Digital Systems & Computer Architecture

SOFE2250U

Lecture 03

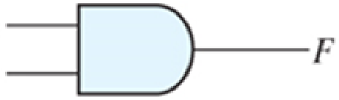
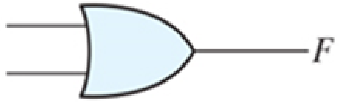
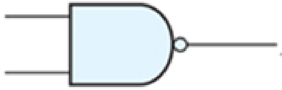
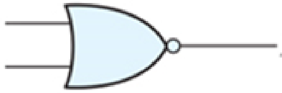
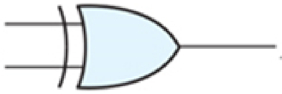
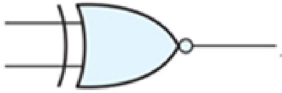
Gate-Level Minimization

Dr. Khalid A. Hafeez

Outline

- Review
- Combinational Gates
- The Map Method

Review -Logic Gates

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Summary of Rules

- Rule1
  - $x + y = y + x$
  - $xy = yx$
- Rule2
  - $x + (y + z) = (x + y) + z$
  - $x(yz) = (xy)z$
- Rule3
  - $x(y + z) = xy + xz$
  - $x + (yz) = (x + y)(x + z)$
  - $(x + y)(z + w) = xz + xw + yz + yw$
  - $xy + zw = (x + z)(x + w)(y + z)(y + w)$
- Rule4:
  - $x \bullet 0 = 0$
  - $x \bullet 1 = x$
- Rule5
  - $x+0=x$
  - $x+1=1$

- Rule6
  - $x + x = x$
  - $x \bullet x = x$
- Rule7
  - $x \bullet x' = 0$
  - $x + x' = 1$
- Rule8:
  - $x = (x')'$
- Rule9
  - $(x + y)' = x'y'$
  - $(xy)' = x' + y'$
- Rule10
  - $x + xy = x$
  - $x(x + y) = x$
- Rule11
  - $x + x'y = x + y$
  - $x' + xy = x' + y$

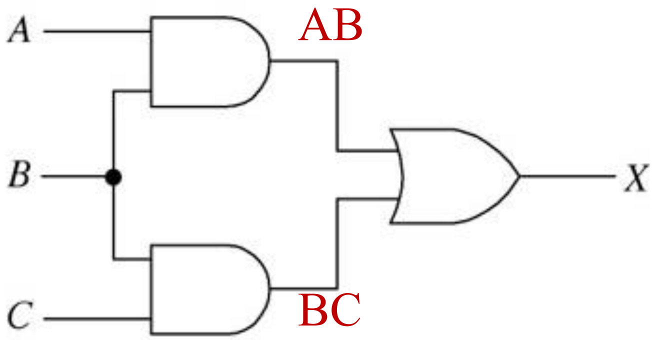
Introduction

- **Gate-level minimization**
  - refers to the design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit.

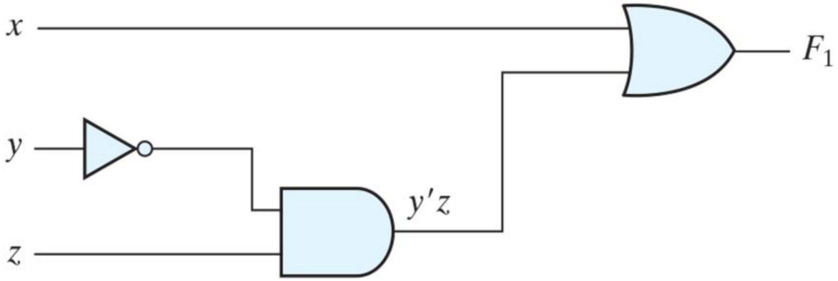
- **Combinational Logic**
  - Using two or more logic gates to perform a more useful, complex function
  - A digital circuit built from gates is called a combinational logic circuit.
  - The output of a combinational circuit depends on the combination of inputs.
  - The three basic Boolean operations (OR, AND, NOT) can describe any logic circuit.
  - We can describe a logic circuit through logic diagrams, or through Boolean expressions.
    - You should be able to convert one from the other.

Describing Logic Circuits Algebraically

- Example: Describe the Logic Circuit (Logic diagram) Algebraically and analyze that using truth table



$=AB + BC$



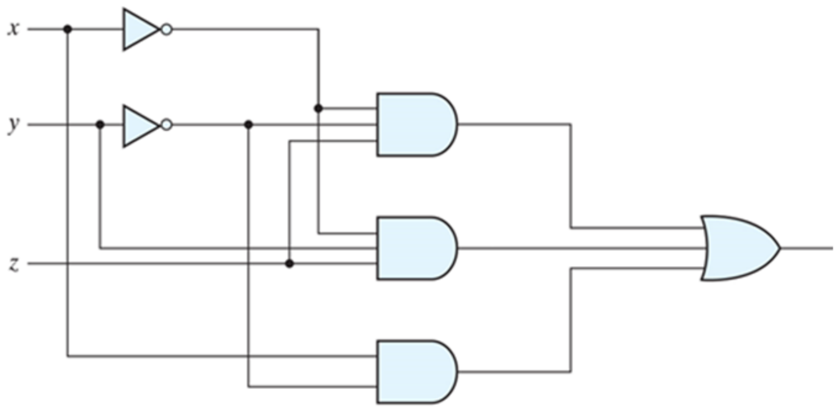
$=x + y'z$

x	y	z	F1
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

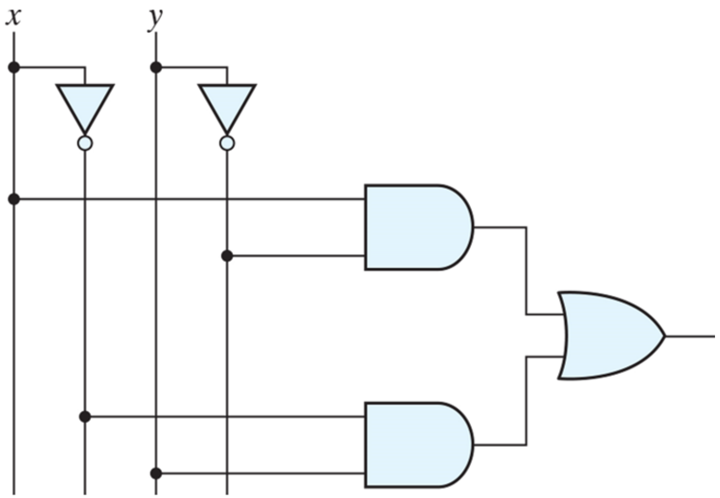
Describing Logic Circuits Algebraically

- Example: Describe the Logic Circuit (Logic diagram) Algebraically and analyze that using truth table



