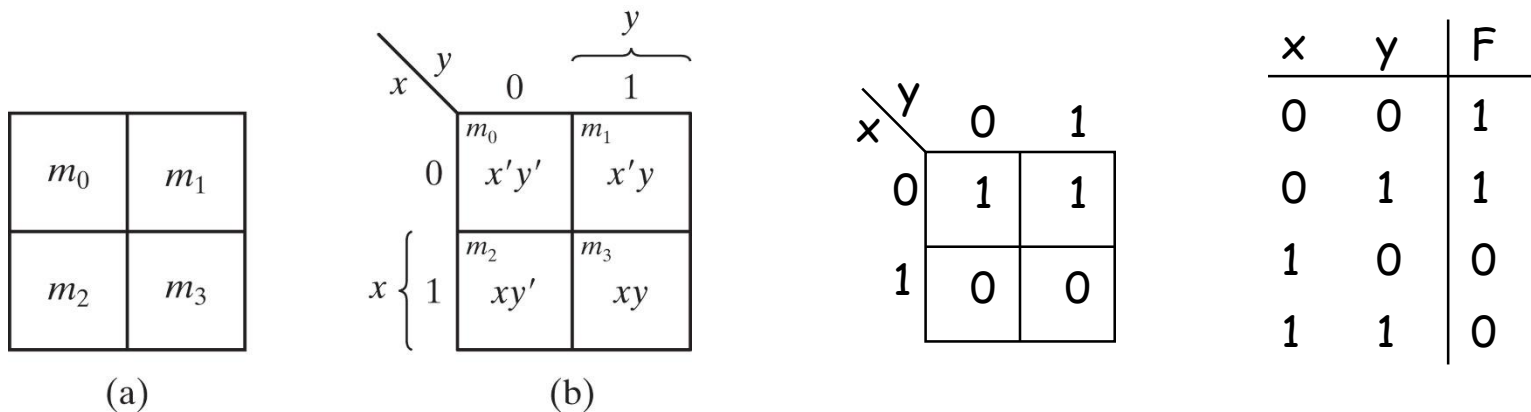
---

- 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.

# Karnaugh Maps

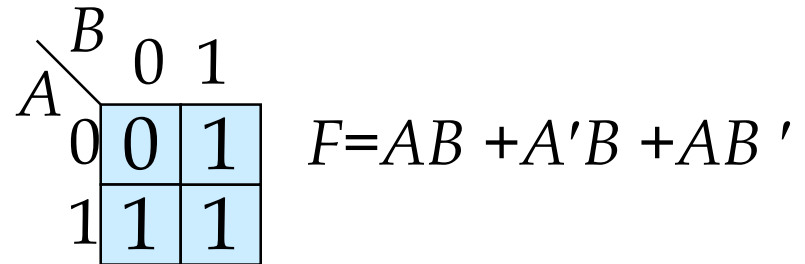
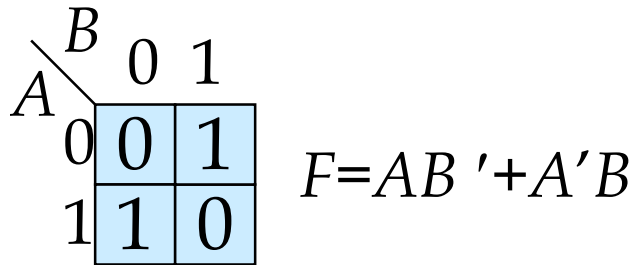
- Alternate way of representing Boolean function
  - All rows of truth table represented with a square
  - Each square represents a minterm: assign the values of outputs to the corresponding minterm in K-map
- Easy to convert between truth table, K-map, and SOP
  - Unoptimized form: number of 1's in K-map equals number of minterms (products) in SOP
  - Optimized form: reduced number of terms

$$F = \Sigma(m_0, m_1) = x'y + x'y'$$

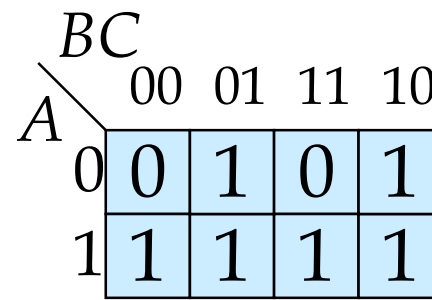
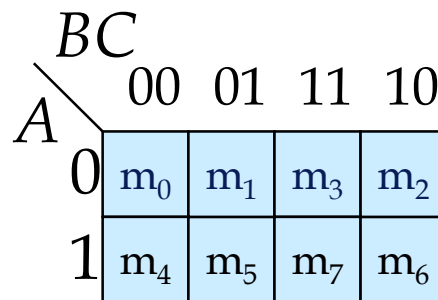
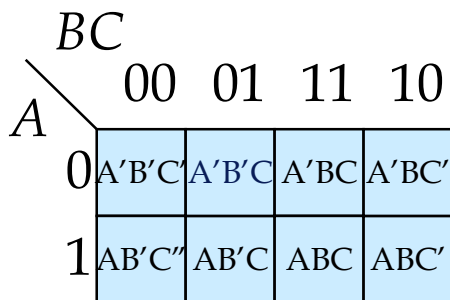


# K-Map with 2 or 3 Variables

- A Karnaugh map is a graphical tool for assisting in the general simplification procedure.
- **Two variable maps**



- **Three variable maps**



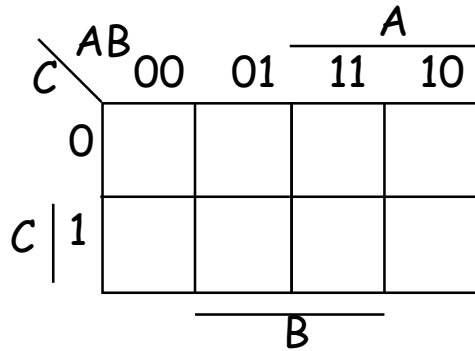
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

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

# K-Map Numbering Scheme

---

- Numbering scheme based on Gray-code
  - e.g., 00, 01, 11, 10
  - Only a single bit changes in code for adjacent map cells
    - *Adjacent cells may not touch each other (wrapping around edges)*
  - This is necessary to observe the variable transitions



# K-Map With 4 Variables

CD \ AB		C			
		00	01	11	10
A	00	$m_0$ $A'B'C'D'$	$m_1$ $A'B'C'D$	$m_3$ $A'B'CD$	$m_2$ $A'B'CD'$
	01	$m_4$ $A'BC'D'$	$m_5$ $A'BC'D$	$m_7$ $A'BCD$	$m_6$ $A'BCD'$
	11	$m_{12}$ $ABC'D'$	$m_{13}$ $ABC'D$	$m_{15}$ $ABCD$	$m_{14}$ $ABCD'$
	10	$m_8$ $AB'C'D'$	$m_9$ $AB'C'D$	$m_{11}$ $AB'CD$	$m_{10}$ $AB'CD'$

# 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			
		00	01	11	10
AB	00	1	0	0	0
	01	1	1	0	0
	11	0	1	0	1
	10	1	0	0	1

