Gate-Level Minimization

Chun-Jen Tsai
National Chiao Tung University
10/11/2012

Gate-Level Minimization

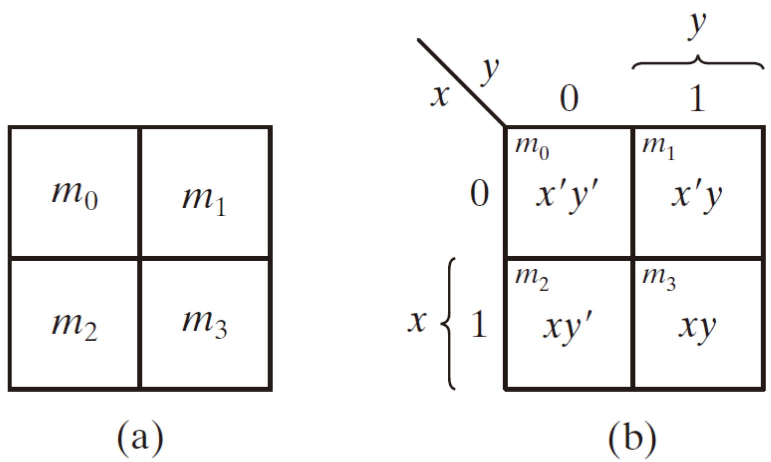
- The design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit.
- Logic minimization approaches:
  - Algebraic approaches: lack specific rules
  - The Karnaugh map (or K-map):
    - a pictorial form of a truth table of a 2-D array of squares
    - each square represents one minterm
    - Certain patterns of adjacent squares have an equivalent (simpler) Boolean expression

Goal of Minimization

- We assume that the simplest algebraic expression is an algebraic expression with:
  - A minimum number of terms
  - The smallest number of literals in each term
- Note that
  - The simplified expression may not be unique
  - In practice, the best expression depends on the target technology used to implement the digital circuit

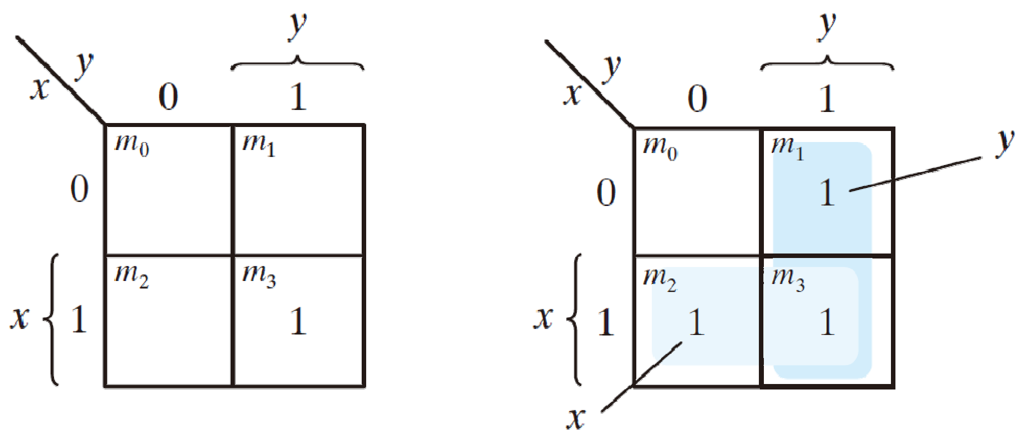
Two-Variable Map (1/2)

- A K-map is a truth table illustrated in square diagram
- Example: a two-variable map
  - A two-variable Boolean function has four minterms
  - x' shows up in row 0; x shows up in row 1
  - y' shows up in column 0; y shows up in column 1



Two-Variable Map (2/2)

- Mapping functions to K-maps:
  - xy is m_3, thus we put a 1 in the square of m_3 when the canonical form of the function contains m_3.
  - $x + y = x'y + xy' + xy$, put 1's in the squares of m_1, m_2, and m_3.



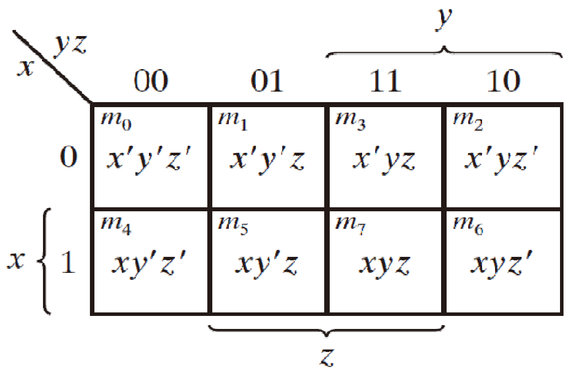
(a) xy (b) $x + y$

Three-Variable Map

- A three-variable Boolean function has eight minterms
- The minterm squares are arranged in the K-map by Gray code sequence
  - Any two adjacent squares in the map differ by one variable (i.e, primed in one square and unprimed in the other) $\rightarrow$ they can be merged into one term
  - Example: m_5 and m_7 can be simplified to xz.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

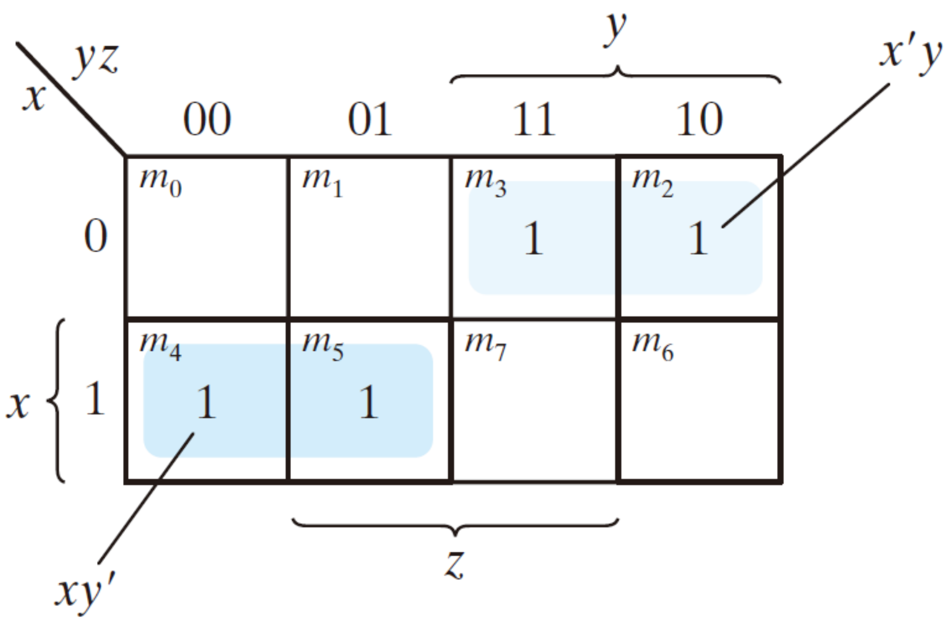
(a)



(b)

Example: Function Simplification

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



Simplification across Boundaries

- Merging of minterms can go across the boundaries
- Example: m_0 and m_2 as well as m_4 and m_6 are adjacent
  - $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
  - $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

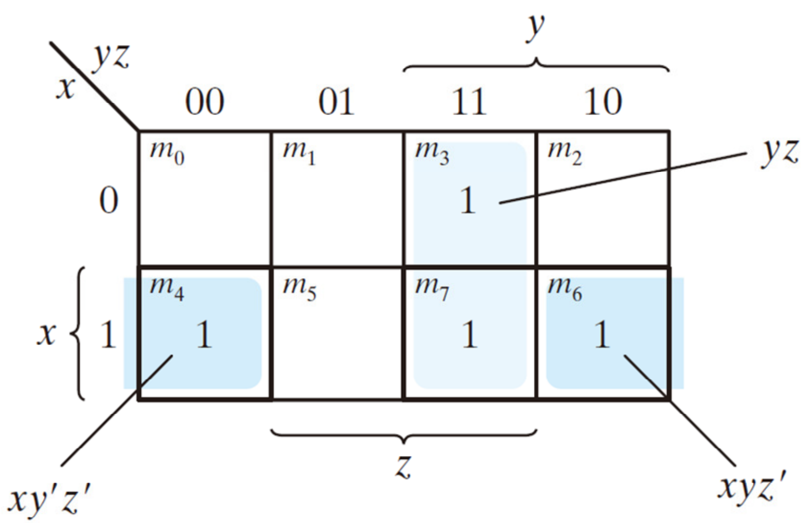
(a)

$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

(b)

Example: $F(x, y, z) = \Sigma(3, 4, 6, 7)$

- F contains four minterms, or two merged terms:
  $F(x, y, z) = (m_3 + m_7) + (m_6 + m_4) = yz + xz'$