F1=$x'y'z + x'yz + xy'$

x	y	z	F1
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

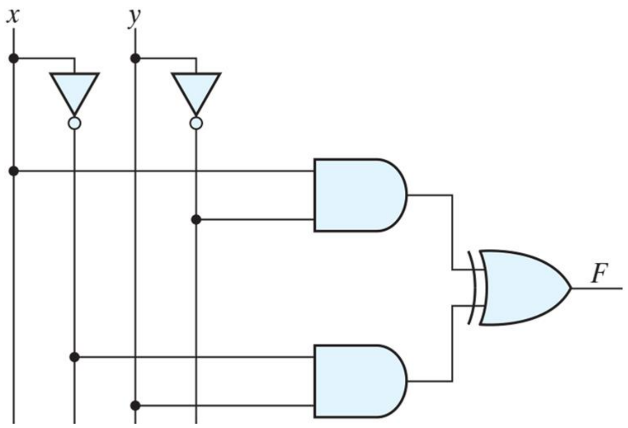


F2=$xy' + x'y$

x	y	F2
0	0	
0	1	
1	0	
1	1	

Describing Logic Circuits Algebraically

- Example: Describe the Logic Circuit (Logic diagram) Algebraically and analyze that using truth table



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

$= (xy').(x'y)' + (xy')'.(x'y)$

x	y	xy'	$x'y$	F
0	0			
0	1			
1	0			
1	1			

Simplification of Combinational Logic Circuits

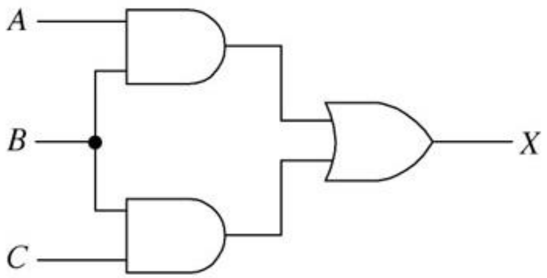
- Equivalent circuits can be formed with fewer gates
  - Cost is reduced
  - Reliability is improved
- **Two methods can be used to simplify a logic circuit:**
  - Boolean algebra theorems;
  - A mapping technique.

- The methods of logic-circuit simplification that we will study require the logic expression to be in a sum-of-products (SOP) form.
- A Sum-of-products (SOP) expression will appear as two or more AND terms ORed together.
- Therefore, AND-OR-INVERT Gates are required for Implementing Sum-of-Products Expressions
  - $ABC + ABC$
  - $AB + ABC + CD + D$

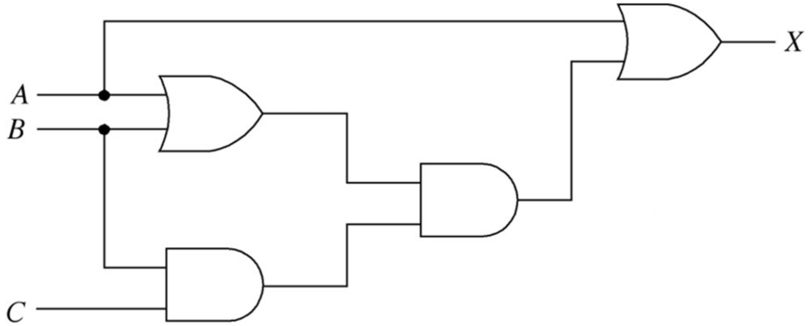
Simplification of Combinational Logic Circuits: Using Boolean Algebra

1. Put your expression into SOP form using standard Boolean algebras and DeMorgan's Laws.
2. Once in SOP form, look for common factors to further reduce your expression.

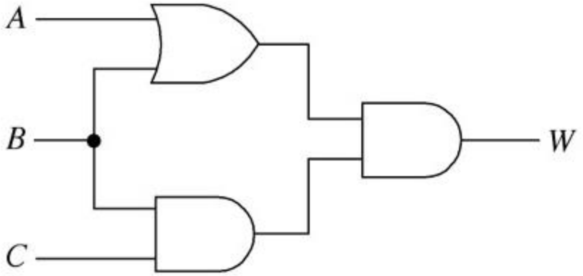
- **Examples:** Simplify the logic circuit shown.



$=AB + BC \ = B(A+C)$



$=A + BC(A+B) \ = A + ABC + BC = A + BC$



$=(A+B)BC \ = ABC + BC = BC(A+1) = BC$

Simplification of Combinational Logic Circuits: Using Boolean Algebra

- **Simplify the following expressions:**

1. $A\bar{B}D + A\bar{B}\bar{D}$

2. $(\bar{A} + B)(A + B)$

3. $A(CD) + \bar{A}B(CD)$

4. $\overline{x + y + z}$

5. $\overline{(A\,\bar{B}) + C}$

6. $\overline{(\bar{A} + C)(B + \bar{D})}$

Minterms

x	y	z	Term	Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

- Examples

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$f_1 = x'y'z + xy'z' + xyz$

$f_2 = x'yz + xy'z + xyz' + xyz$

How to design a logic circuit?

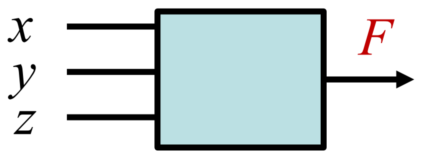
- **Design procedure**

1. Interpret the problem, determine the number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the relationship between inputs and outputs.
3. Obtain the Boolean function for each output as a function of input variables by writing the AND (product) term for each case where the output equals 1, then combine the terms in SOP form.
4. Simplify the function if possible.
5. Draw the logic circuit.

How to design a logic circuit?

- **Design Example 1**
  - There are three judges, each have a push-button and would push the button whenever they would like to vote. Design a logic circuit that turns on a light when the majority of the judges vote.

1. How to describe the problem
2. Truth table:
3. Simplify the function
4. Implementation

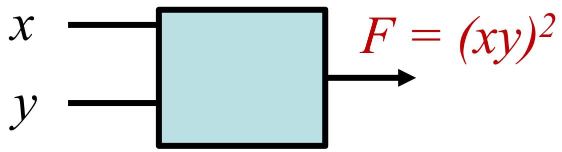


x	*y*	*z*	*F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$
\begin{aligned}
F &= x'yz + xy'z + xyz' + xyz \\
&= x'yz + xy'z + xy \\
&= x'yz + x(y'z + y) \\
&= x'yz + x(y+y')(y+z) \\
&= x'yz + xy + xz \\
&= y(x'z + x) + xz \\
&= y(x+z)(x+x') + xz \\
&= xy + yz + xz
\end{aligned}
$$

- **Design Example 2:** Design a circuit to obtain the square of a two-digit binary number