→ Consider the squares with 1's to simplify **SOP**

→ Consider the squares with 0's to simplify **POS**

# Karnaugh Maps - grouping squares

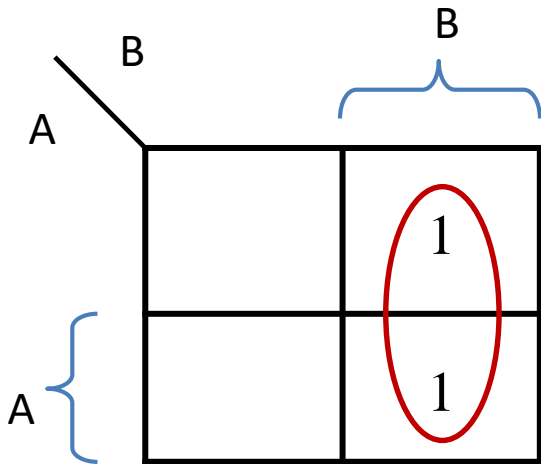
---

- **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

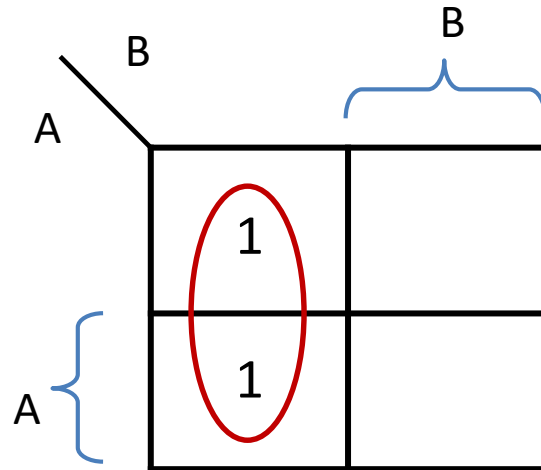
$$x + x' = 1$$

$$F = A'B + AB$$



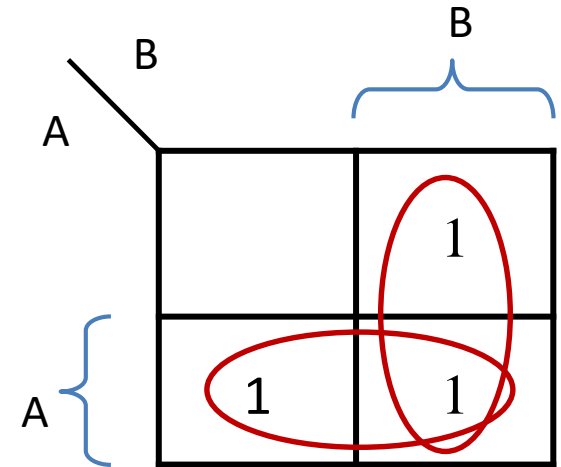
$$F = B$$

$$F = A'B' + AB'$$



$$F = B'$$

$$F = A'B + AB + AB'$$



$$F = A + B$$

# 3 Variable Karnaugh Map

**AB**

**C**

<b>A'B'C'</b>	<b>A'BC'</b>	<b>ABC'</b>	<b>AB'C'</b>
<b>A'B'C</b>	<b>A'BC</b>	<b>ABC</b>	<b>AB'C</b>

**AB**

**C**

	00	01	11	10
<b>C</b> 0	0	2	6	4
<b>C</b> 1	1	3	7	5

**B**

$$F = XY'Z' + XYZ'$$

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

**YZ**

**X**

	00	01	11	10
<b>X</b> 0			1	1
<b>X</b> 1			1	

$$F = X'Y + YZ$$

**YZ**

**X**

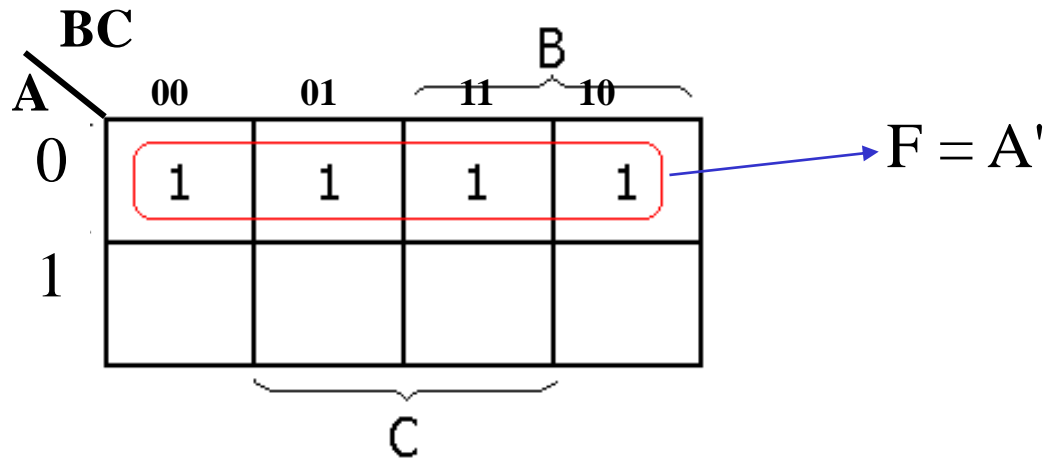
	00	01	11	10
<b>X</b> 0				
<b>X</b> 1	1			1

