Gate-Level Minimization

ENCS2340 - Digital Systems

Dr. Ahmed I. A. Shawahna

Electrical and Computer Engineering Department

Birzeit University

Presentation Outline

- **Boolean Function Minimization**
- **The Karnaugh Map (K-Map)**
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- Don't Cares
- Five and Six-Variable K-Maps
- Multiple Outputs
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- Odd and Even functions
- Parity Generators and Checkers

Boolean Function Minimization

- Complexity of a Boolean function is directly related to the complexity of the algebraic expression
- The *truth table* of a function is unique
- However, the *algebraic expression* is not unique
- Boolean function can be simplified by algebraic manipulation
- However, algebraic manipulation depends on experience
- Algebraic manipulation does not guarantee that the simplified Boolean expression is minimal

Example: Sum of Minterms

Truth Table

x y z	f	Minterm
0 0 0	0	
0 0 1	1	$m_1 = x'y'z$
0 1 0	1	$m_2 = x'yz'$
0 1 1	1	$m_3 = x'yz$
1 0 0	0	
1 0 1	1	$m_5 = xy'z$
1 1 0	0	
1 1 1	1	$m_7 = xyz$

Focus on the '**1**' entries

$$f = m_1 + m_2 + m_3 + m_5 + m_7$$

$$f = \sum(1, 2, 3, 5, 7)$$

$$f = x'y'z + x'yz' + x'yz + xy'z + xyz$$

❖ Sum-of-Minterms has 15 literals ➔ Can be simplified

Algebraic Manipulation

❖ **Simplify:** $f = x'y'z + x'yz' + x'yz + xy'z + xyz$ (15 literals)

$f = x'y'z + x'yz' + x'yz + xy'z + xyz$	(Sum-of-Minterms)
$f = x'y'z + x'yz + x'yz' + xy'z + xyz$	Reorder
$f = x'z(y' + y) + x'yz' + xz(y' + y)$	Distributive $\cdot$ over +
$f = x'z + x'yz' + xz$	Simplify (7 literals)
$f = x'z + xz + x'yz'$	Reorder
$f = (x' + x)z + x'yz'$	Distributive $\cdot$ over +
$f = z + x'yz'$	Simplify (4 literals)
$f = (z + x'y)(z + z')$	Distributive + over $\cdot$
$f = z + x'y$	Simplify (3 literals)

Drawback of Algebraic Manipulation

- No clear steps in the manipulation process
  - Not clear which terms should be grouped together
  - Not clear which property of Boolean algebra should be used next
- Does not always guarantee a minimal expression
  - Simplified expression may or may not be minimal
  - Different steps might lead to different non-minimal expressions
- However, the goal is to minimize a Boolean function
- Minimize the **number of literals** in the Boolean expression
  - The **literal count** is a good measure of the **cost** of logic implementation
  - Proportional to the number of transistors in the circuit implementation

Karnaugh Map

- Called also K-map for short
- The Karnaugh map is a diagram made up of squares
- It is a reorganized version of the truth table
- Each **square** in the Karnaugh map represents a minterm
- Adjacent squares differ in the value of one variable
- Simplified expressions can be derived from the Karnaugh map
  - By recognizing patterns of squares
- Simplified sum-of-products expression (AND-OR circuits)
- Simplified product-of-sums expression (OR-AND circuits)

Next . . .

- ❖ Boolean Function Minimization
- ❖ The Karnaugh Map (K-Map)
- ❖ **Two, Three, and Four-Variable K-Maps**
- ❖ Prime and Essential Prime Implicants
- ❖ Minimal Sum-of-Products and Product-of-Sums
- ❖ Don't Cares
- ❖ Five and Six-Variable K-Maps
- ❖ Multiple Outputs
- ❖ Universality of NAND and NOR gates
- ❖ NAND-NAND and NOR-NOR implementations
- ❖ Odd and Even functions
- ❖ Parity Generators and Checkers

Two-Variable Karnaugh Map

- Minterms m_0 and m_1 are adjacent (also, m_2 and m_3)
  - They differ in the value of variable y
- Minterms m_0 and m_2 are adjacent (also, m_1 and m_3)
  - They differ in the value of variable x