   1. Interpreting

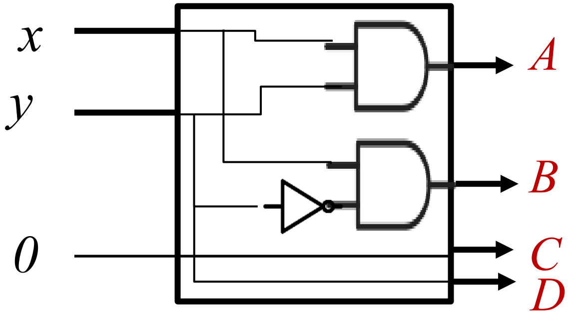


   2. Truth table

x	**y**	**F**	**ABCD**
0	0	$0^2= 0$	0000
0	1	$1^2= 1$	0001
1	0	$2^2= 4$	0100
1	1	$3^2= 9$	1001

F needs 4 digits

3. Boolean Function $A=xy$ $B= xy'$ $C= 0$ $D=x'y + xy=y(x+x') = y$
4. Simplifying
5. Implementation



The Map Method

- Logic minimization
  - Algebraic approaches: lack specific rules
  - The Karnaugh map (**K-Map**)
    - A simple straight forward procedure
    - A pictorial form of a truth table
    - Applicable if the number of variables < 7
- A diagram made up of squares, where each square represents one minterm

- Boolean function
  - It is the sum of minterms
  - Or it is the sum of products (or product of sum) in the simplest form
  - Use the minimum number of terms and the minimum number of literals

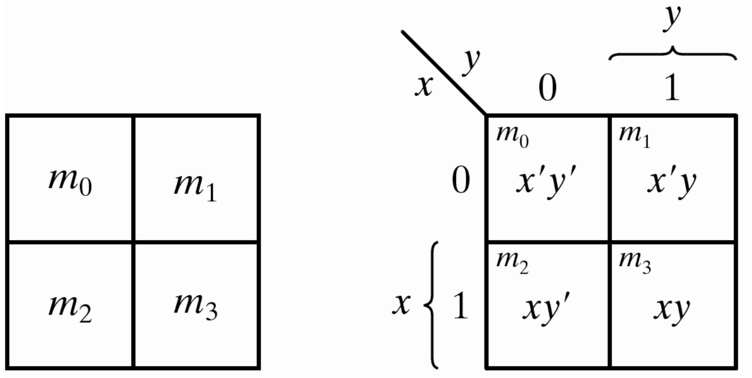
- Note: The simplified expression may not be unique

Two-Variable Map

- A two-variable map
  - Four minterms
  - x' = row 0; x = row 1
  - y' = column 0; y = column 1

x	y	**Designation**
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

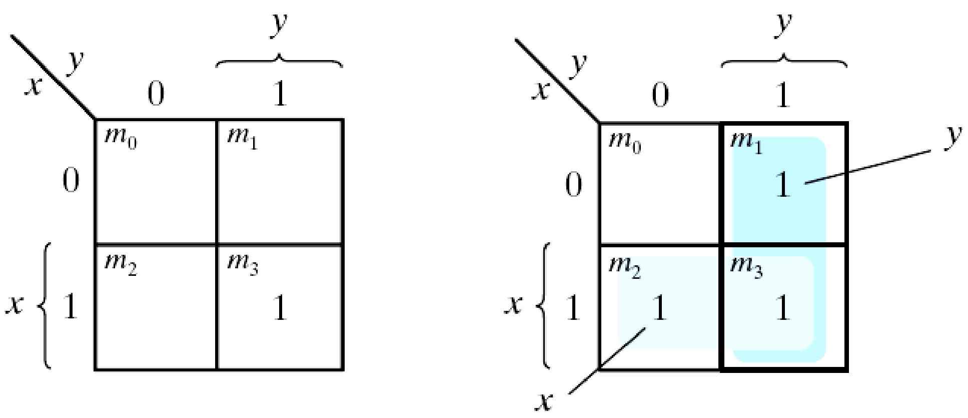
  - A truth table in square diagram



(a) (b)

Figure 3.1 Two-variable Map

- Examples:
  - Fig. 3.2(a): $xy = m_3$
  - Fig. 3.2(b): $x+y = x'y+xy' +xy = m_1+m_2+m_3$



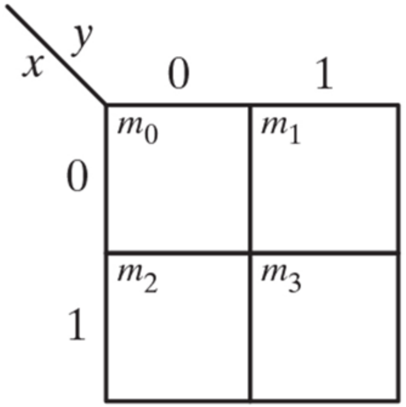
(a) xy (b) $x + y$

Figure 3.2 Representation of functions in the map

Two-Variable Map

- Example:

x	y	
0	0	
0	1	
1	0	
1	1	



Karnaugh Mapping (K-Map)

- **The simplification process:**
  1. Construct a K-map. Place the appropriate 1's and 0's.
  2. Look for adjacent 1's. Connect them with a loop.
     - Loops must encircle groups of 2^m 1's.
     - Make your loops as large as possible.
     - Loops are allowed to wrap around the sides/edges.
     - Cover the 1's with the MINIMUM number of loops.
  3. Use these loops to define a minimum realization in SOP form.

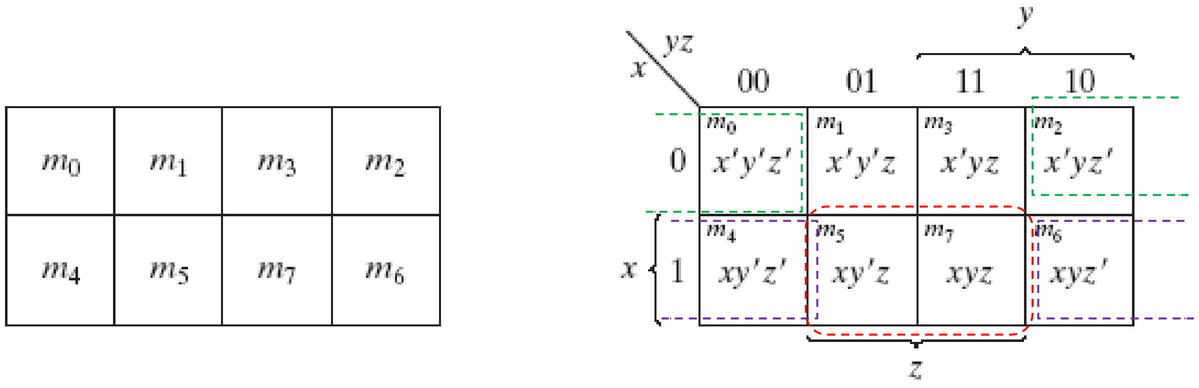
- **Don't Care Conditions:**
  - Logic circuits can have output levels that don't matter. These correspond to input levels that are unused. Entries for these DON'T CARE outputs are marked with an X.
  - When creating loops in a K-map you can choose whether or not to include these in your loops.