$$F = XZ'$$

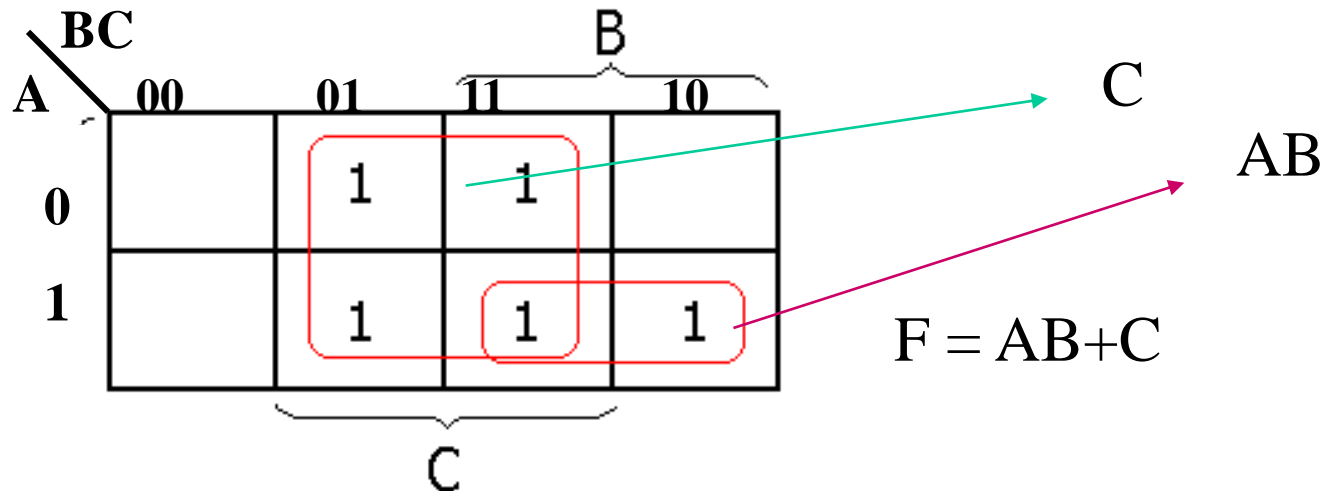
*Wrapping around edges*

# 3 variable Karnaugh Map

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

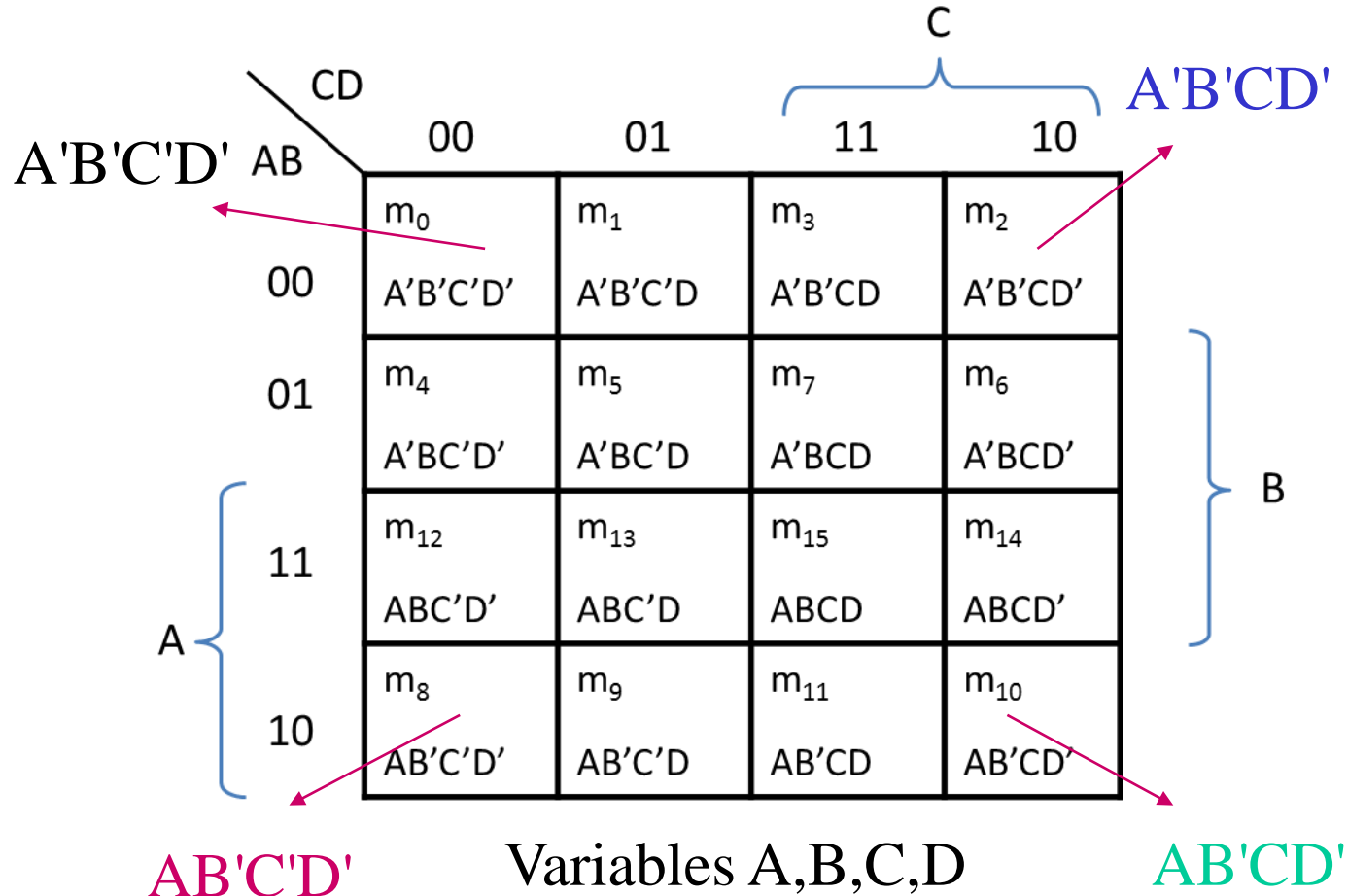


$$F(A,B,C) = A'BC + A'B'C + AB'C + ABC + ABC'$$

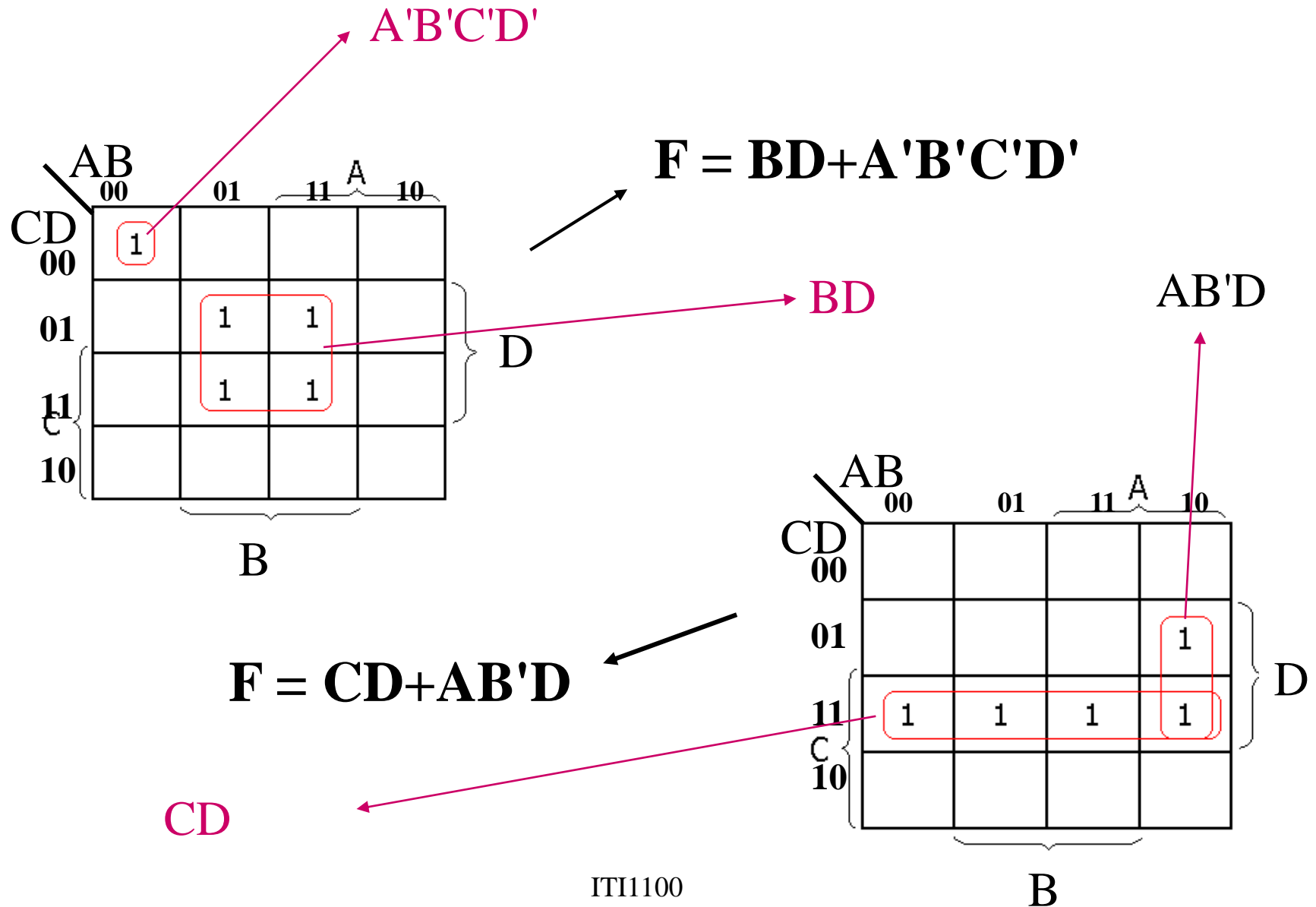


# 4 variables K-MAP

**Are these ADJACENT SQUARES ?**



# 4 variable K-MAP



# 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.

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

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

# 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:  $F(A,B,C,D) = \Sigma m(1,3,7,11,15)$  with  $\Sigma d(0,2,5)$

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

The K-map shows a 4x4 grid of cells. The top row (CD=00) contains X, 1, 1, X. The second row (CD=01) contains 0, X, 1, 0. The third row (CD=11) contains 0, 0, 1, 0. The bottom row (CD=10) contains 0, 0, 1, 0. A pink box groups the top row (X, 1, 1, X). A blue box groups the third column (1, 1, 1, 1). A red arrow points from the pink box to the equation  $F = A'B' + CD$ . A green arrow points from the blue box to the equation  $F = A'D + CD$ .