Two-variable K-map

$x \backslash y$	0	1
0	m_0	m_1
1	m_2	m_3

$x \backslash y$	0	1
0	$x'y'$	$x'y$
1	xy'	xy

Note: adjacent squares horizontally and vertically ***NOT*** *diagonally*

From a Truth Table to Karnaugh Map

- ❖ Given a truth table, construct the corresponding K-map
- ❖ Copy the function values from the truth table into the K-map
- ❖ Make sure to copy each value into the proper K-map square

Truth Table

x y	f
0 0	1
0 1	0
1 0	1
1 1	1

K-map

$x \backslash y$	0	1
0	1	0
1	1	1

K-Map Function Minimization

- ❖ Two adjacent cells containing 1's can be combined
- ❖ $f = m_0 + m_2 + m_3$
- ❖ $f = x'y' + xy' + xy$ (6 literals)
- ❖ $m_0 + m_2 = x'y' + xy' = (x' + x)y' = y'$
- ❖ $m_2 + m_3 = xy' + xy = x(y' + y) = x$
- ❖ Therefore, f can be simplified as: $f = x + y'$ (2 literals)

K-map

$x \backslash y$	0	1
0	1	0
1	1	1

Example - Two-Variable Karnaugh Map

❖ Given the truth table of the Boolean function f, express f in the minimal sum-of-products form.

Truth Table

x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

K-map

$x \backslash y$	0	1
0	0	1
1	1	1

Therefore, f can be simplified as: $f = x + y$ (2 literals)

Three-Variable Karnaugh Map

- Have eight squares (for the 8 minterms), numbered 0 to 7
- The last two columns are not in numeric order: 11, 10
  - Remember the numbering of the squares in the K-map
- Each square is adjacent to three other squares
- Minterms in adjacent squares can always be combined
  - This is the key idea that makes the K-map work
- Labeling of rows and columns is also useful

$x \backslash yz$	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

	$x \backslash yz$	00	01	11	10
		y'		y	
x'	0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
x	1	$xy'z'$	$xy'z$	xyz	xyz'
		z'	z		z'

Simplifying a Three-Variable Function

Simplify the Boolean function: $f(x, y, z) = \sum(3, 4, 5, 7)$

$f = x'yz + xy'z' + xy'z + xyz$ (12 literals)

1. Mark **'1'** all the K-map squares that represent function f

2. Find possible adjacent squares

	y'	y'	y	y
x \ yz	00	01	11	10
x' 0	0	0	1	0
x 1	1	1	1	0
	z'	z	z	z'

$x'yz + xyz = (x' + x)yz = yz$

$xy'z' + xy'z = xy'(z' + z) = xy'$

Therefore, $f = xy' + yz$ (4 literals)

Simplifying a Three-Variable Function (2)

Here is a second example: $f(x, y, z) = \sum(3, 4, 6, 7)$

$f = x'yz + xy'z' + xyz' + xyz$ (12 literals)

Learn the locations of the 8 indices based on the variable order

$x'yz + xyz = (x' + x)yz = yz$

Corner squares can be combined

$xy'z' + xyz' = xz'(y' + y) = xz'$

Therefore, $f = xz' + yz$ (4 literals)

		y'		y	
$x \backslash yz$		00	01	11	10
x'	0	0	0	1	0
x	1	1	0	1	1
		z'	z	z	z'

Combining Squares on a 3-Variable K-Map

- By combining squares, we reduce number of literals in a product term, thereby reducing the cost

- On a 3-variable K-Map:

  - One square represents a minterm with 3 variables

  - Two adjacent squares represent a term with 2 variables

  - Four adjacent squares represent a term with 1 variable

  - Eight adjacent square is the constant **‘1’** (no variables)

Minimal Sum-of-Products Expression

Consider the function: $f(x, y, z) = \sum(2, 3, 4, 5)$

Find a minimal sum-of-products (SOP) expression

Solution:

Green block: term = $x'y$

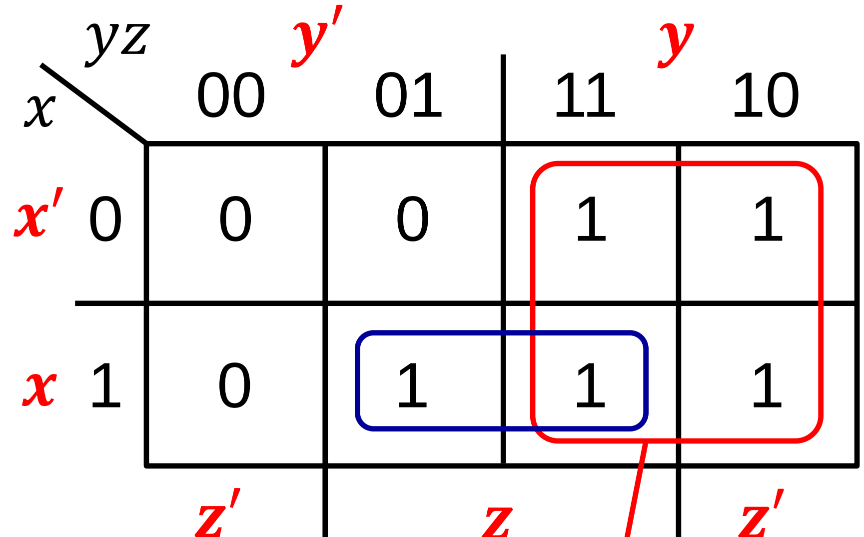
Blue block: term = xy'

$x \backslash yz$	00 (y', z')	01 (y', z)	11 (y, z)	10 (y, z')
x' 0	0	0	1	1
x 1	1	1	0	0

Minimal sum-of-products: $f = x'y + xy'$ (4 literals)

Example of Combining Squares

- Consider the Boolean function: $f(x, y, z) = \sum(2, 3, 5, 6, 7)$
- $f = x'yz' + x'yz + xy'z + xyz' + xyz$
- The four minterms that form the 2×2 red square are reduced to the term y
- The two minterms that form the blue rectangle are reduced to the term xz
- Therefore: $f = y + xz$



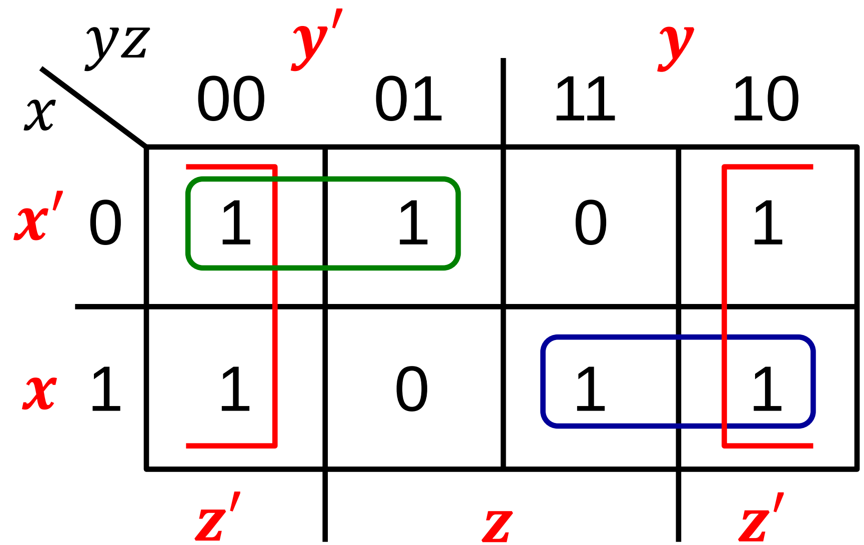
$$x'yz + x'yz' + xyz + xyz'$$
$$= x'y(z + z') + xy(z + z')$$
$$= x'y + xy = (x' + x)y = y$$

Minimal Sum-of-Products Expression

Consider the function: $f(x, y, z) = \sum(0, 1, 2, 4, 6, 7)$

Find a minimal sum-of-products (SOP) expression

Solution:



Red block: term = z'

Green block: term = $x'y'$

Blue block: term = xy

Minimal sum-of-products: $f = z' + x'y' + xy$ (5 literals)