Three-Variable Map

- A three-variable map
  - Eight minterms
  - The Gray code sequence
  - Any two adjacent squares in the map differ by only one variable
    - Primed in one square and unprimed in the other
    - e.g., m_5 and m_7 can be simplified
    - $m_5 + m_7 = xy'z + xyz = xz\,(y'+y) = xz$

x	y	z	Designation
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

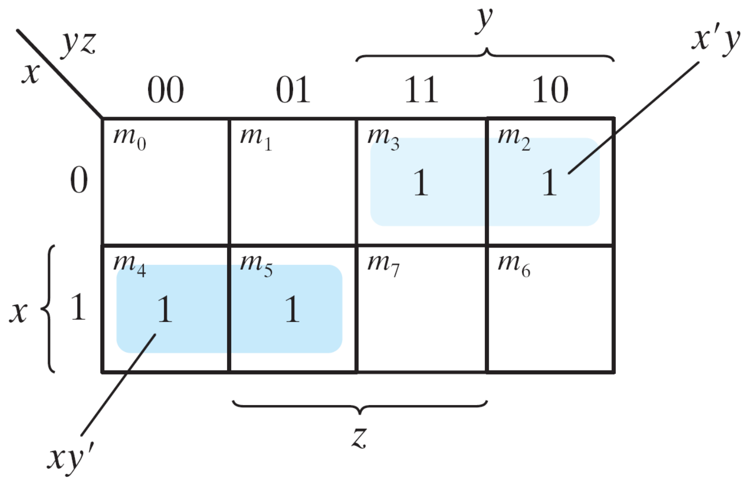
  - m_0 and m_2 (m_4 and m_6) are adjacent
  - $m_0 + m_2 = x'y'z' + x'yz' = x'z'\,(y'+y) = x'z'$
  - $m_4 + m_6 = xy'z' + xyz' = xz'\,(y'+y) = xz'$



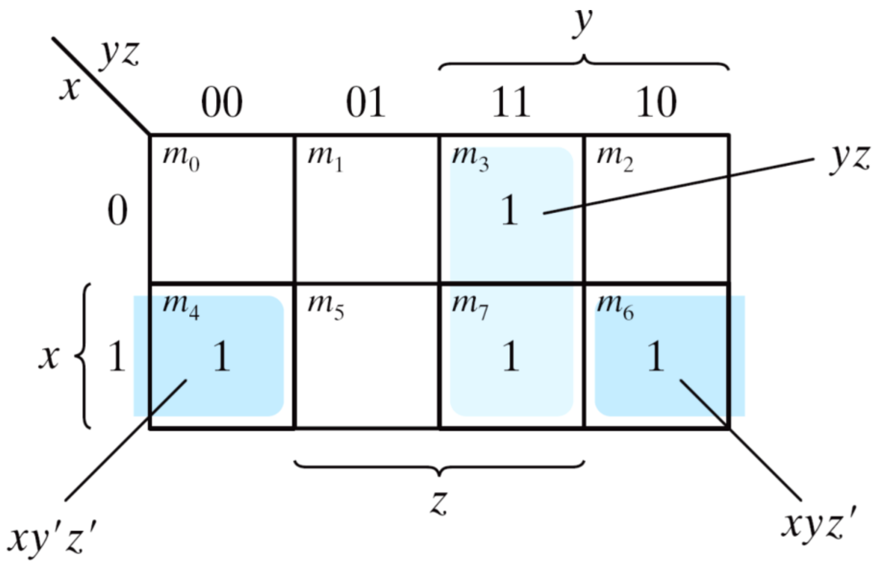
(a) Three-variable Map (b)

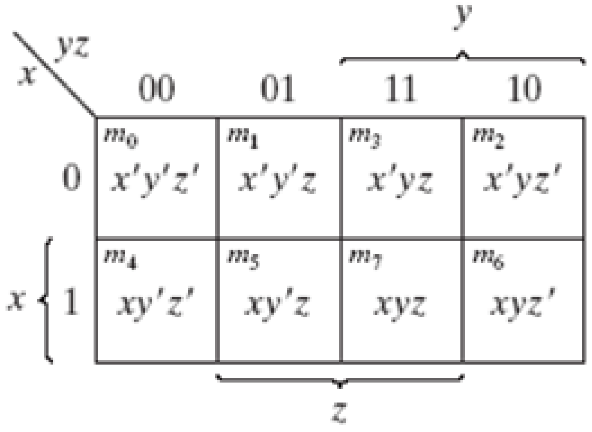
Three-Variable Map

- Example 3.1: simplify the Boolean function F(x, y, z) = Σ(2, 3, 4, 5)
  - $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$



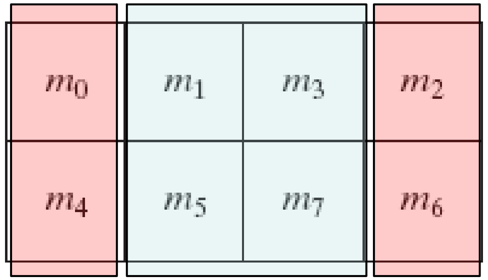
- *Example 3.2: simplify F(x, y, z) = Σ(3, 4, 6, 7)*
  - $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$



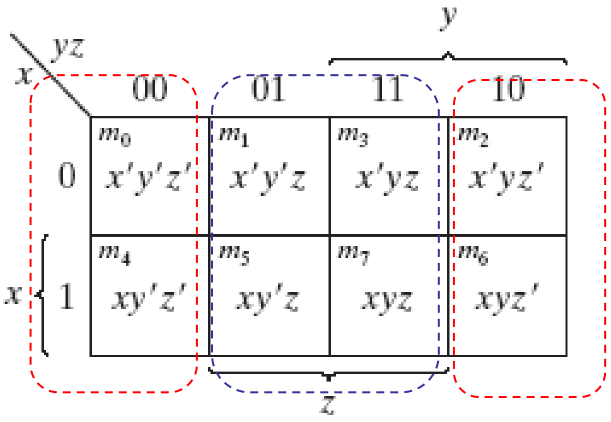


Three-Variable Map

- Consider **four** adjacent squares
  - 2, 4, and 8 squares
  - $m_0+m_2+m_4+m_6 = x'y'z'+x'yz'+xy'z'+xyz' = x'z'(y'+y) +xz'(y'+y) = x'z' + xz' = z'$
  - $m_1+m_3+m_5+m_7 = x'y'z+x'yz+xy'z+xyz = x'z(y'+y) + xz(y'+y) = x'z + xz = z$