$$F = A'B' + CD \quad \text{or} \quad F = A'D + CD$$

- 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

# 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				

Using K-maps for the 4 variable we obtain:

$$W = A + BD + BC$$

$$X = BC'D' + AD$$

$$Y = CD + B'C + AD'$$

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

Don't care terms

# *K-Maps- Examples*

*1- simplify the following expression using K-Maps*

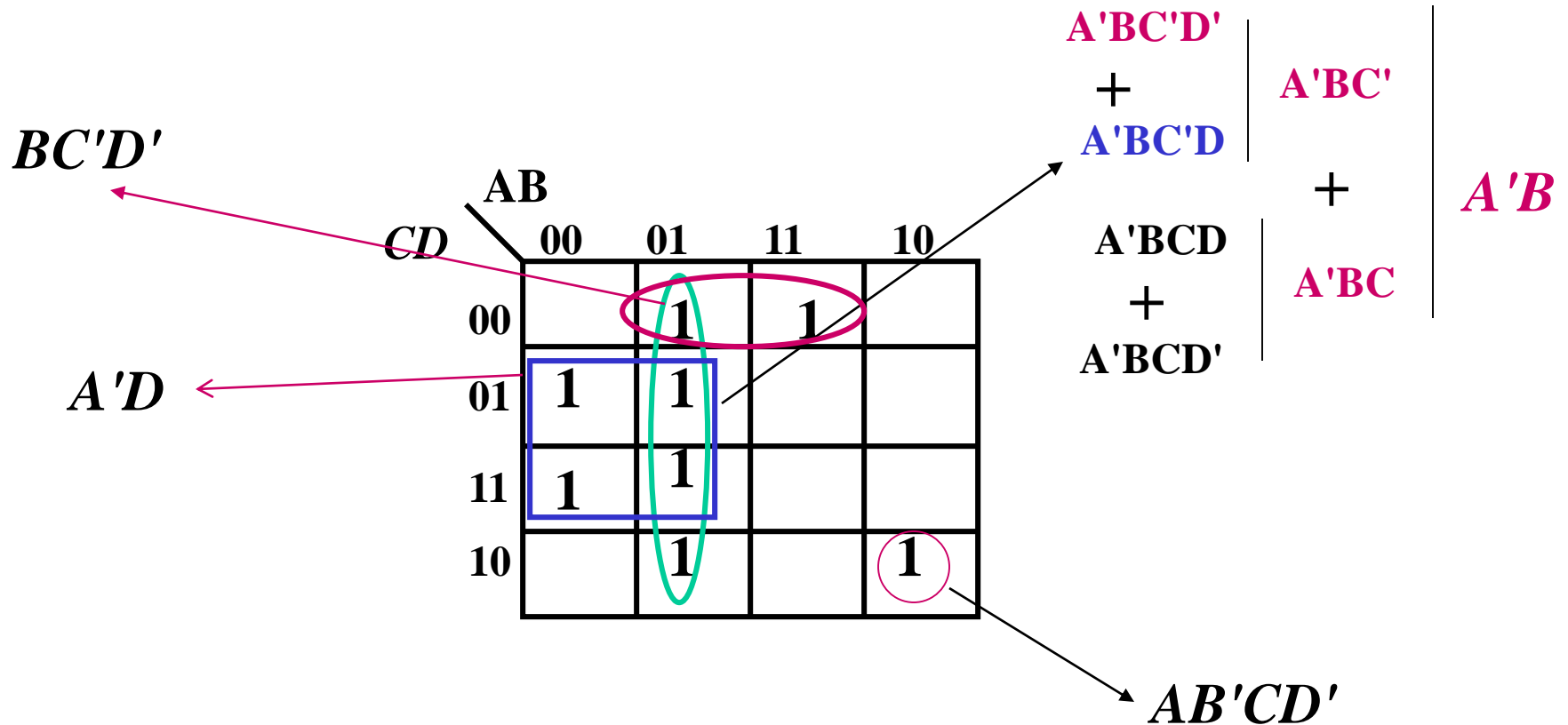
$$F(A,B,C,D) = \sum m (1, 3, 4, 5, 6, 7, 10,12)$$

*a) Building K-Map for F*

CD \ AB		A			
		00	01	11	10
C	00	0	4 <b>1</b>	12 <b>1</b>	8
	01	<b>1</b> 1	5 <b>1</b>	13	9
	11	<b>1</b> 3	7 <b>1</b>	15	11
	10	2	6 <b>1</b>	14	<b>1</b> 10
		B			

The K-Map shows the function F(A,B,C,D) with minterms 1, 3, 4, 5, 6, 7, 10, and 12 marked with a pink '1'. The map is organized with AB as columns and CD as rows. Brackets indicate the grouping of variables: A for columns, B for columns, C for rows, and D for rows.

## b) Grouping of squares



c) Write the Simplified Expression

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

a) Building K-map from the truth table

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>
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

		<i>AB</i>			
		00	01	11	10
<i>CD</i>	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

		AB			
		00	01	11	10
CD	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

A Karnaugh map for a function F of variables A, B, C, and D. The map is a 4x4 grid with columns labeled AB (00, 01, 11, 10) and rows labeled CD (00, 01, 11, 10). The cells contain values: (00,00)=1, (01,00)=1, (11,00)=X, (10,00)=1, (01,01)=1, (11,01)=X, (11,11)=X, (10,11)=X, (11,10)=X, (10,10)=X. A green oval groups the 1s in the CD=00 row, labeled C'D'. A pink square groups the 1s in the AB=01 column, labeled BC'.

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

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

# 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 $F$

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

Essential Prime  
Implicant

Essential Prime  
Implicant

Optional Prime  
Implicant

Function made up  
of only essential  
prime implicants

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

## b) Obtain Sum of Products for $F$

cd \ ab	00	01	11	10
00	$m_0$ 0	$m_1$ 1	$m_3$ 1	$m_2$ 0
01	$m_4$ 1	$m_5$ 1	$m_7$ 0	$m_6$ 0
11	$m_{12}$ 1	$m_{13}$ 1	$m_{15}$ 1	$m_{14}$ 1
10	$m_8$ 0	$m_9$ 0	$m_{11}$ 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 $F$

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

- 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.

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

## b) Obtain Sum of Products for $F$

$ab \backslash cd$	00	01	11	10
00	$m_0$ 1	$m_1$ 1	$m_3$ 0	$m_2$ 1
01	$m_4$ 1	$m_5$ 1	$m_7$ 1	$m_6$ 1
11	$m_{12}$ 0	$m_{13}$ 0	$m_{15}$ 1	$m_{14}$ 0
10	$m_8$ 0	$m_9$ 0	$m_{11}$ 0	$m_{10}$ 1

The table shows a 4x4 Karnaugh map for variables a, b, c, and d. The rows are labeled 'ab' and the columns are labeled 'cd'. The cells contain either a '1' or a '0'. The cells containing '1' are grouped into prime implicants: three groups circled in red (essential prime implicants) and one group circled in blue (optional prime implicant). The red groups are: a vertical group of (0,0), (1,0), and (1,1); a horizontal group of (0,0), (0,1), and (1,1); and a vertical group of (0,1), (1,1), and (1,0). The blue group is a horizontal group of (0,0), (0,1), (1,1), and (1,0).

- 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

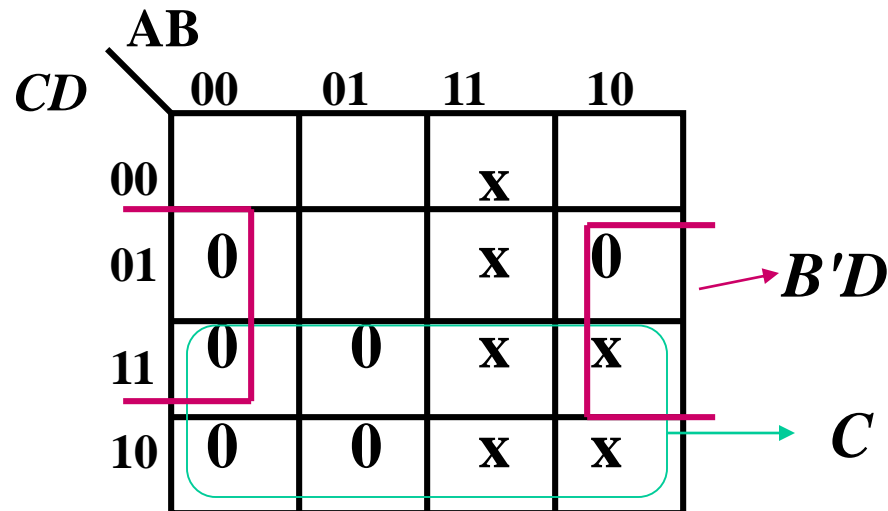
1 - use Minterms to simplify and obtain  $F'$