Note: $xy'z' + xyz' = xz'$

Four-Square Simplification

- Four adjacent squares can be simplified as well:
  - $m_0 + m_2 + m_4 + m_6 = x'y'z' + x'yz' + xy'z' + xyz'$
    $= x'z'(y' + y) + xz'(y' + y)$
    $= x'z' + xz' = z'$
  - $m_1 + m_3 + m_5 + m_7 = x'y'z + x'yz + xy'z + xyz$
    $= x'z(y' + y) + xz(y' + y)$
    $= x'z + xz = z$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

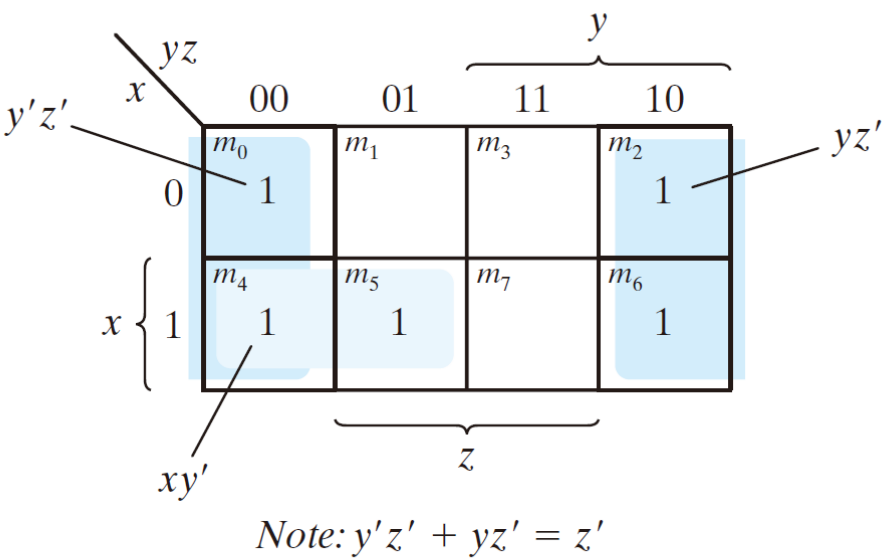
$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

y spans columns 11, 10; z spans columns 01, 11.

(b)

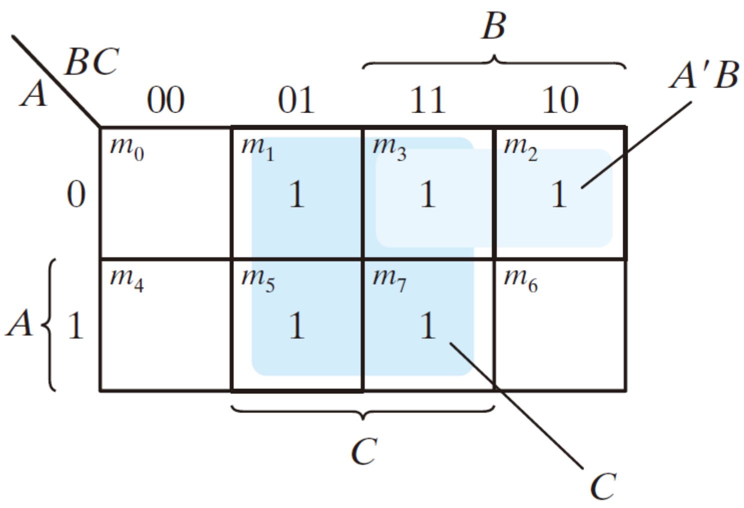
Example: $F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$

- F contains five minterms, with two overlapping areas, thus, the simplified function becomes $F = z' + xy'$



Equivalent Function Discovery

- K-Map can be used to find various forms of the same Boolean expression.
- Example: $F = A'C + A'B + AB'C + BC$
  - Express it in sum of minterms: $m_1 + m_2 + m_3 + m_5 + m_7$
  - Find the minimal sum of products expression: $A'B + C$



Four-Variable Maps (1/2)

- A four-variable Boolean function has 16 minterms
- There are sums of 2, 4, 8, and 16 adjacent squares

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

$wx \backslash yz$	00	01	11 (y)	10 (y)
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01 (x)	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11 (w, x)	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10 (w)	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

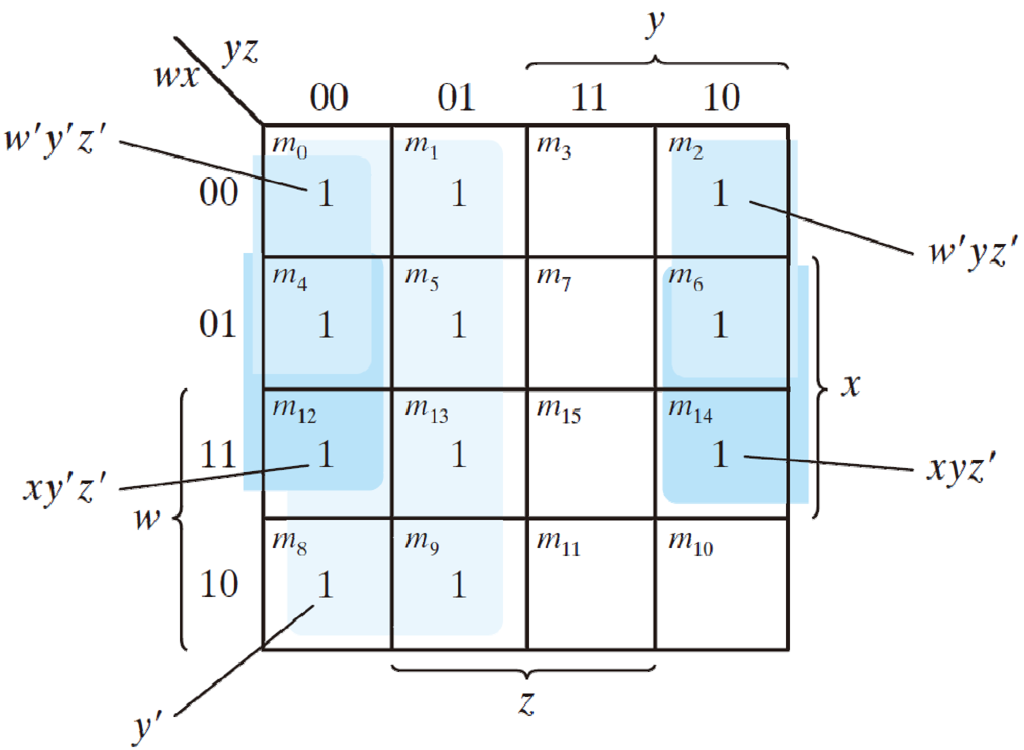
(b)

Four-Variable Maps (2/2)

- Simplification rules of adjacent squares that has 1's in the K-map are as follows
  - a 1-square represents one minterm with four literals
  - a 2-square reduces to a product term with three literals
  - a 4-square reduces to a product term with two literals
  - a 8-square reduces to a product term with one literal
  - a 16-square reduces to a constant 1
- Note:
  - There are usually more than one ways to simplify a function
  - Overlapping of n-squares do not affect the rules
- Goal: find the minimal number of n-squares that can cover all the 1's in the K-map

Example: Sum of 11 minterms

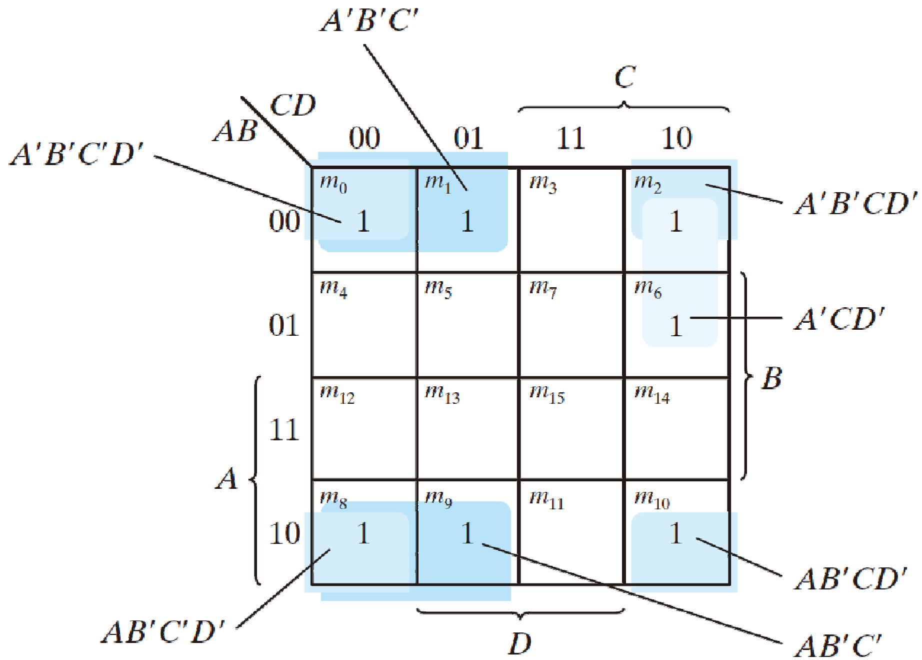
- $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$
  $\rightarrow F = y' + w'z' + xz'$



Note: $w'y'z' + w'yz' = w'z'$
$xy'z' + xyz' = xz'$

Example: $F = A'B'C' + B'CD' + A'B'CD' + AB'C'$

- $F \to B'D' + B'C' + A'CD'$:
  - Four corners: $A'B'C'D' + A'B'CD' + AB'C'D' + AB'CD' = B'D'$.
  - 4-squares on the left: $B'C'$.
  - 2-squares on the top-right: $A'CD'$.