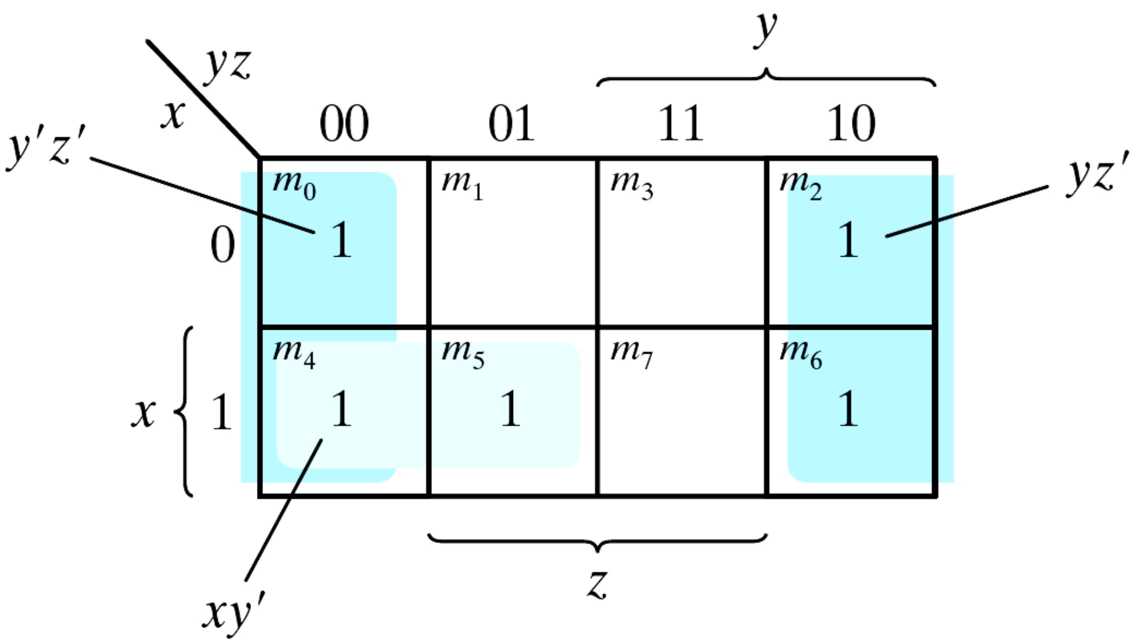
(a)



(b)

Three-Variable Map

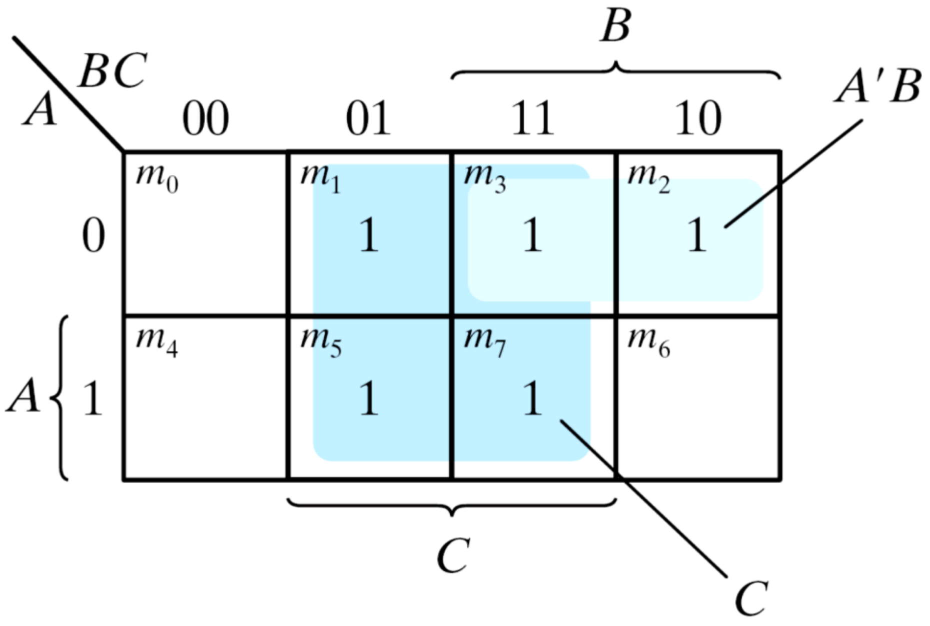
- Example 3.3: simplify $F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$
  - $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$



Note: $y'z' + yz' = z'$

Three-Variable Map

- Example 3.4: let $F = A'C + A'B + AB'C + BC$
  a) Express it in sum of minterms.
  b) Find the minimal sum of products expression.
  Ans:
  $$F(A, B, C) = \Sigma(1, 2, 3, 5, 7) = C + A'B$$



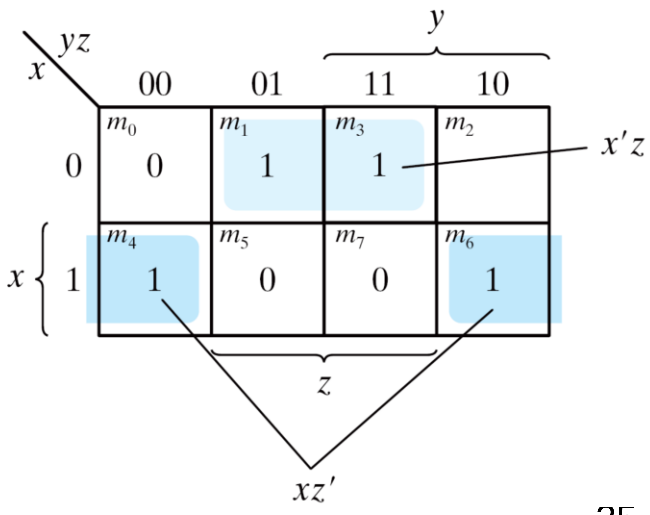
Sum-of-Minterm Procedure

- Consider the function defined in Table 3.2.
- In sum-of-minterm: $$\boxed{F(x,y,z)=\sum(1,3,4,6)}$$
- In sum-of-maxterm: $$\boxed{F'(x,y,z)=\Pi(0,2,5,7)}$$
- Taking the complement of F′ $$\boxed{F(x,y,z)=(x'+z')(x+z)}$$
  - Combine the 1’s: $$\boxed{F(x,y,z)=x'z+xz'}$$
  - Combine the 0’s : $$\boxed{F'(x,y,z)=xz+x'z'}$$