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

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

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

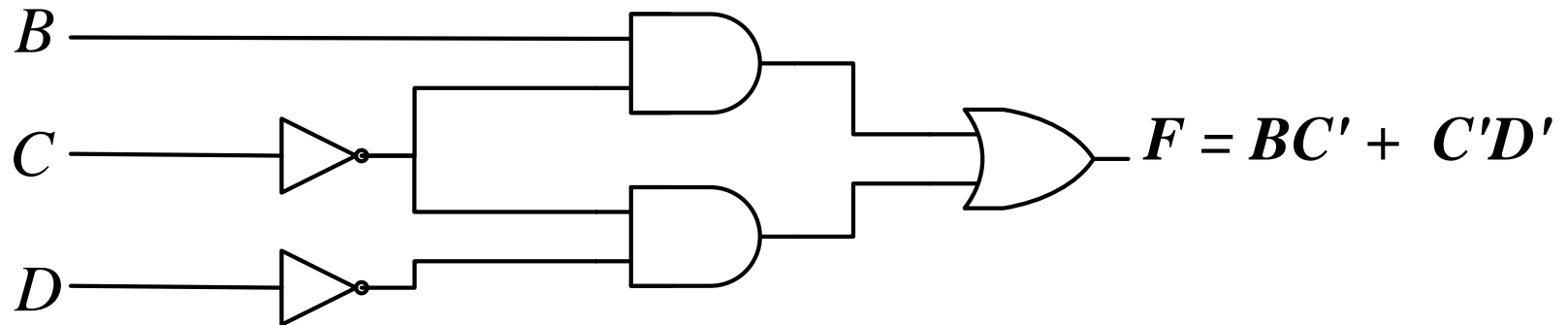
$$F = (B + D') \cdot C'$$



# Two Level Implementations

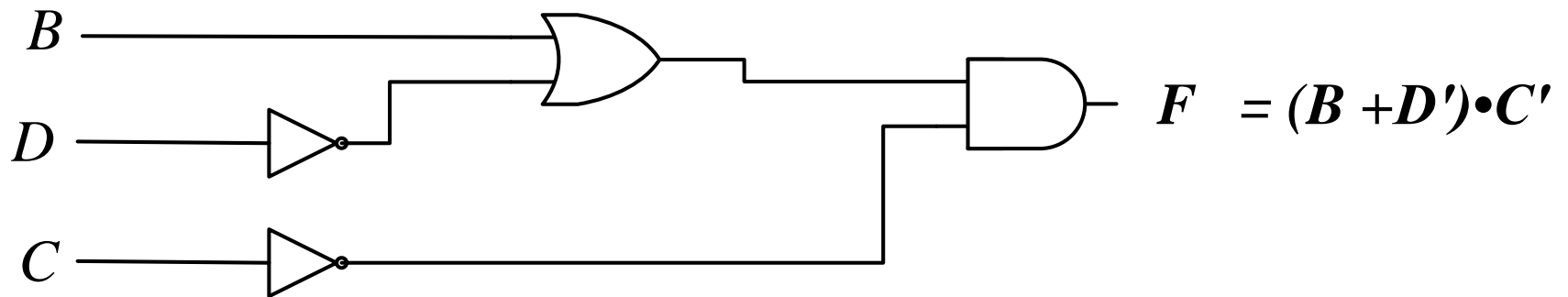
---

## *SOP : Two Level Implementation*



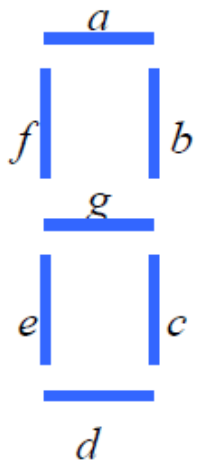
---

## *POS : Two Level Implementation*



# Seven Segment Decoder -Example

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

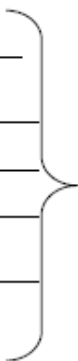


(a) Segment designation

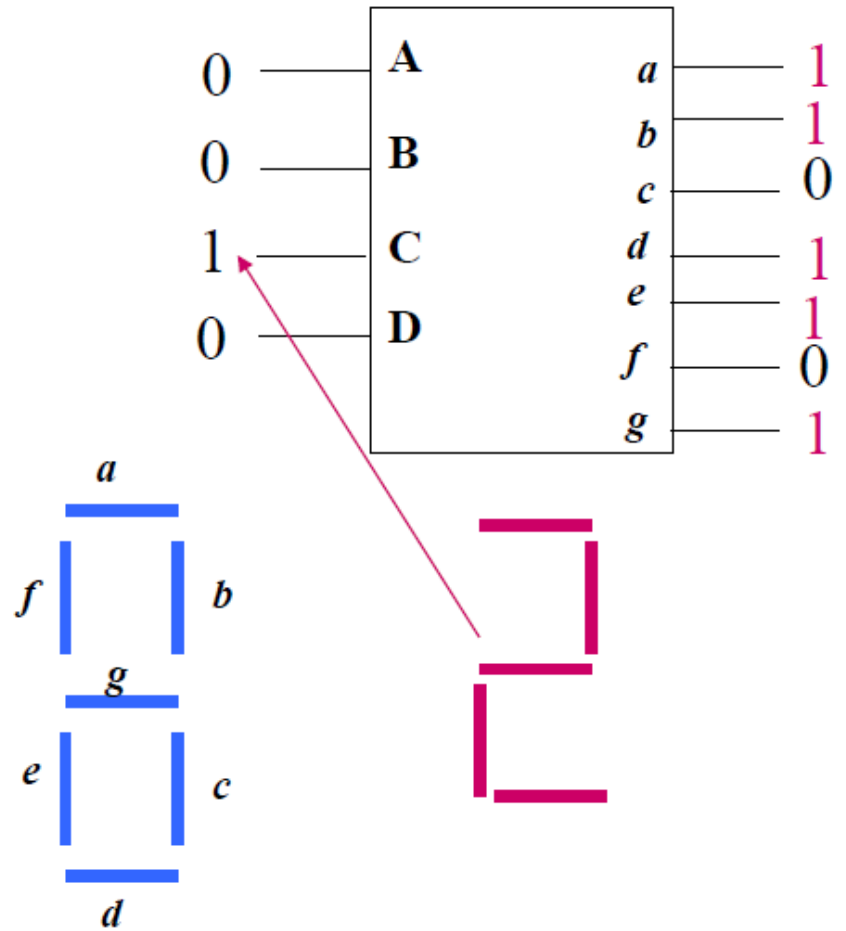
(b) Numerical designation for display

# 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

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

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

$a =$

$b =$

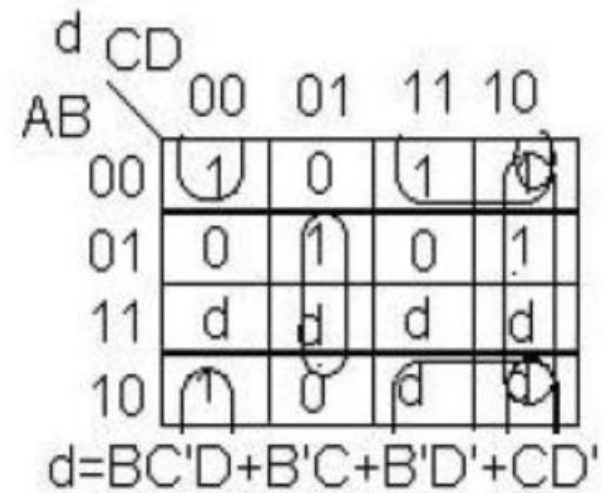
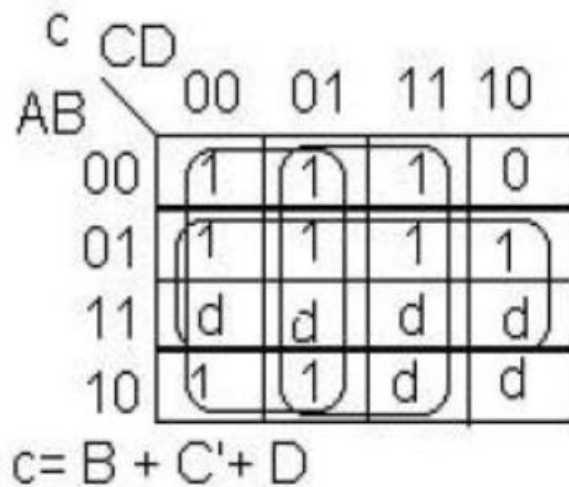
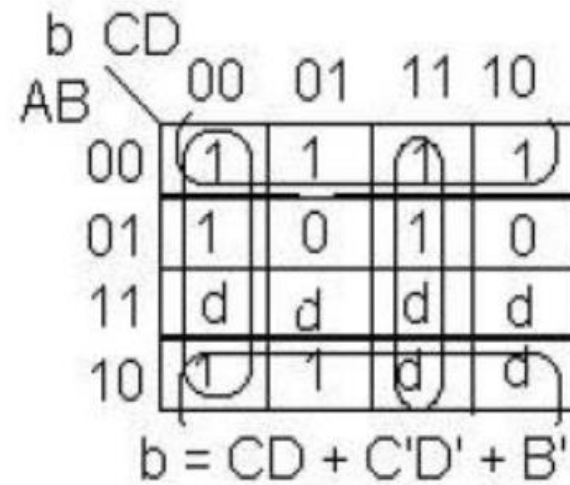
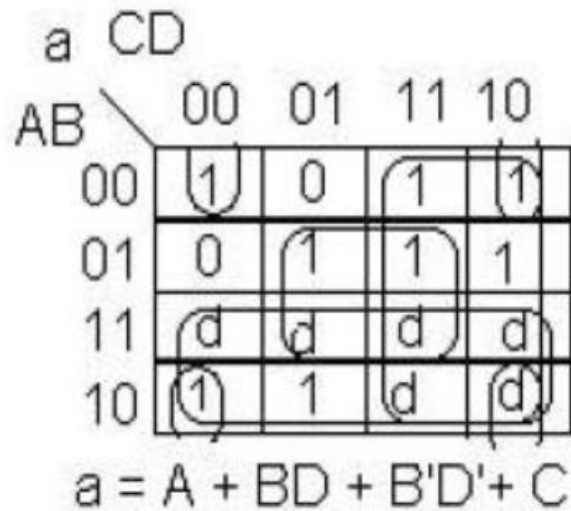
		CD			
		00	01	11	10
AB	00	1	1	1	
	01	1	1	1	1
	11				
	10	1	1		