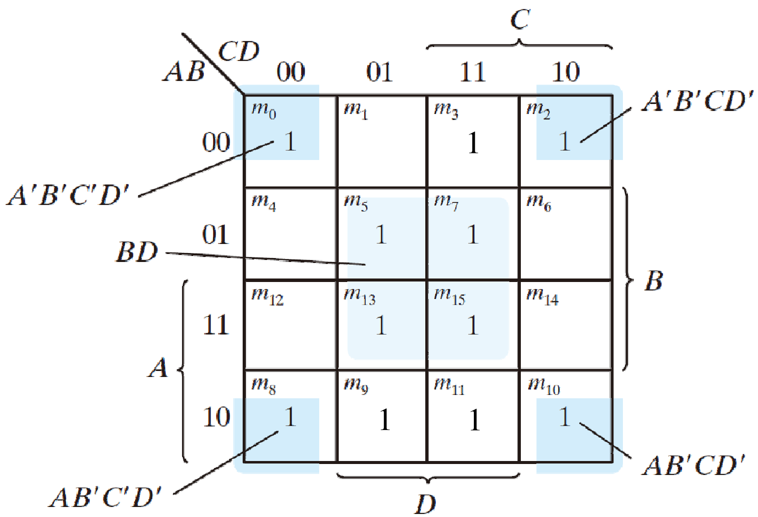
Prime Implicants

- A prime Implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.
  - A subset of a prime implicant is not a prime implicant. E.g., a 2-square contained in a 4-square is not a prime implicant.
- An essential prime implicant is the only prime implicant that covers certain minterms.
  - At least one of the minterms covered by the essential prime implicant is unique.
- A function can be simplified to summations of prime implicants.

Example: Prime Implicants

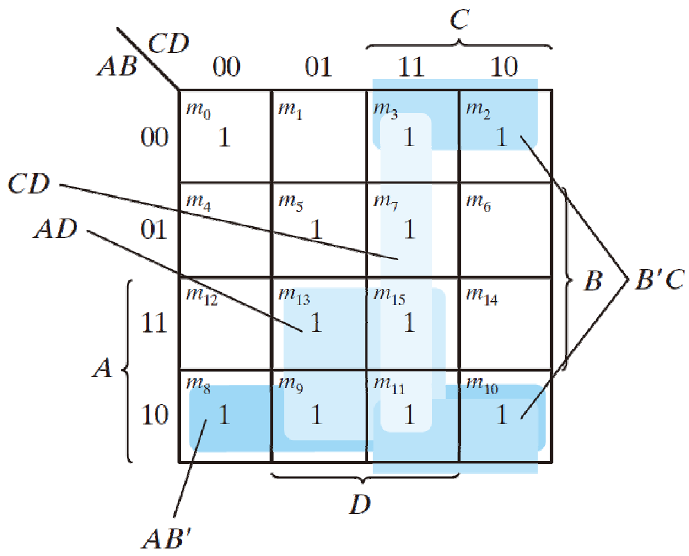
- $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$.
- $F = BD + B'D' + CD + AD = BD + B'D' + CD + AB'$

  $= BD + B'D' + B'C + AD = BD + B'D' + B'C + AB'$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$

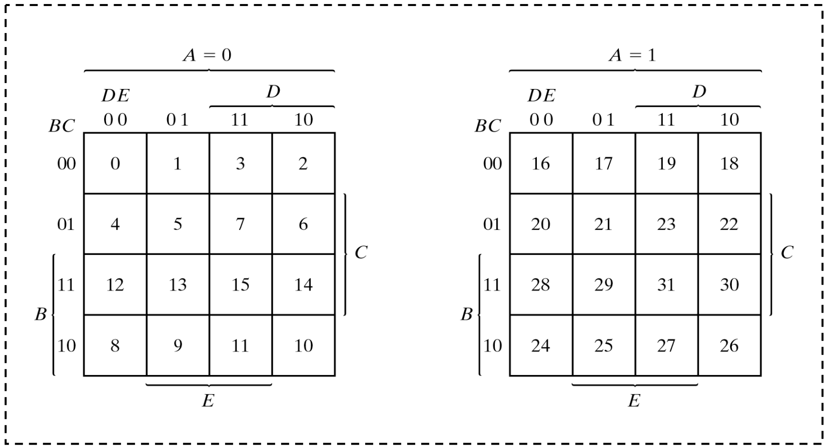
(a) Essential prime implicants BD and $B'D'$

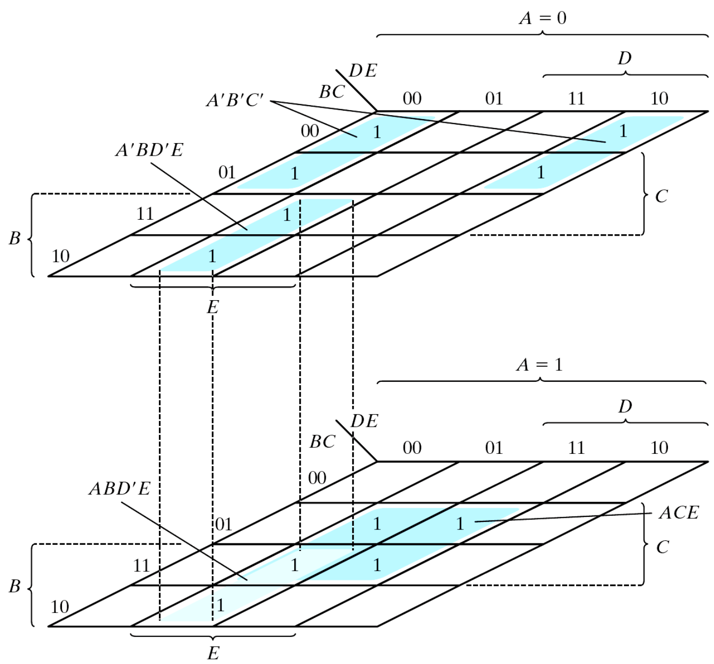


(b) Prime implicants CD, $B'C$, AD, and AB'

Five-Variable Map

- Map for more than four variables becomes very complicated
- A possible way to draw five-variable map is to stack two four-variable maps together:



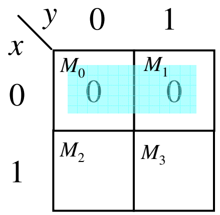


Product of Sum (PoS) Simplification

- K-map is designed for sum-of-product simplification
- To have simplified forms in product-of-sum, you can:

  (1) Apply duality principle in K-map:

  - $(x+y')(x+y) = x$

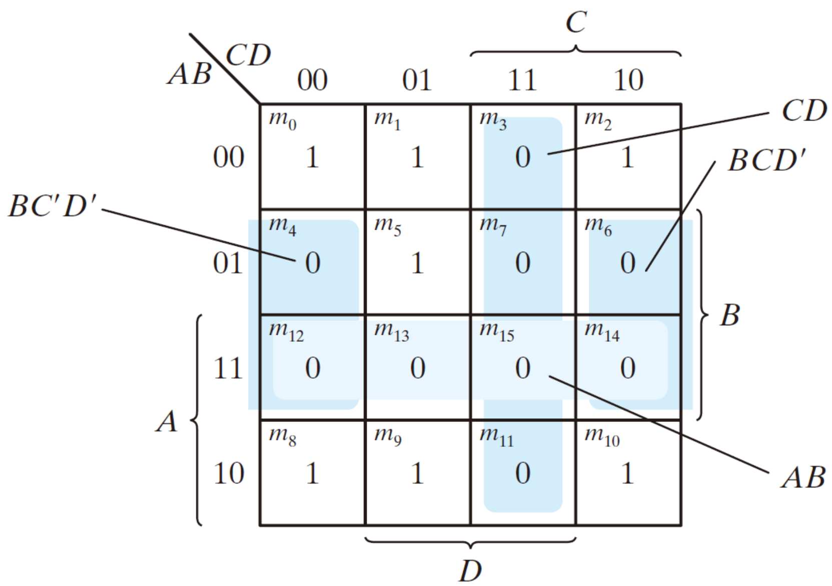


  (2) Start with sum-of-product form, and follow the steps:

  - Simplify F' in the form of sum-of-product
  - Apply DeMorgan's theorem $F = (F')'$ to turn sum-of-product into product-of-sum