Table 3.2
Truth Table of Function F

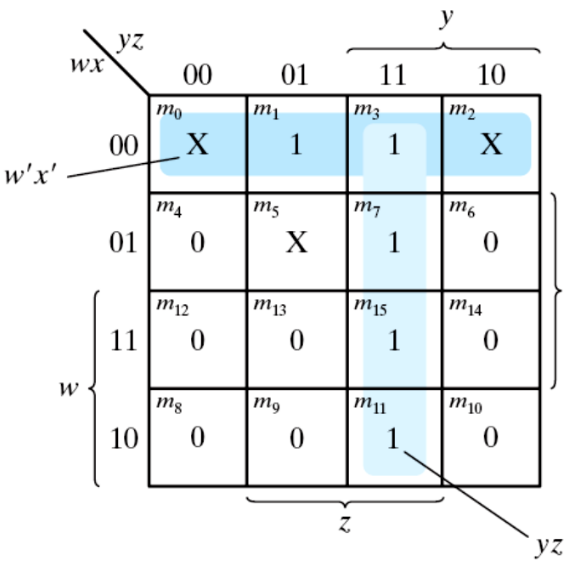
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Map for the function of Table 3.2

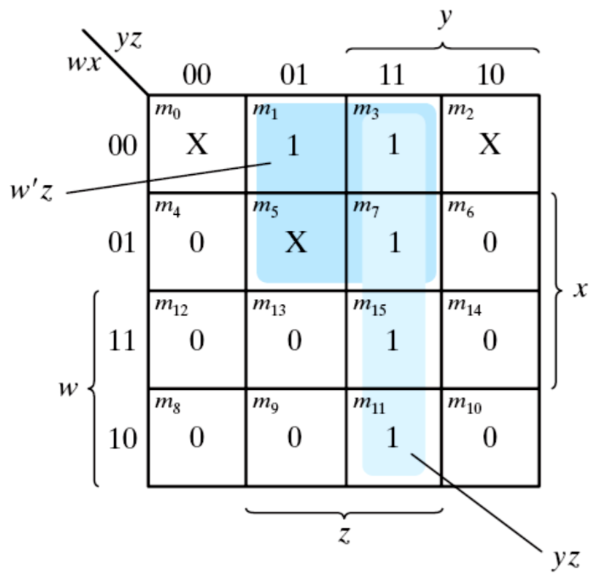


Don't-Care Conditions

- The value of a function is not specified for certain combinations of variables
- The don't-care conditions can be utilized in logic minimization
  - Can be implemented as 0 or 1
- **Example 3.9:**
  - Simplify $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$ which has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$.
  - $F = yz + w'x'$; *or* $F = yz + w'z$
  - $F = \Sigma(0, 1, 2, 3, 7, 11, 15)$; or $F = \Sigma(1, 3, 5, 7, 11, 15)$
  - Either expression is acceptable



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

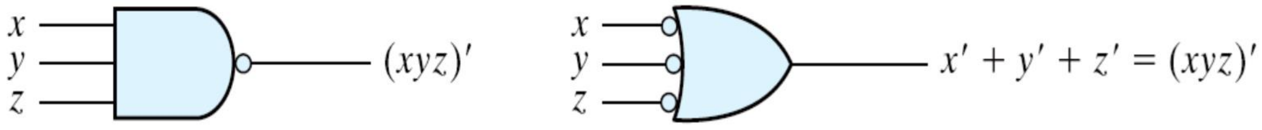


(b) $F = yz + w'z$

NAND and NOR Implementation

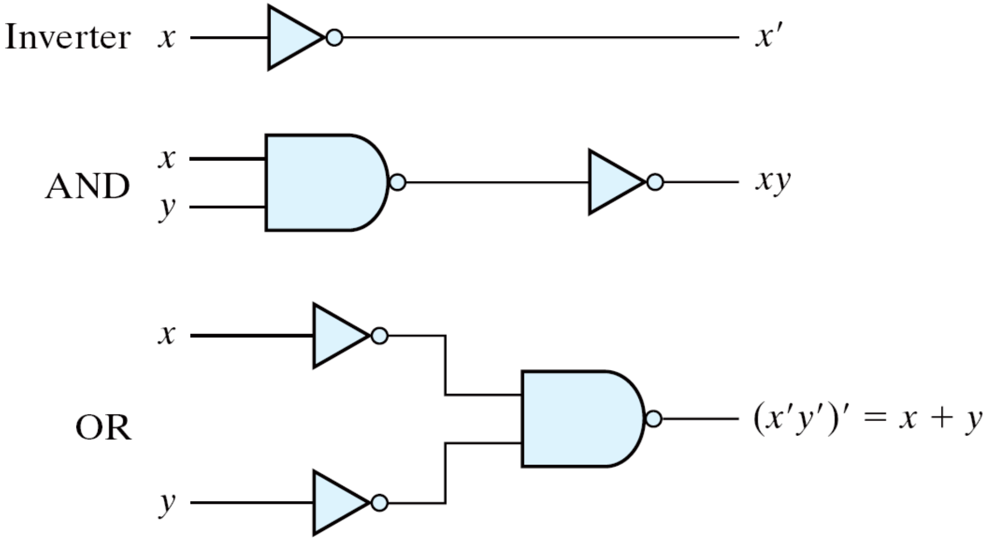
- NAND gate is a universal gate
  - Can implement any digital system

- Two graphic symbols for a NAND gate



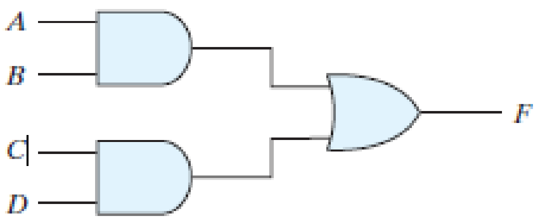
(a) AND-invert (b) Invert-OR

Logic Operations with NAND Gates

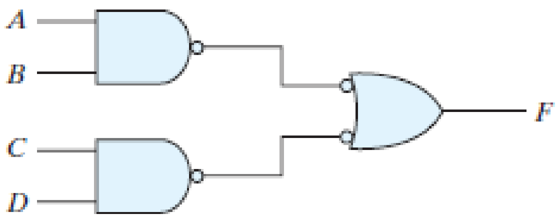


Two-level Implementation

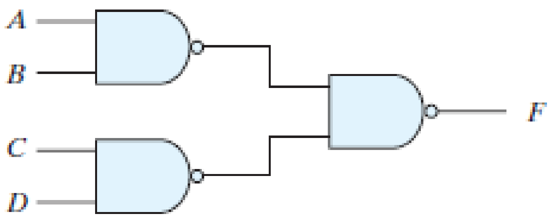
- Two-level logic
  - NAND-NAND = sum of products
  - Example: $F = AB+CD$
  - $F = ((AB)'(CD)')' = AB+CD$
    - Three ways to implement F = AB + CD



(a)