Example

❖ For the Boolean function

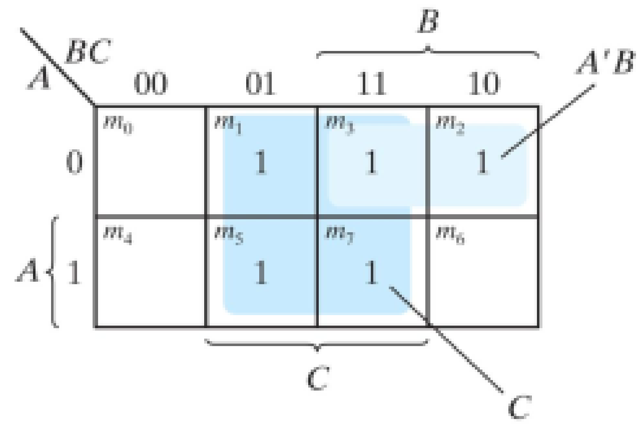
$$\boldsymbol{f(A,B,C) = A'C + A'B + AB'C + BC}$$

a) Express the function as a sum-of-minters

b) Find the minimal sum-of-products expression

Solution:

$$\begin{aligned}
&= A'(B+B')C + A'B(C+C') + AB'C \\
&\quad +(A+A')BC \\
&= A'BC + A'B'C + \cancel{A'BC} + A'BC' + AB'C \\
&\quad +ABC + \cancel{A'BC} \\
&= A'BC + A'B'C + A'BC' + AB'C + ABC \\
&= \sum(1,2,3,5,7)
\end{aligned}$$



$$f(A,B,C) = A'B + C$$

Four-Variable Karnaugh Map

4 variables ➔ 16 squares

Remember the numbering of the squares in the K-map

Each square is adjacent to four other squares

$$
\begin{aligned}
m_0 &= w'x'y'z' & m_1 &= w'x'y'z \\
m_2 &= w'x'y\,z' & m_3 &= w'x'y\,z \\
m_4 &= w'x\,y'z' & m_5 &= w'x\,y'z \\
m_6 &= w'x\,y\,z' & m_7 &= w'x\,y\,z \\
m_8 &= w\,x'y'z' & m_9 &= w\,x'y'z \\
m_{10} &= w\,x'yz' & m_{11} &= w\,x'y\,z \\
m_{12} &= w\,x\,y'z' & m_{13} &= w\,x\,y'z \\
m_{14} &= w\,x\,y\,z' & m_{15} &= w\,x\,y\,z
\end{aligned}
$$

Notice the order of Rows 11 and 10 and the order of columns 11 and 10

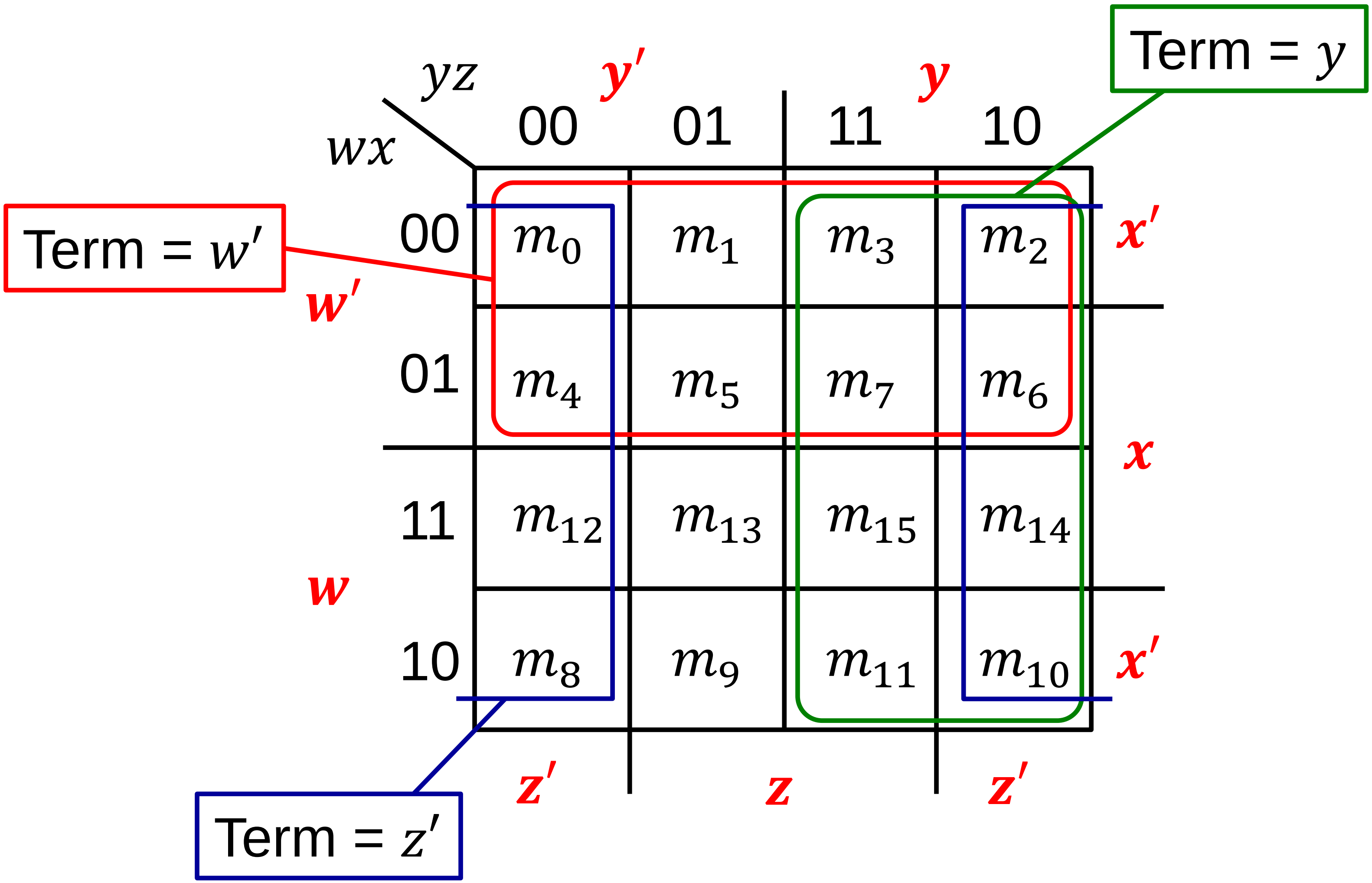
	wx \ yz	00 (y')	01 (y')	11 (y)	10 (y)	
w'	00	m_0	m_1	m_3	m_2	x'
w'	01	m_4	m_5	m_7	m_6	x
w	11	m_{12}	m_{13}	m_{15}	m_{14}	x
w	10	m_8	m_9	m_{11}	m_{10}	x'
		z'	z	z	z'	