Example: PoS Simplification (1/2)

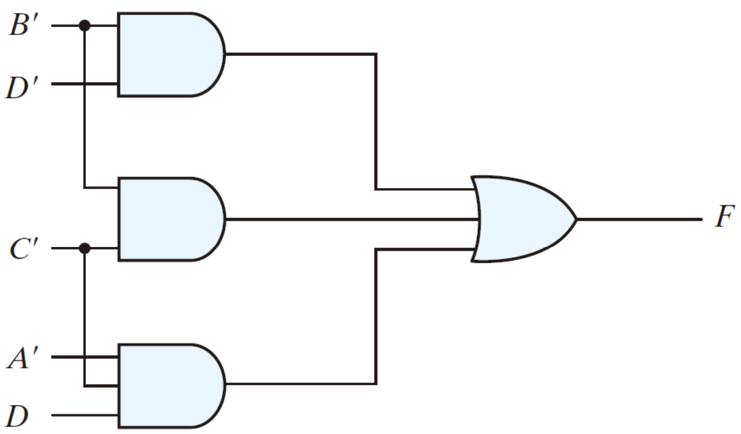
- $F = \Sigma(0, 1, 2, 5, 8, 9, 10) \rightarrow B'D' + B'C' + A'C'D$
- To get the simplified PoS form, we can combine zeros in the K-map to get the simplified form of F′
  - $F' = AB + CD + BD'$, then $F = (A'+B')(C'+D')(B'+D)$



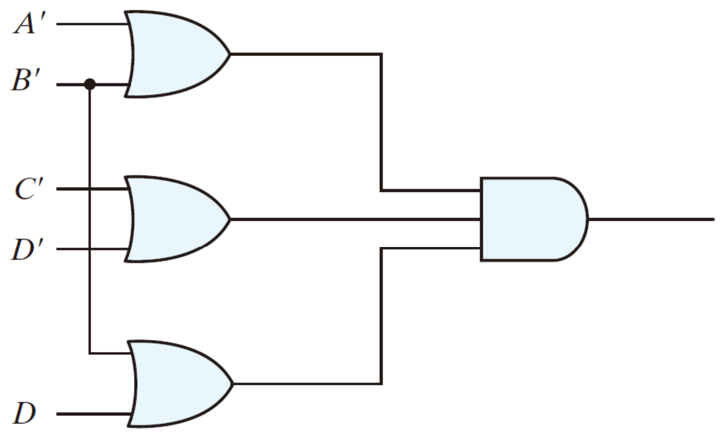
Note: $BC'D' + BCD' = BD'$

Example: PoS Simplification (2/2)

- Gate implementations of the simplified function



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



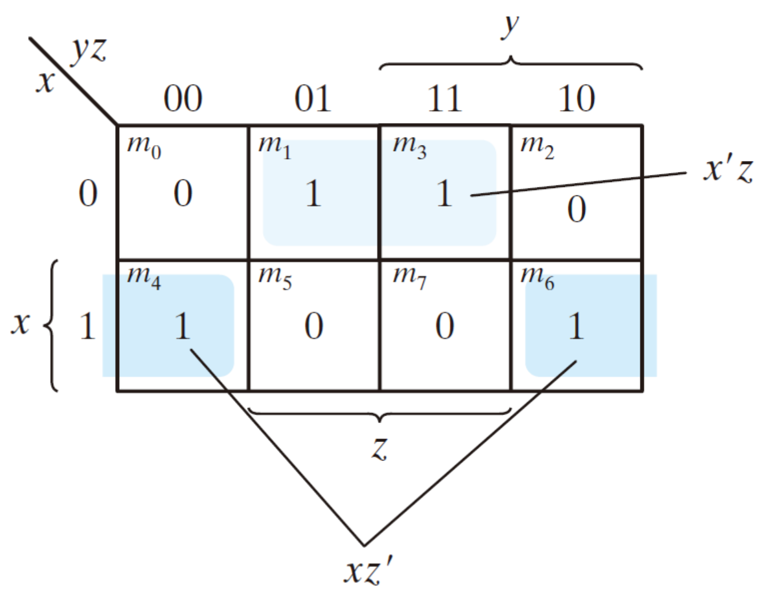
(b) $F = (A' + B')(C' + D')(B' + D)$

Summary Example

- $F = \Sigma(1, 3, 4, 6) = (\Sigma(0, 2, 5, 7))' = \Pi(0, 2, 5, 7)$
  - To simplify $\Sigma(1, 3, 4, 6)$, we combine 1's: $F = x'z + xz'$
  - To simplify $(\Sigma(0, 2, 5, 7))'$, we combine 0's: $F' = xz + x'z'$
  - To simplify $\Pi(0, 2, 5, 7)$, we complement the function to draw K-map, then complement the simplified form of $(\Sigma(0, 2, 5, 7))'$

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

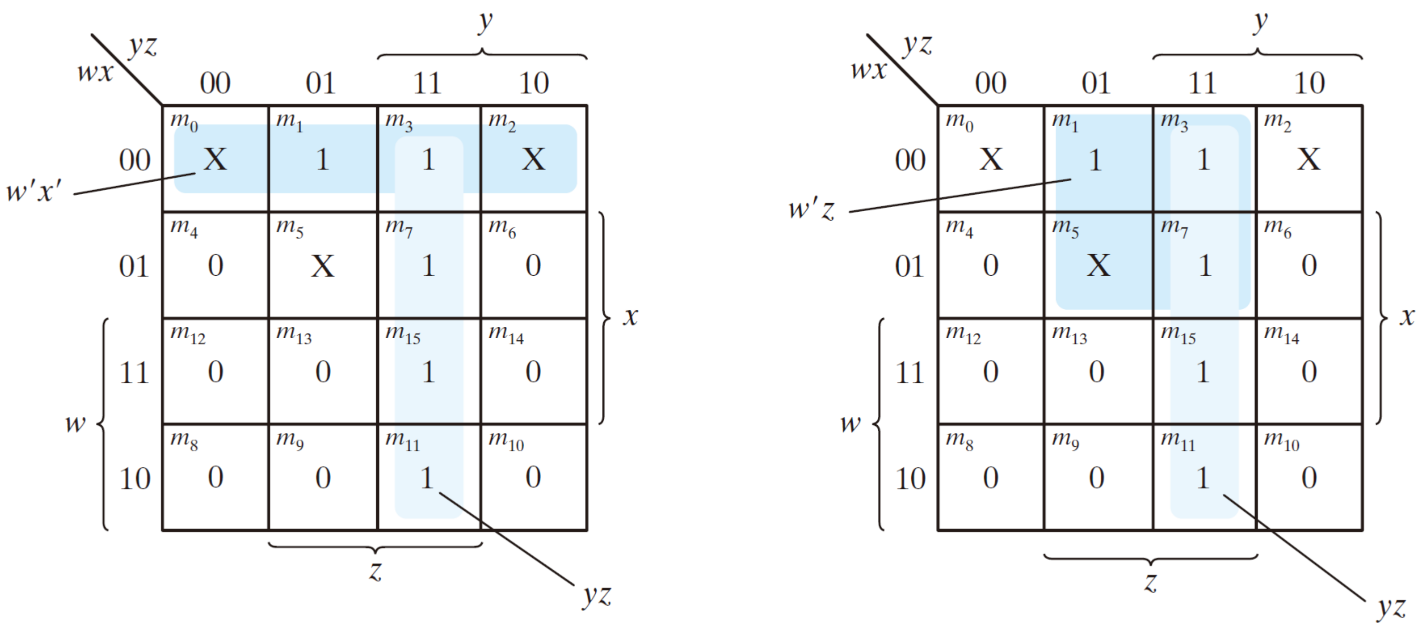


Don’t Care Conditions

- The value of a function is not specified for certain combinations of variables
  - In BCD codes; $1010 \sim 1111$: don't care
- The don't care conditions can be utilized in logic minimization
  - can be implemented as 0 or 1

Example: Don't Care Conditions

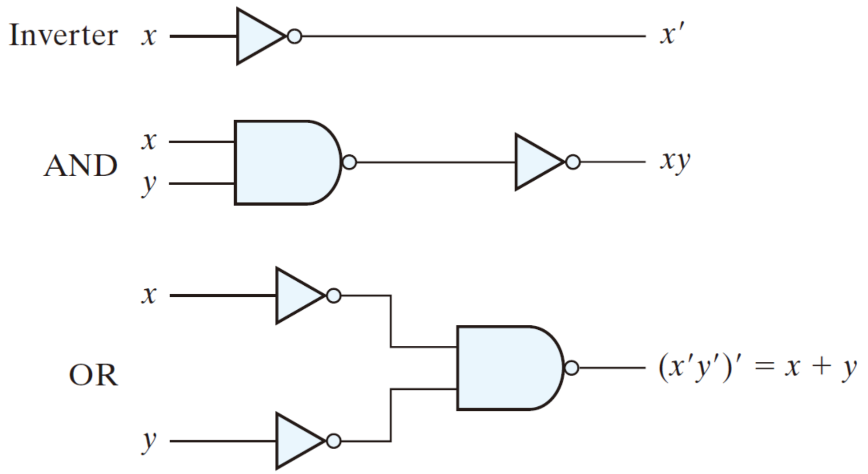
- $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$, $d(w, x, y, z) = \Sigma(0, 2, 5)$
  - (a) : $F = yz + w'x'$, from $F = \Sigma(0, 1, 2, 3, 7, 11, 15)$
  - (b) : $F = yz + w'z$, from $F = \Sigma(1, 3, 5, 7, 11, 15)$
  - Either expression is acceptable