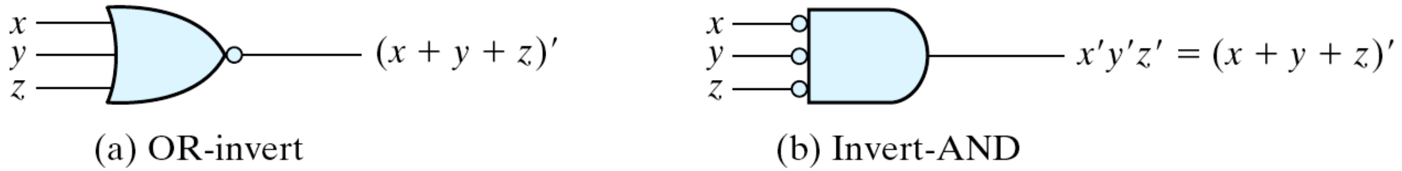
(b)



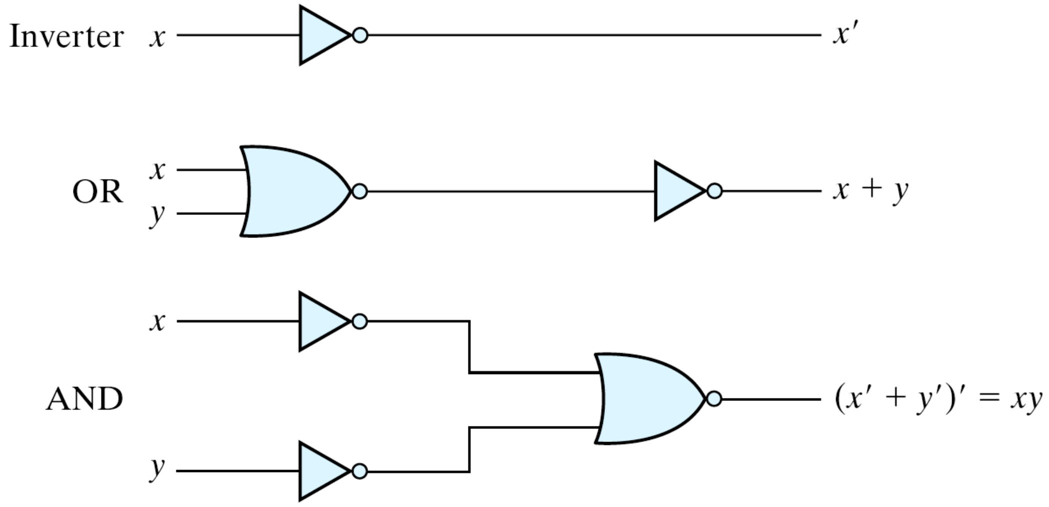
(c)

NOR Implementation

- Two Graphic Symbols for NOR Gate

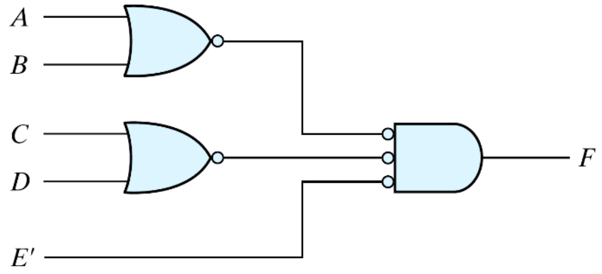


- NOR function is the dual of NAND function.
- The NOR gate is also universal.



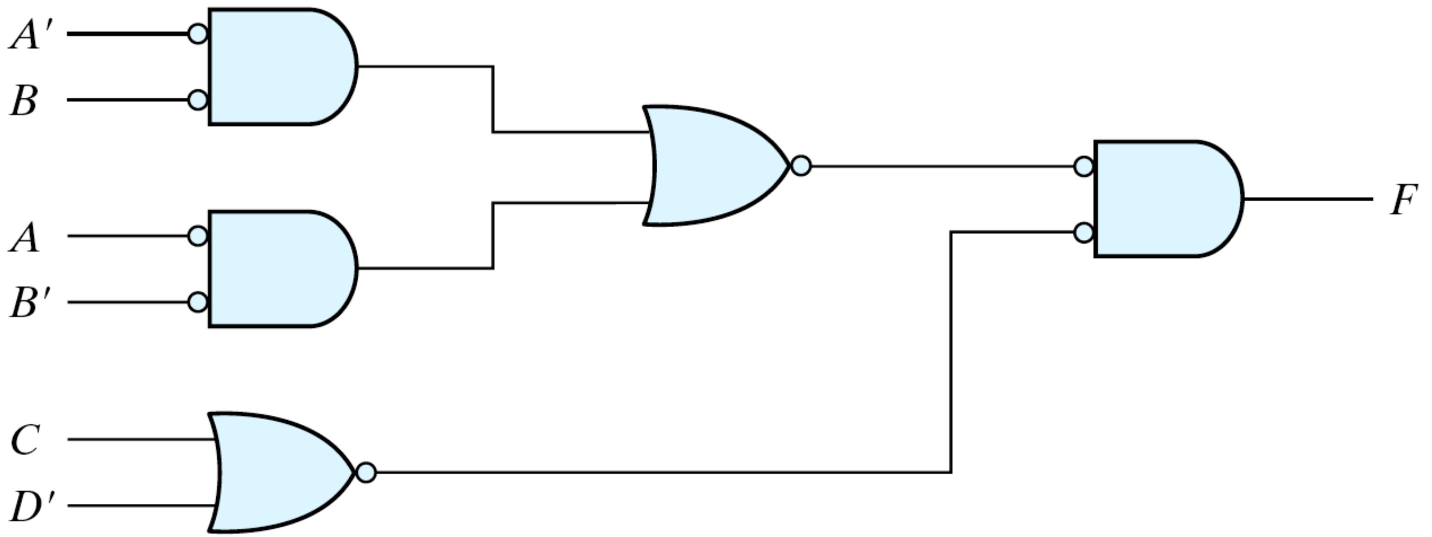
Logic Operation with NOR Gates

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



NOR Implementation

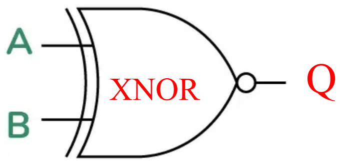
Example: Implementing $F = (AB' + A'B)(C + D')$ with NOR gates



Exclusive-OR Function

- Exclusive-OR (XOR)
  - $x \oplus y = xy' + x'y$
- Exclusive-NOR (XNOR)
  - $(x \oplus y)' = xy + x'y'$
- Some identities
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = (x \oplus y)'$
  - $x' \oplus y = (x \oplus y)'$
- Commutative and associative
  - $A \oplus B = B \oplus A$
  - $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