Combining Squares on a 4-Variable K-Map

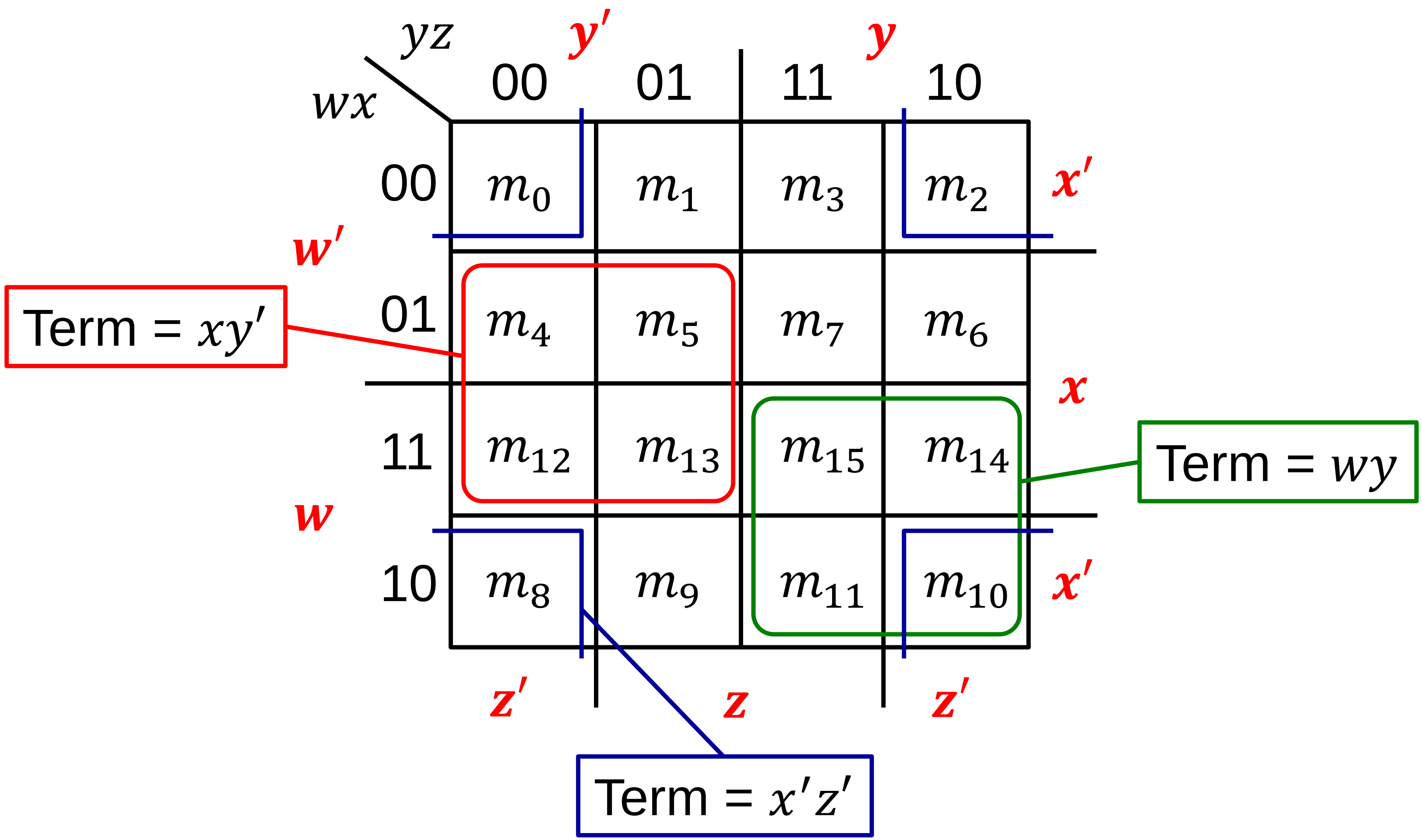
❖ On a 4-variable K-Map:

- ✧ One square represents a minterm with 4 variables
- ✧ Two adjacent squares represent a term with 3 variables
- ✧ Four adjacent squares represent a term with 2 variables
- ✧ Eight adjacent squares represent a term with 1 variable
- ✧ Combining all 16 squares is the constant '**1**' (no variables)

Combining Eight Squares



Combining Four Squares



Combining Two Squares

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

y' y

x' x x'

w' w

z' z z'

Term = $w'yz$

Term = $w'xy'$

Term = $wx'z'$

Term = $wy'z$

Simplifying a 4-Variable Function

Given $f(w, x, y, z) = \sum(0, 2, 4, 5, 6, 7, 8, 12)$

Draw the K-map for function f

Minimize f as sum-of-products

Solution:

$f = w'x + y'z' + w'z'$

Term = $w'z'$

$wx \backslash yz$	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	0	0
10	1	0	0	0

y' y x' w' x w x' z' z z'

Term = $w'x$

Term = $y'z'$

Example

❖ For the Boolean function

$$\boldsymbol{F = W'X'Y' + X'YZ' + W'XYZ' + WX'Y'}$$

a) Express the function as a sum-of-minters

b) Find the minimal sum-of-products expression

Solution:

a) $F = W'X'Y' + X'YZ' + W'XYZ' + WX'Y'$

$$= W'X'Y'(Z + Z') + (W + W')X'YZ' + W'XYZ' + WX'Y'(Z + Z')$$