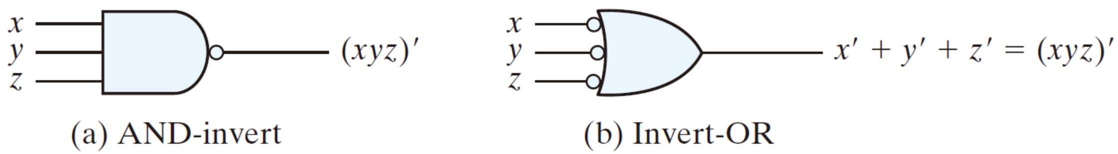
(a) $F = yz + w'x'$ (b) $F = yz + w'z$

NAND and NOR Implementation

- NAND gate is a universal gate
  - An universal gate can implement any digital system

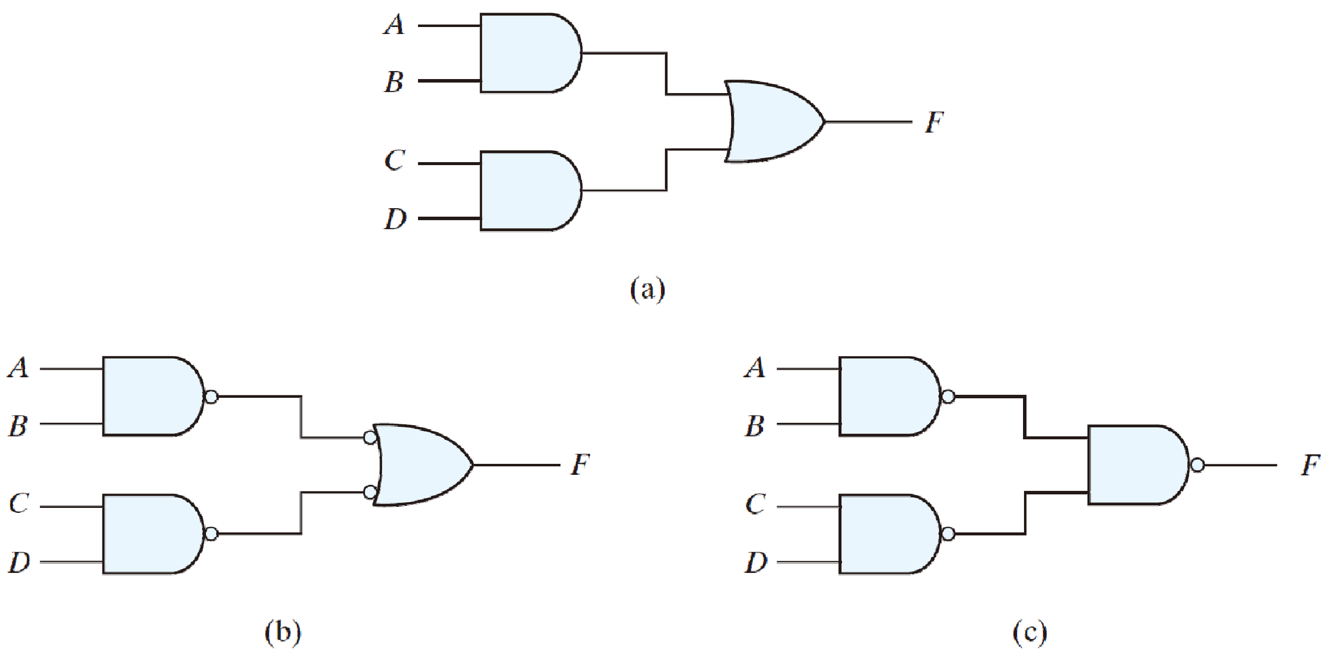


- Two symbols for a NAND gate:



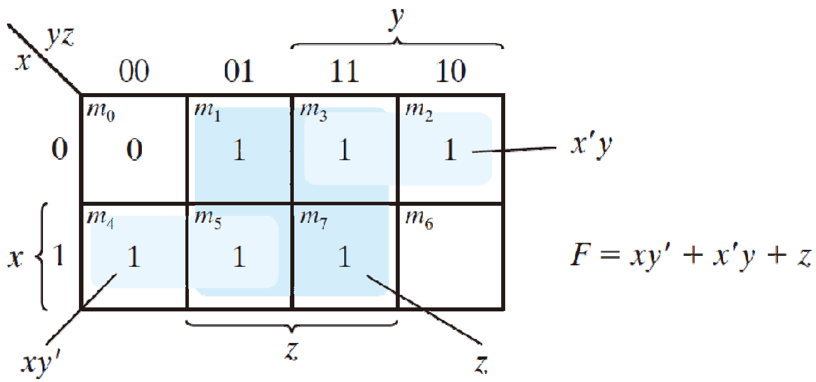
Two-level Implementation of Func.

- Use two-level logic to implement $F = AB + CD$
  - Sum of products can be implemented by NAND-NAND logic
  - $F = AB + CD = ((AB)'(CD)')'$

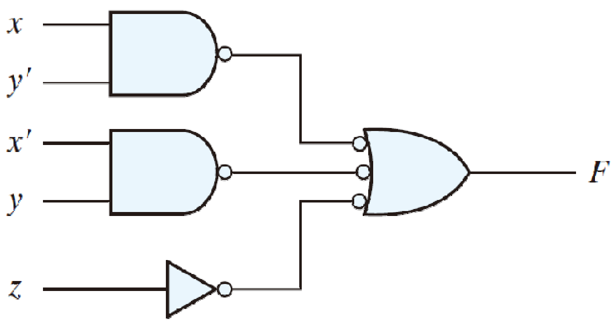


Example: NAND Implementation

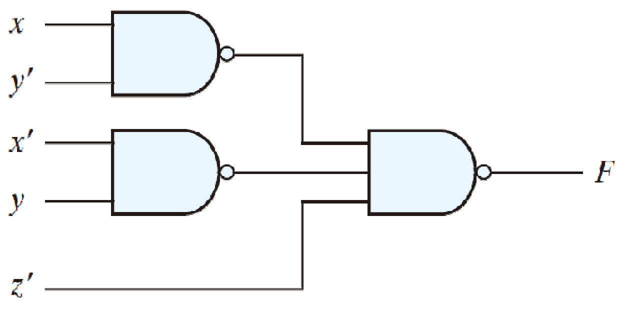
- $F(x, y, z) = \Sigma(1, 2, 3, 4, 5, 7) = xy' + x'y + z$,



(a)



(b)



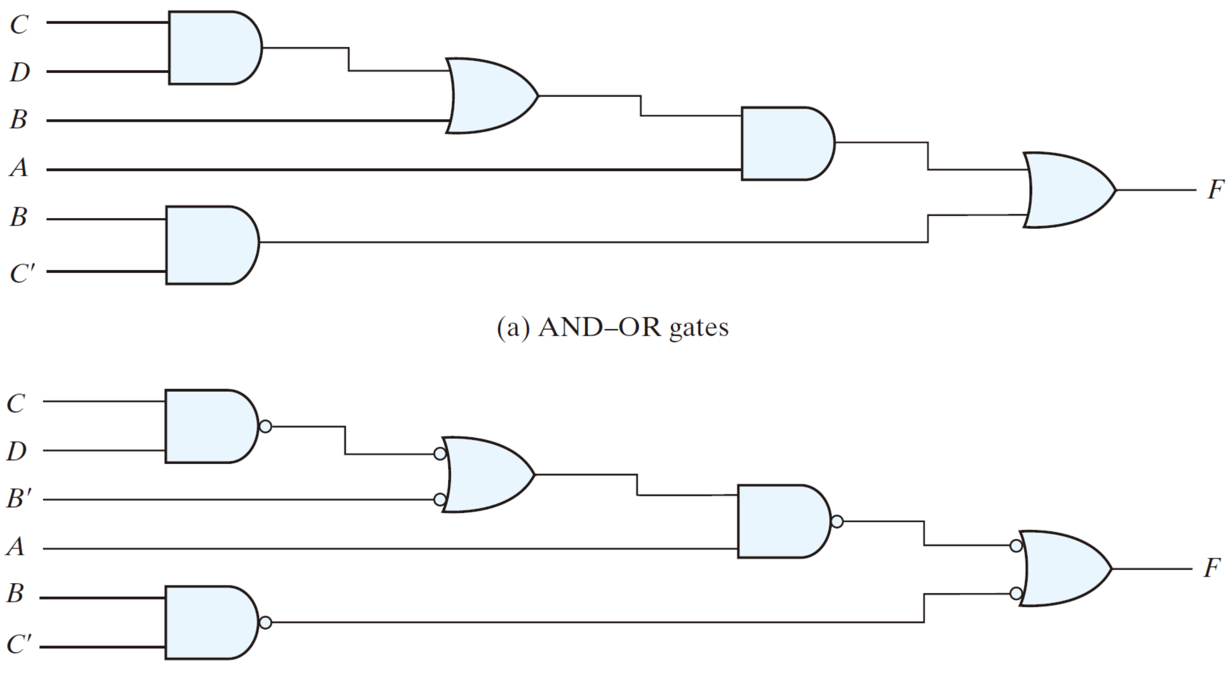
(c)