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

$c =$

ITI1100

$d =$



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

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

# Implementations using NAND & NOR Gates

---

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

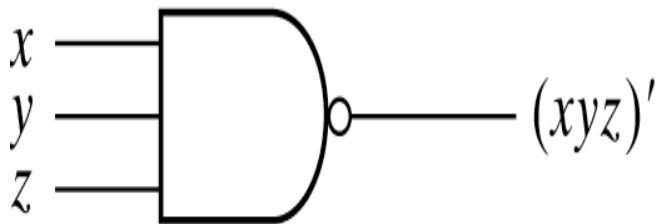
→ Both NAND and NOR gates are very valuable as any design can be realized using either one.

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

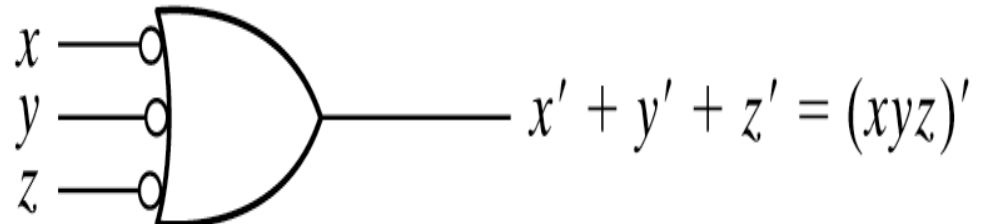
- NAND/NOR gates are typically faster and cheaper to produce.

# Logic Operations with NAND Gates

---



(a) AND-invert

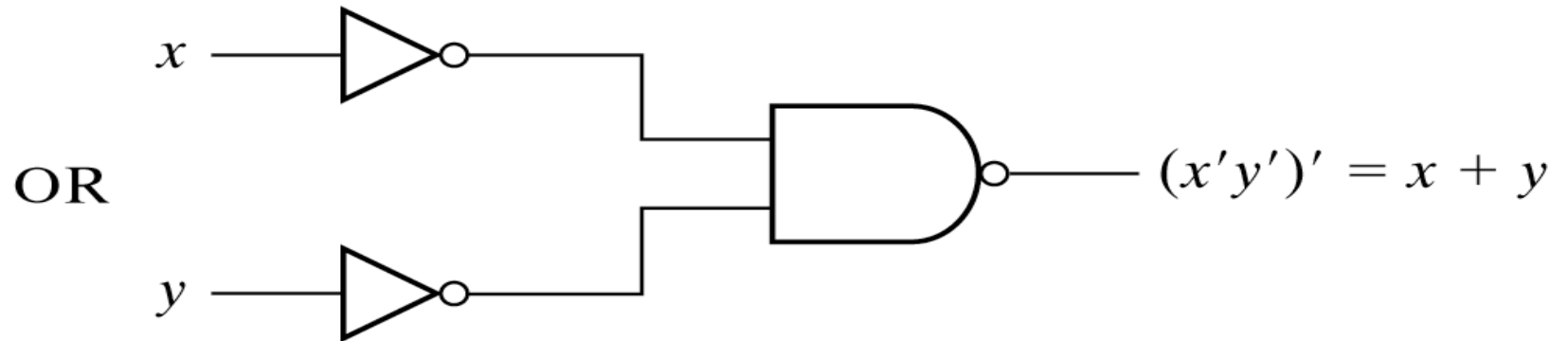
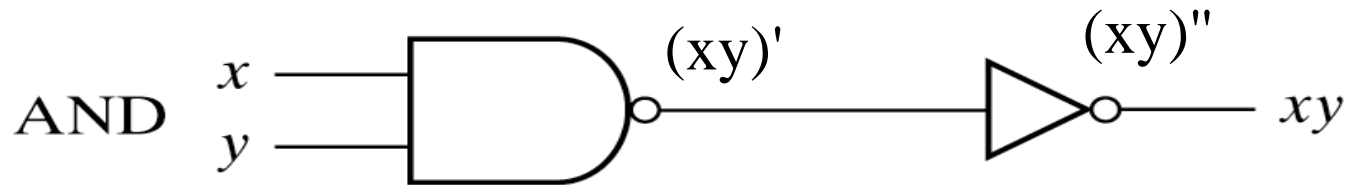
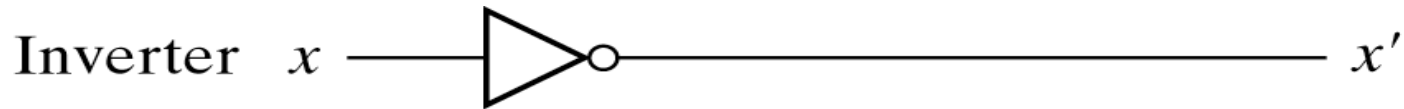


(b) Invert-OR

Two Graphic Symbols for NAND Gate

# Logic Operations with NAND Gates

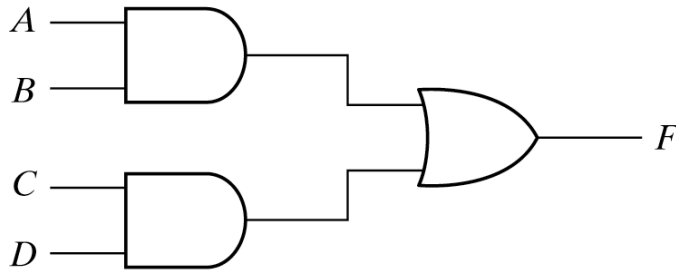
---



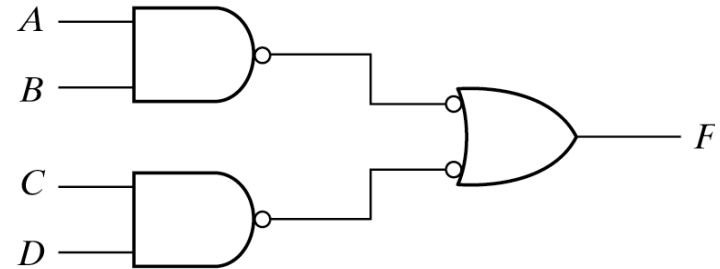
Logic Operations with NAND Gates

# NAND gates Implementations

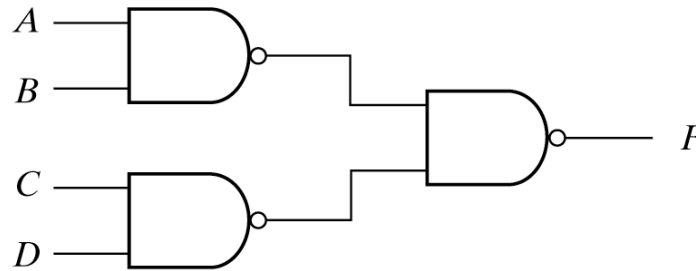
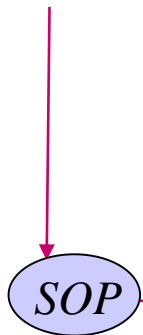
$$(AB)'' + (CD)'' = ((AB)'(CD)')$$