$$= W'X'Y'Z + W'X'Y'Z' + WX'YZ' + W'X'YZ' + W'XYZ' +$$

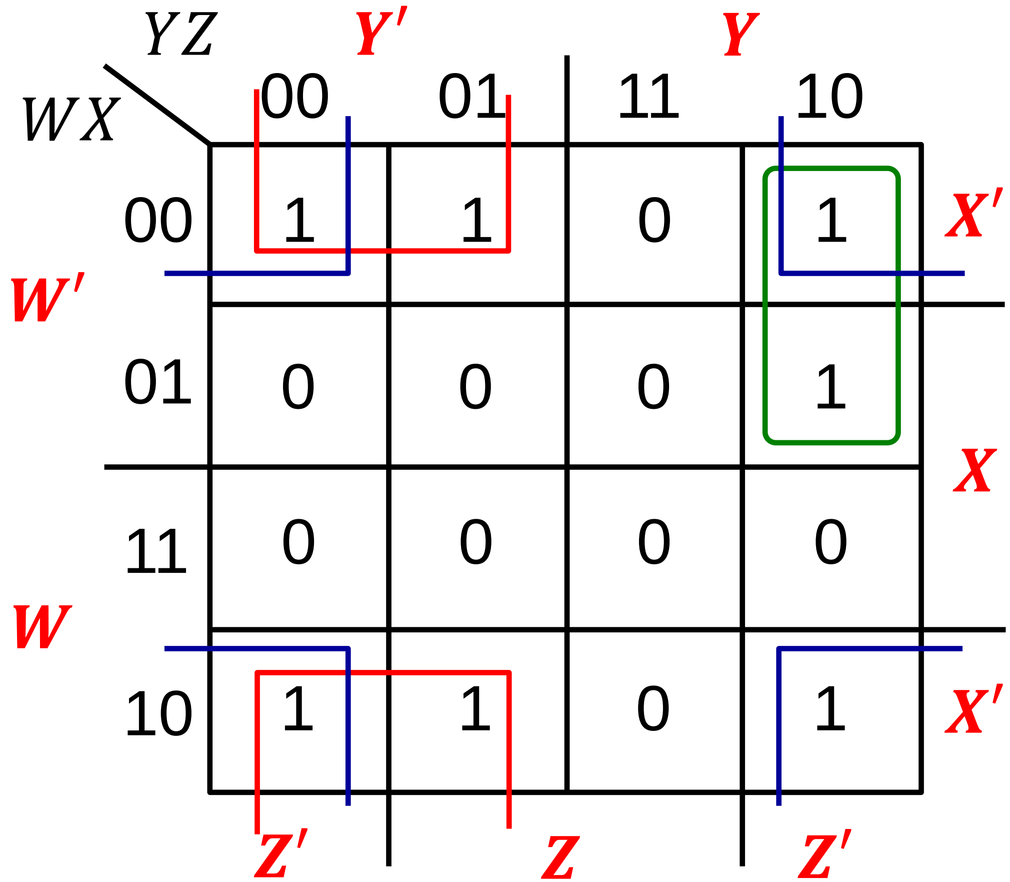
$$WX'Y'Z + WX'Y'Z'$$

$$= \sum(0, 1, 2, 6, 8, 9, 10)$$

Example (Cont.)

Solution:

b) $F = \sum(0, 1, 2, 6, 8, 9, 10)$



Red block: term = $X'Y'$

Blue block: term = $X'Z'$

Green block: term = $W'YZ'$

Minimal sum-of-products: $F = X'Y' + X'Z' + W'YZ'$ (7 literals)

Next . . .

- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- **Prime and Essential Prime Implicants**
- **Minimal Sum-of-Products and Product-of-Sums**
- Don't Cares
- Five and Six-Variable K-Maps
- Multiple Outputs
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- Odd and Even functions
- Parity Generators and Checkers

Prime Implicants

- **Prime Implicant:** a product term obtained by combining the **maximum number of adjacent squares** in the K-map

- The number of combined squares must be a **power of 2**

- **Essential Prime Implicant:** is a prime implicant that covers at least one minterm not covered by the other prime implicants

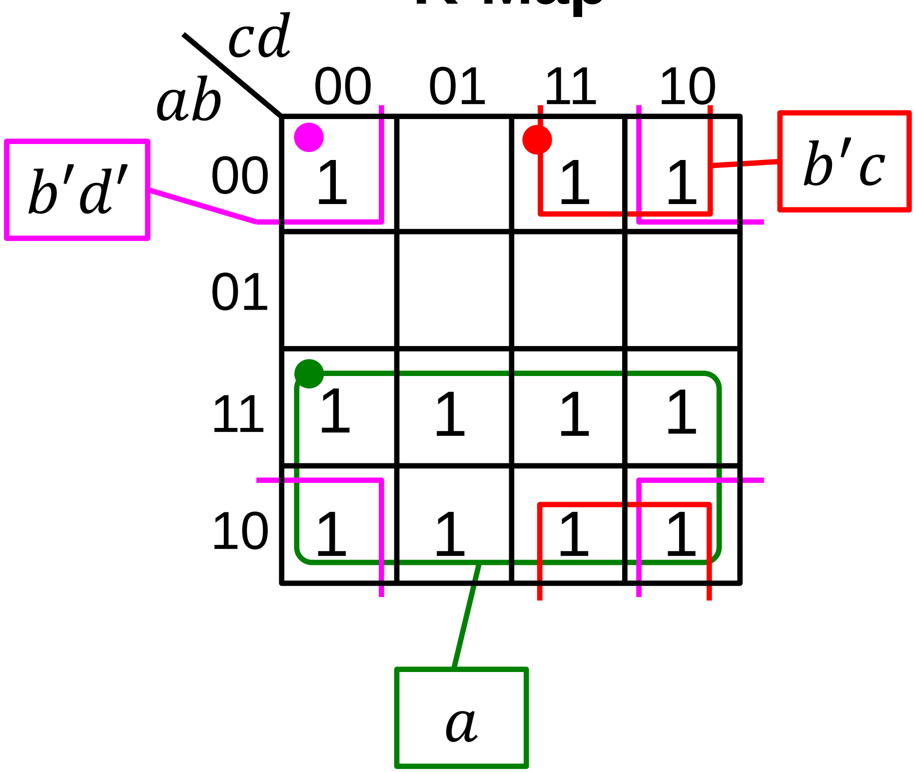
- The prime implicants and essential prime implicants can be determined by inspecting the K-map