NAND Implementation Procedure

- To implement a function using NAND gates alone, execute the following procedure
  - Simplified in the form of sum of products
  - A NAND gate for each product term;
    the inputs to each NAND gate are the literals of the term
  - A single NAND gate for the second sum term
  - A single literal requires an inverter in the first level

Multilevel NAND Circuits

- Steps to convert AND-OR logic implementations to NAND-NAND logic implementations
  - AND $\rightarrow$ NAND + inverter-to-next-input
  - OR $\rightarrow$ inverted-input + OR ($\equiv$ NAND)

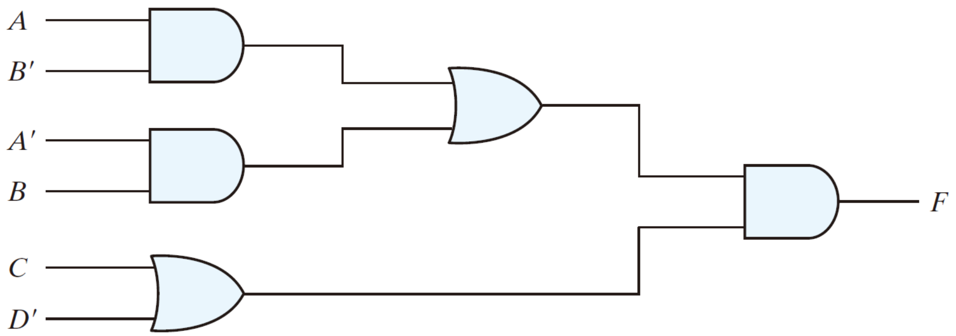


(a) AND–OR gates

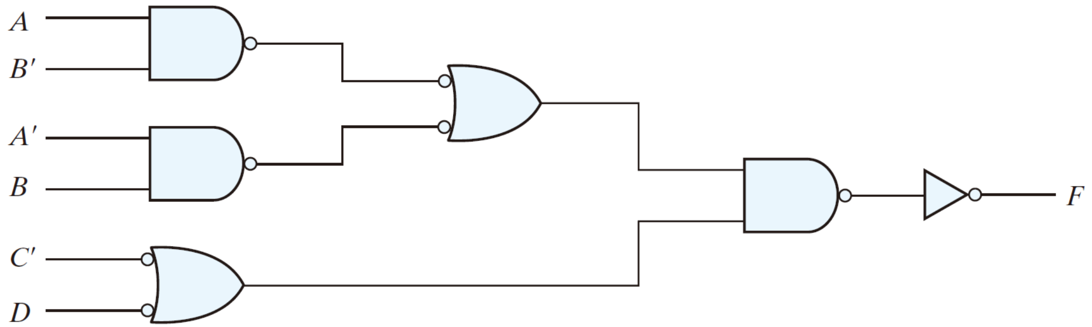
(b) NAND gates

NAND Implementation

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



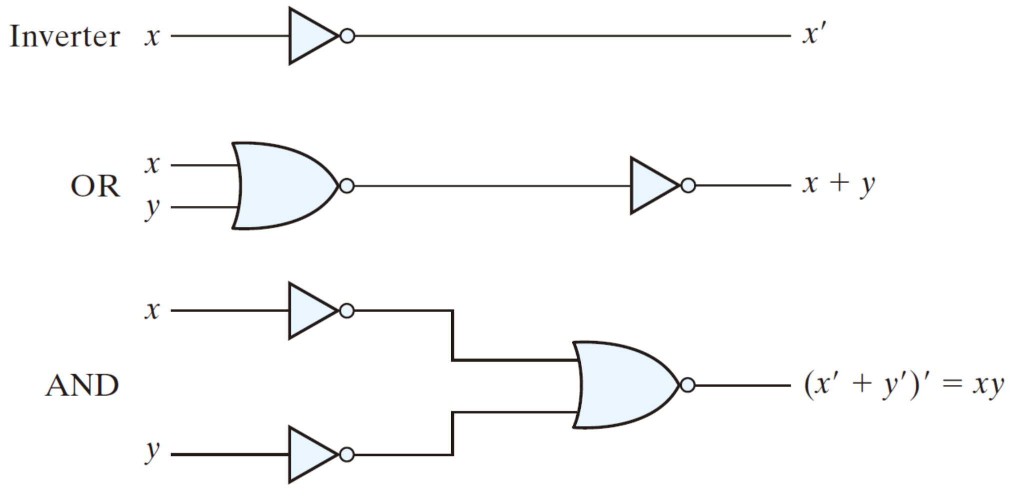
(a) AND–OR gates



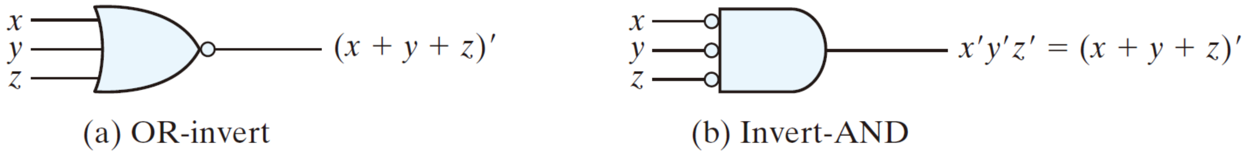
(b) NAND gates

NOR Implementation

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



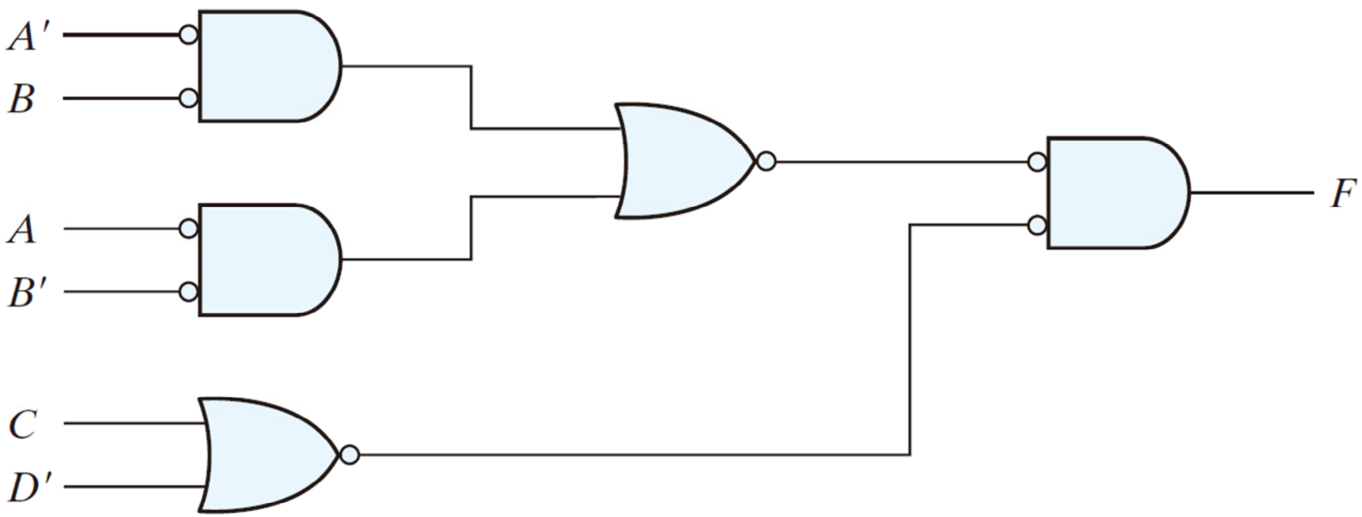
- Two symbols for a NOR gate:



Multilevel NOR Circuits

- Steps to convert AND-OR logic to NOR-NOR logic
  - AND $\rightarrow$ inverted-input + AND ($\equiv$ NOR)
  - OR $\rightarrow$ NOR