a) Two level with AND-OR



b) Two level with NAND & Invert-OR



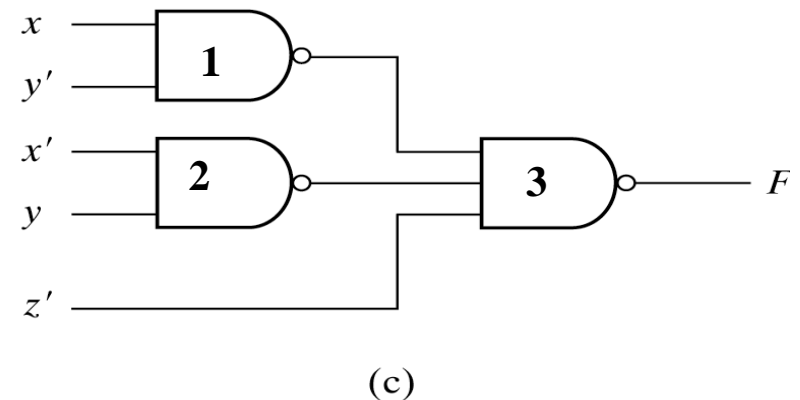
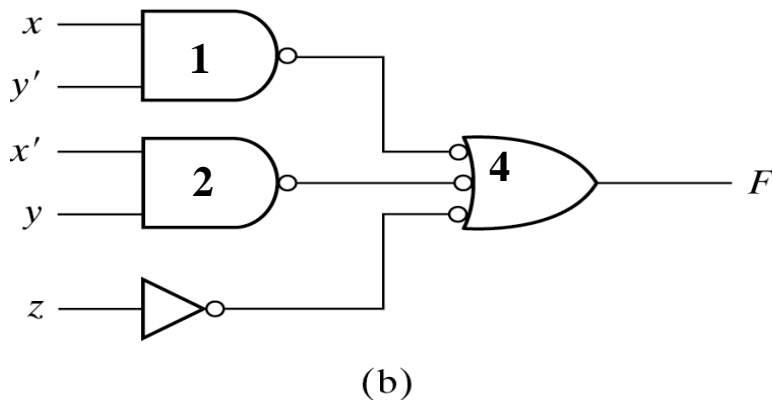
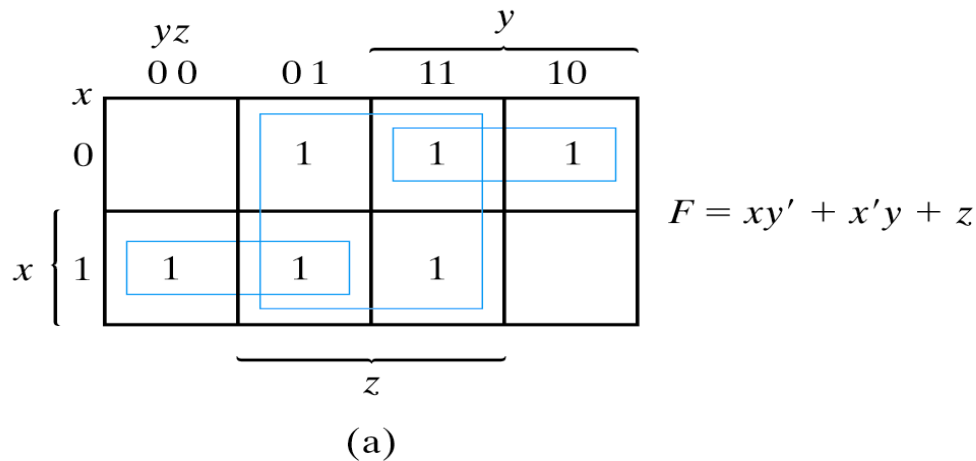
c) Two level with NAND gates  
(use this in exam)

Three Ways to Implement  $F = AB + CD$

# NAND gates implementations -Examples

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

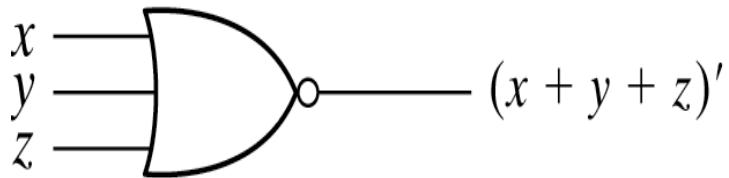
**3:**  $((xy)'(x'y)'(z'))' = (xy)'' + (x'y)'' + (z)'$  =  $xy' + x'y + z$



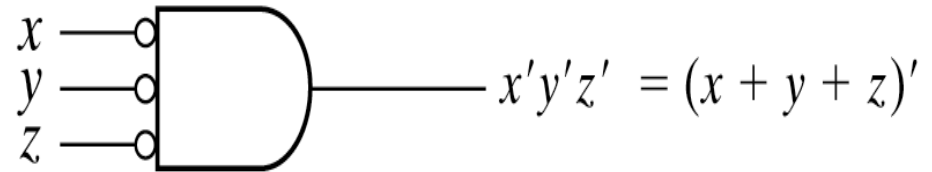
Using NAND gates to implement SOP

# Logic Operations with NOR Gates

---



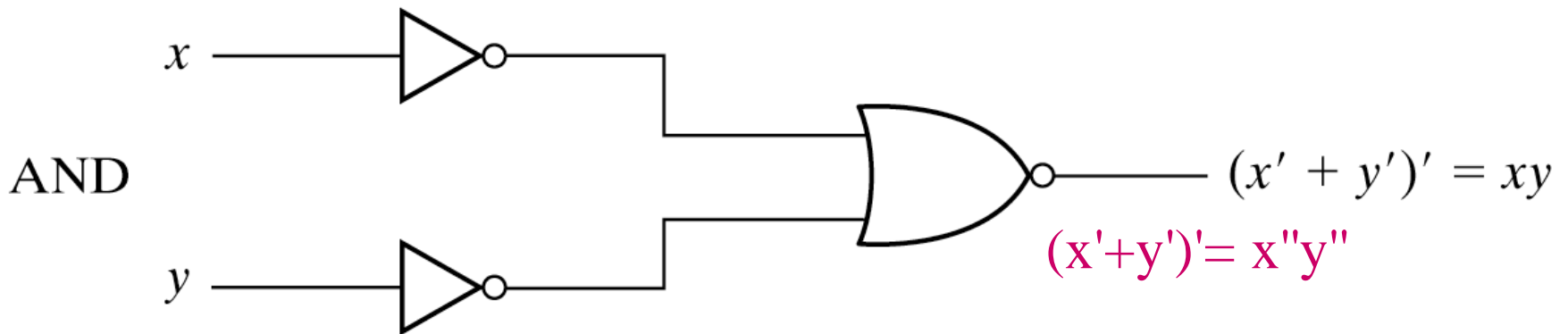
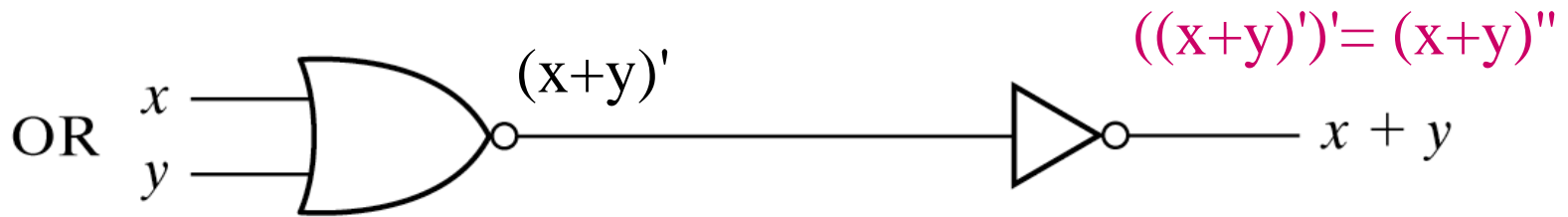
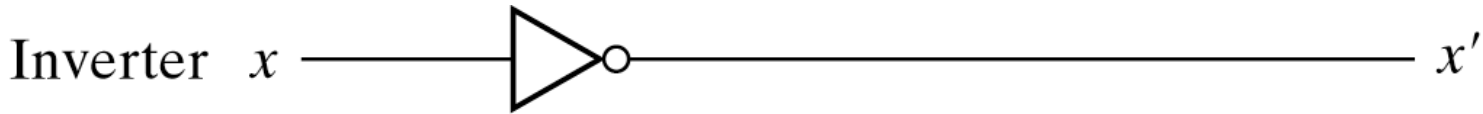
(a) OR-invert