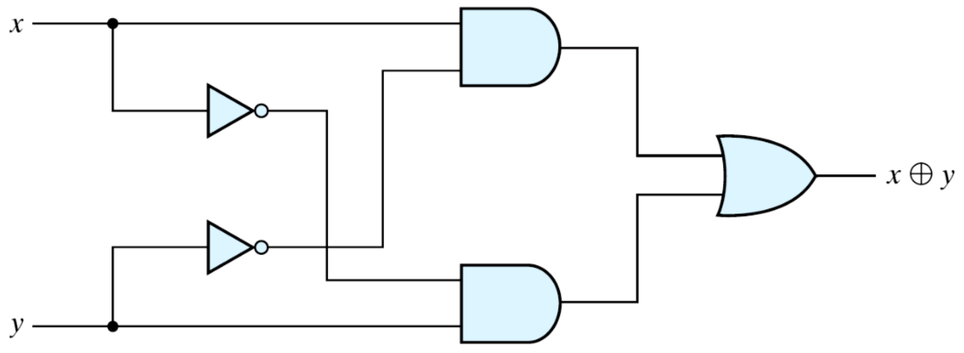


Truth Table

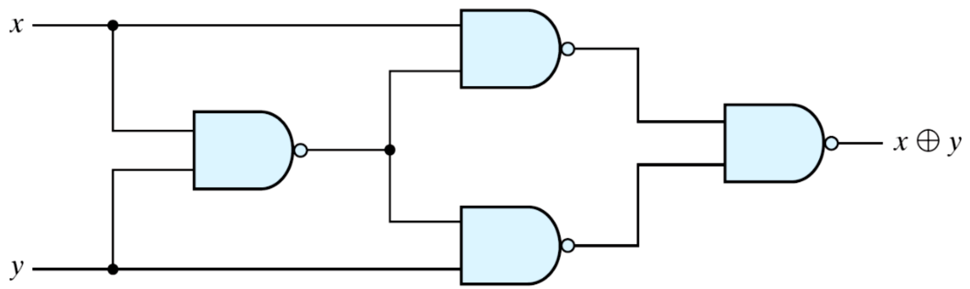
Input A	Input B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Exclusive-OR Implementations

- Exclusive-OR Implementations
  - $(x'+y')x + (x'+y')y = xy'+x'y = x \oplus y$



(a) With AND–OR–NOT gates



(b) With NAND gates

Odd Function

- $A \oplus B \oplus C = (AB'+A'B)C' + (AB+A'B')C = AB'C'+A'BC'+ABC+A'B'C = \Sigma(1, 2, 4, 7)$
- **XOR** is an odd function → an odd number of 1's, then $F = 1$.
- **XNOR** is an even function → an even number of 1's, then $F = 1$.

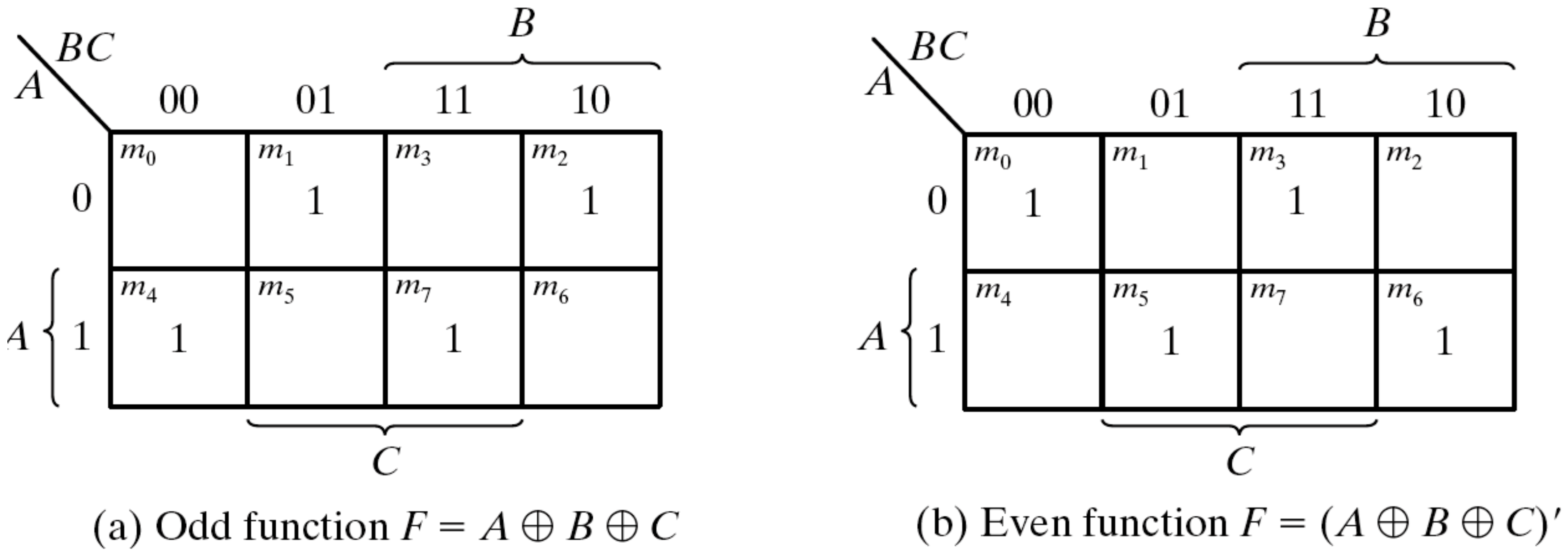
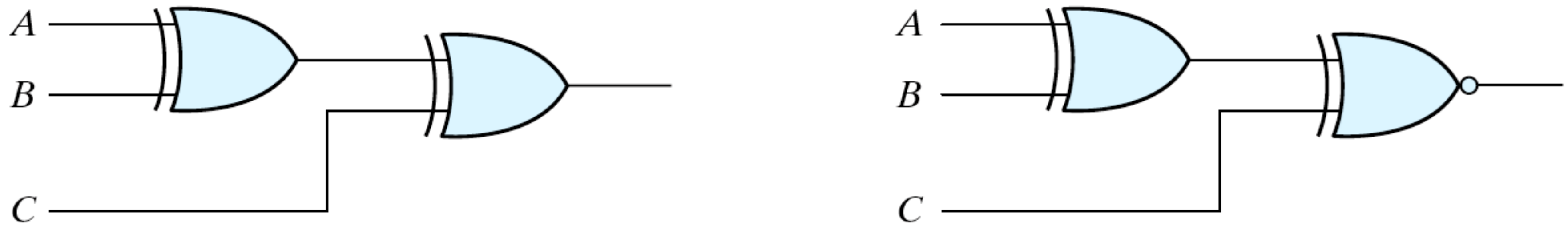


Figure 3.33 Map for a Three-variable Exclusive-OR Function

XOR and XNOR

- Logic diagram of odd and even functions



(a) 3-input odd function

(b) 3-input even function