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

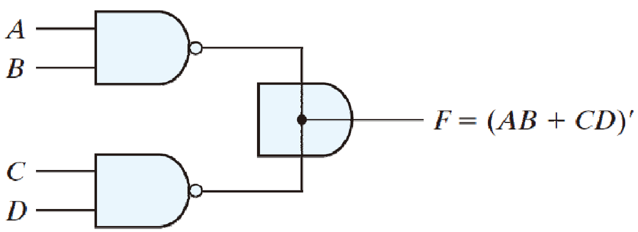


Other Two-level Implementations

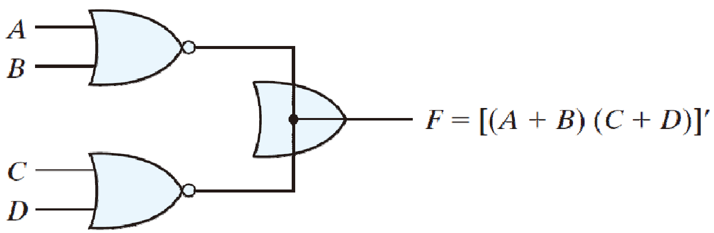
- Two-level function implementation is the basic of digital computing, we have studied AND-OR, OR-AND, NAND-NAND, and NOR-NOR
- There are other possibilities of two-level implementations
  - Wired logic
  - Non-degenerate forms of NOR-OR, NAND-AND, OR-AND, and AND-OR

Wired Logic Implementation

- Some gates can use a wire connection of the outputs to "*emulate an extra 2nd-level gate*"
- Examples:
  - wire TTL NAND gates gives us wired-AND logic $\rightarrow$ AND-OR-INVERT function: $F = (AB)'(CD)'$
  - wire ECL NOR gates gives us wired-OR logic $\rightarrow$ OR-AND-INVERT function $F = (A+B)' + (C+D)'$



(a) Wired-AND in open-collector TTL NAND gates.



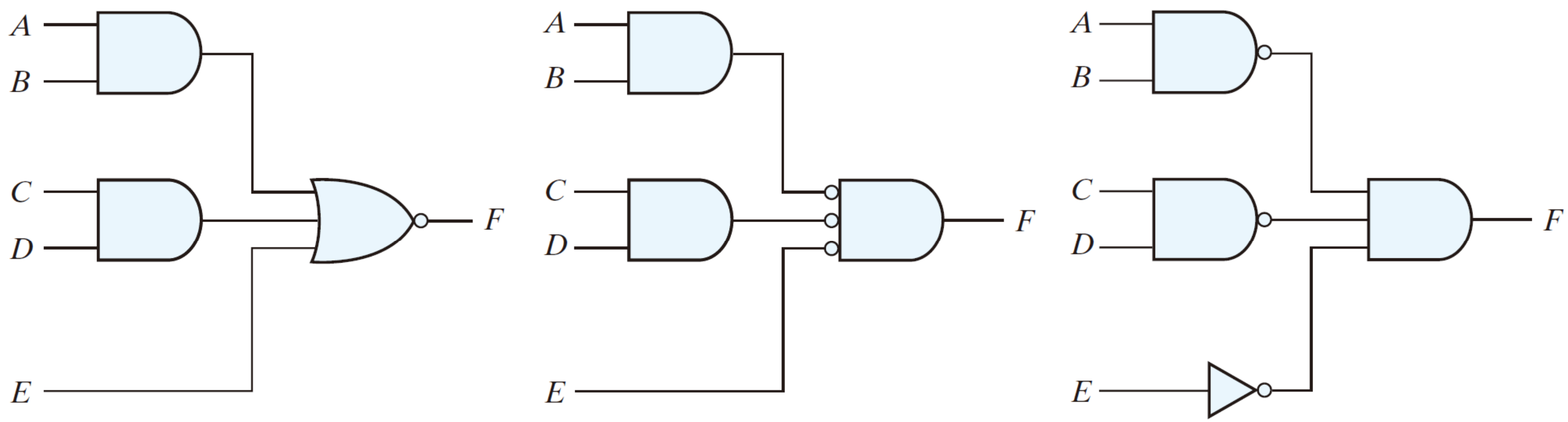
(b) Wired-OR in ECL gates

Non-degenerate Forms

- If four types of gates, AND, OR, NAND, NOR, are used for two-level logics, there are 16 combinations
- Eight of them are degenerate forms, i.e., equivalent to a single-level logic of multiple inputs
  - Examples: AND-AND $\rightarrow$ AND; OR-NOR $\rightarrow$ NOR
- The other eight non-degenerate forms are:
  - AND-OR, OR-AND, NAND-NAND, NOR-NOR, NAND-AND, AND-NOR, OR-AND, NOR-OR
- Popular forms:
  - AND-OR and NAND-NAND = sum of products
  - OR-AND and NOR-NOR = product of sums

AND-OR-INVERT Circuits

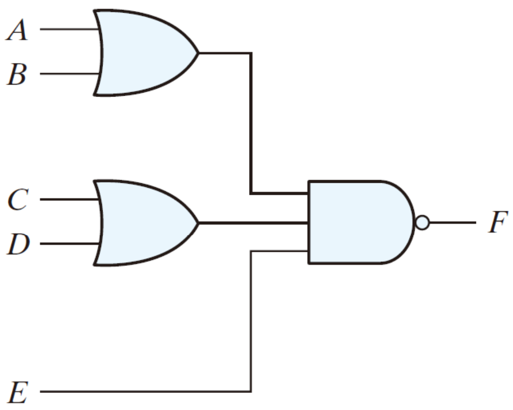
- AND-OR-INVERT (AOI) Implementation
  - AND-NOR = NAND-AND
  - $F = (AB + CD + E)'$



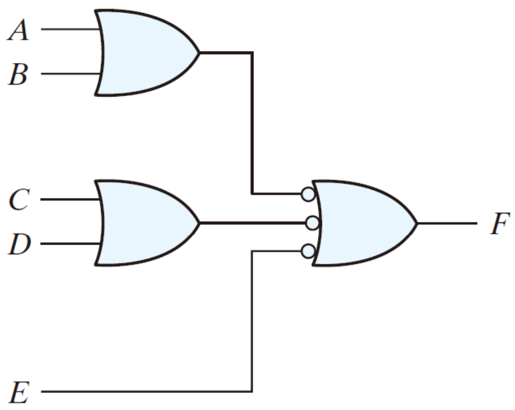
(a) AND–NOR (b) AND–NOR (c) NAND–AND

OR-AND-INVERT Circuits

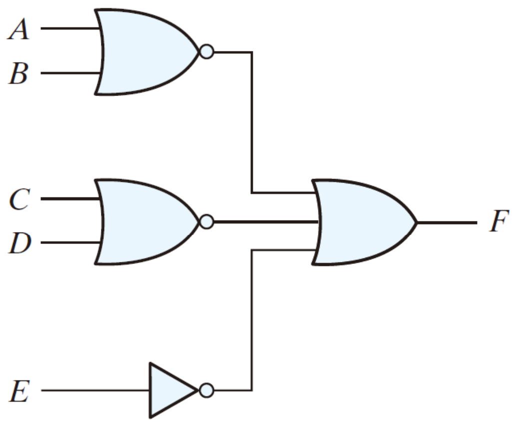
- OR-AND-INVERT (OAI) Implementation
  - OR-NAND = NOR-OR
  - $F = ((A+B)(C+D)E)'$



(a) OR–NAND



(b) OR–NAND



(c) NOR–OR

Summary of Two-Level Forms

- Implementation with other two-level forms:

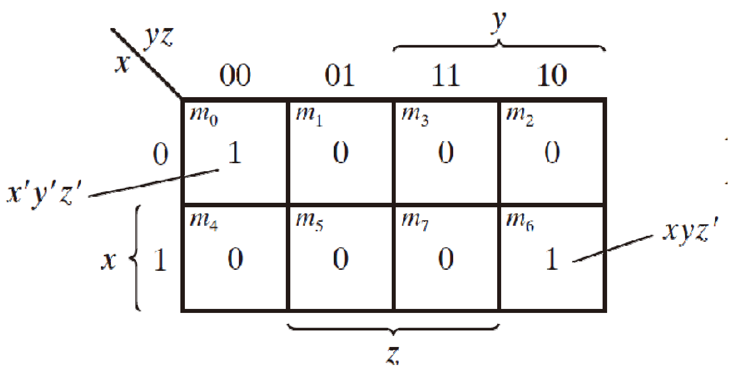
Implementation with Other Two-Level Forms

Equivalent Nondegenerate Form (a)	Equivalent Nondegenerate Form (b)*	Implements the Function	Simplify F' into	To Get an Output of
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0's in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1's in the map and then complementing.	F

*Form (b) requires an inverter for a single literal term.