(a) Invert-AND

Two Graphic Symbols for NOR Gate

# Logic Operations with NOR Gates

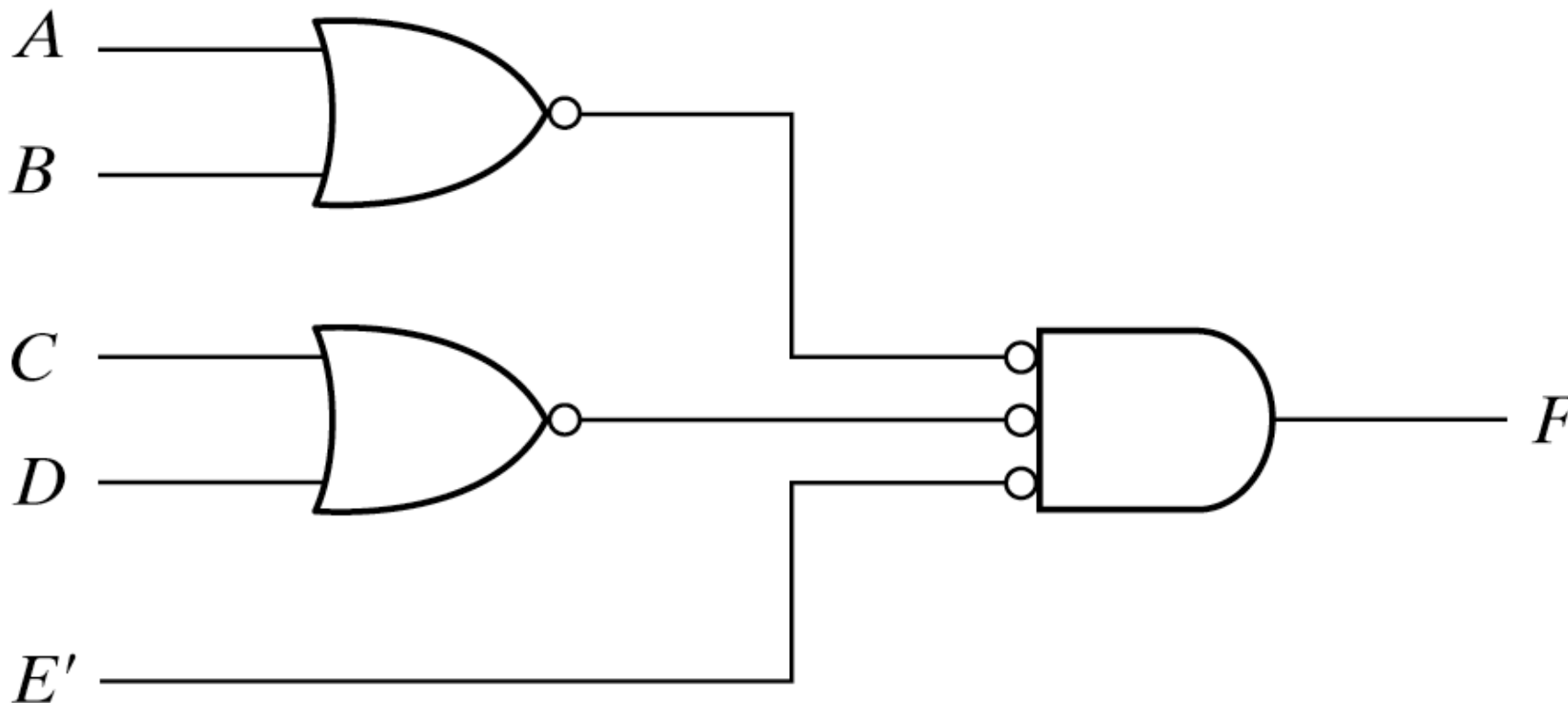


Logic Operations with NOR Gates

# NOR gates Implementation -Examples

---

POS with NOR



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

# Other implementation examples

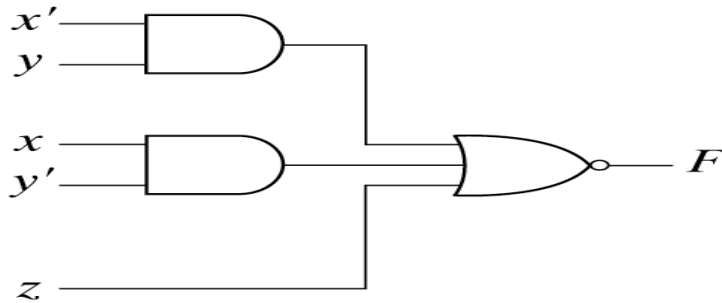
		$yz$		$y$	
		00	01	11	10
$x$	$x$	0	1	1	0
	$x'$	1	0	0	1

$\underbrace{\hspace{10em}}_z$

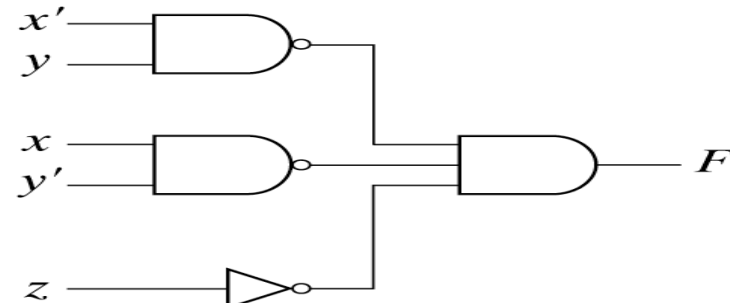
$$F = x'y'z' + xyz'$$

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

(a) Map simplification in sum of products.

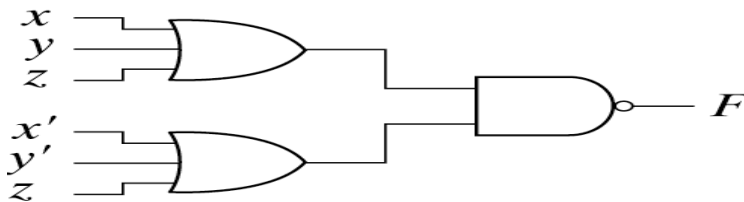


AND-NOR

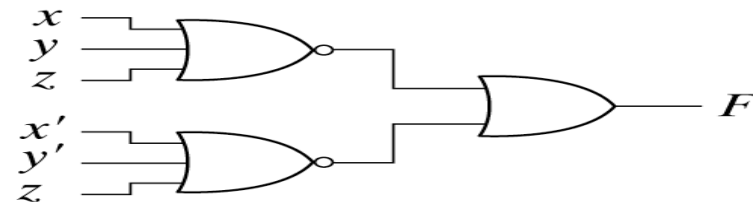


NAND-AND

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



OR-NAND



NOR-OR

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

## Other Two-level Implementations

1111100