Example: $F = x'y'z' + xyz'$ (1/2)

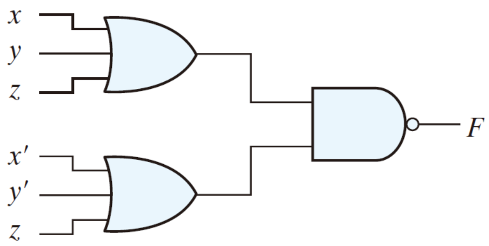
- Compliment of the function is $F' = (x+y+z)(x'+y'+z)$
  - OAI expression: $((x+y+z)(x'+y'+z))'$
- K-map of the function:



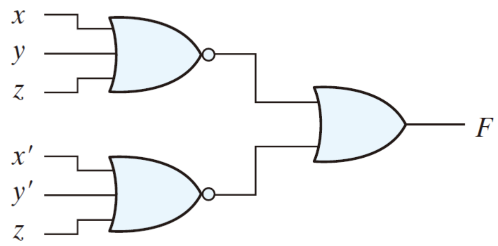
- Complement of the simplified form is $F' = x'y + xy' + z$
  - F' is simplified by combining zeros in K-map
  - AOI expression: $F = (x'y + xy' + z)'$

Example: $F = x'y'z' + xyz'$ (2/2)

- Gate Implementations:
  - OAI: $F = ((x+y+z)(x'+y'+z))'$

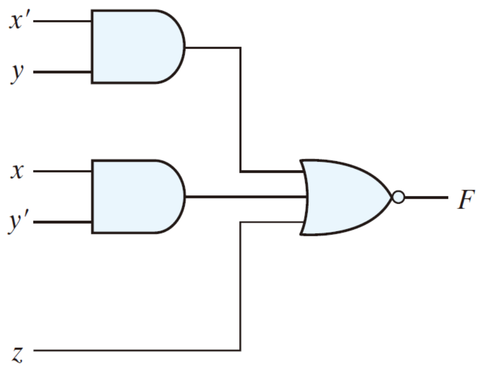


OR–NAND

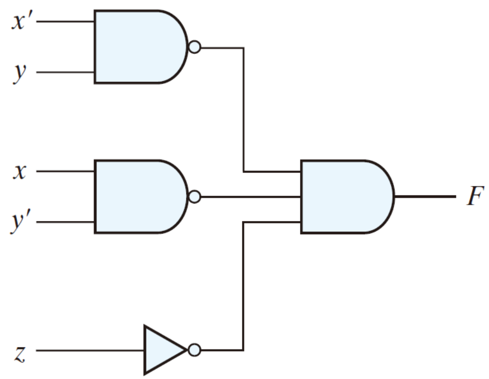


NOR–OR

  - AOI: $F = (x'y + xy' + z)'$



AND–NOR



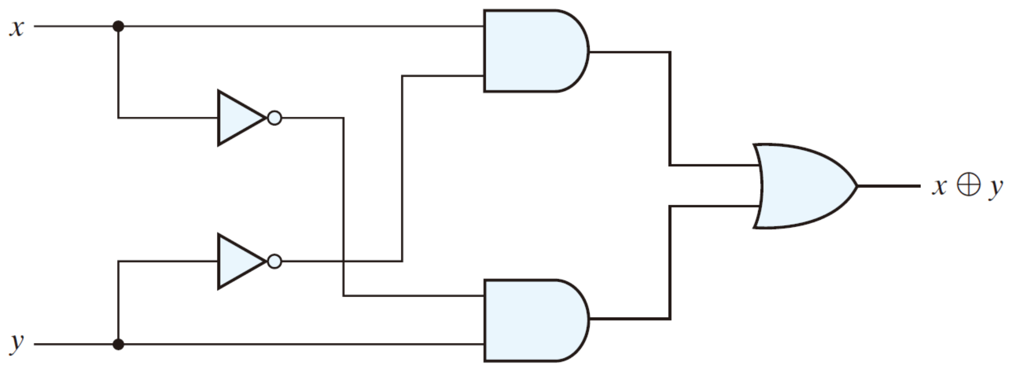
NAND–AND

Exclusive-OR Function

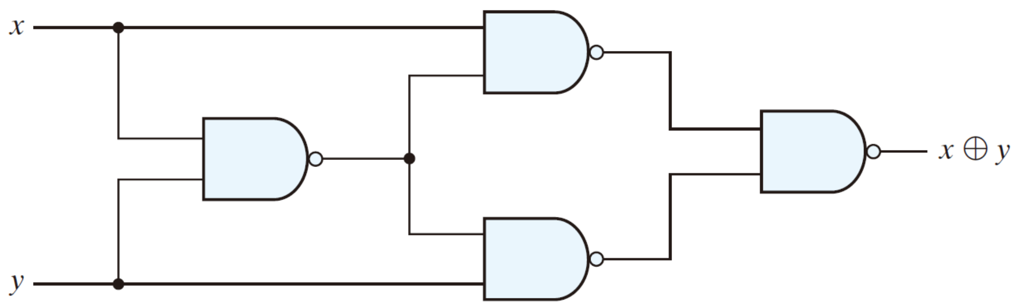
- Exclusive-OR (XOR): $x \oplus y = xy' + x'y$
- Exclusive-NOR (XNOR): $(x \oplus y)' = xy + x'y'$
- Some identities:
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = x' \oplus y = (x \oplus y)'$
- Commutative and associative
  - $A \oplus B = B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

Implementations

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



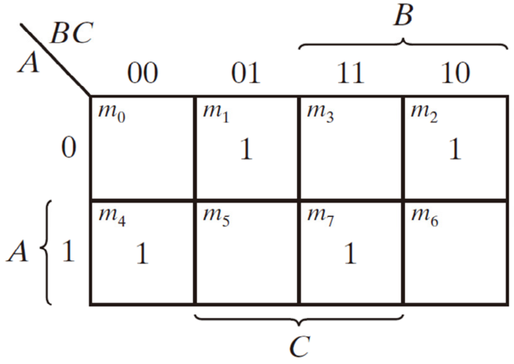
(a) Exclusive-OR with AND–OR–NOT gates



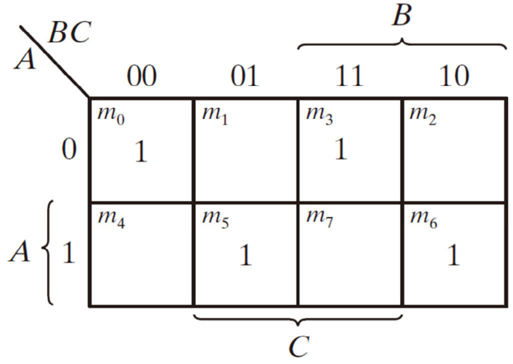
(b) Exclusive-OR with NAND gates

Odd Function

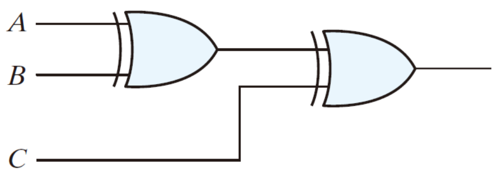
- Exclusive-OR can be used to determine whether the number of 1's is odd
- Example: $A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C$
  $= AB'C' + A'BC' + ABC + A'B'C = \Sigma(1, 2, 4, 7)$



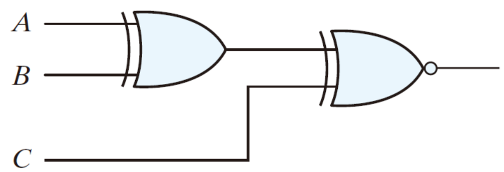
(a) Odd function $F = A \oplus B \oplus C$



(b) Even function $F = (A \oplus B \oplus C)'$



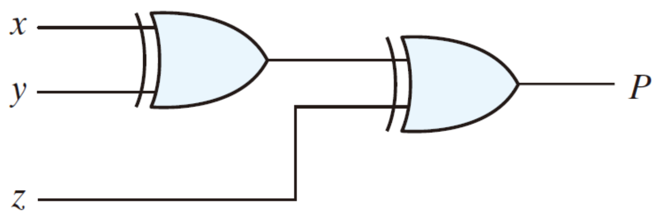
(a) 3-input odd function



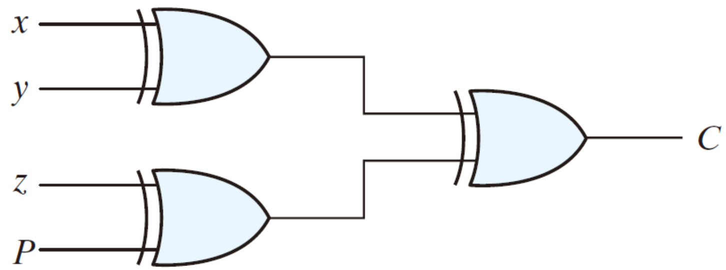
(b) 3-input even function

Parity Generation and Checking

- XOR can be used for parity generation and checking
  - Parity bit of three variables: $P = x \oplus y \oplus z$
  - Parity check function: $C = x \oplus y \oplus z \oplus P$, if even parity is used
    - $C = 1$: an odd number of data bit error
    - $C = 0$: correct or an ever number of data bit error



(a) 3-bit even parity generator



(b) 4-bit even parity checker