Example of Prime Implicants

Find all the prime implicants and essential prime implicants for:

$$f(a,b,c,d) = \sum(0,2,3,8,9,10,11,12,13,14,15)$$

K-Map



Three Prime Implicants

$b'd'$, $b'c$, a

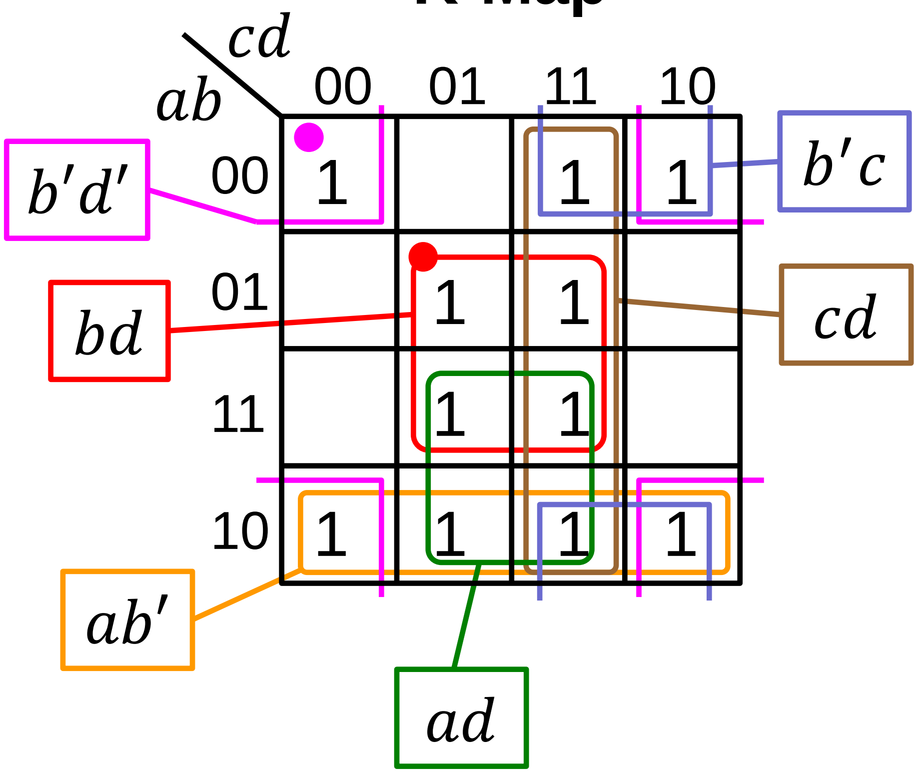
All Prime Implicants are essential

Example of Prime Implicants

Find all the prime implicants and essential prime implicants for:

$$f(a,b,c,d) = \sum(0,2,3,5,7,8,9,10,11,13,15)$$

K-Map



Six Prime Implicants

bd, $b'd'$, ab', ad, cd, $b'c$

Only Two Prime Implicants are essential bd and $b'd'$

Simplification Procedure Using the K-Map

1. Find all the essential prime implicants
   - ✧ Covering maximum number (power of 2) of 1's in the K-map
   - ✧ Mark the minterm(s) that make the prime implicants essential
2. Add prime implicants to cover the function
   - ✧ Choose a minimal subset of prime implicants that cover all remaining 1's
   - ✧ Make sure to cover all 1's not covered by the essential prime implicants
   - ✧ Minimize the overlap among the additional prime implicants

❖ Sometimes, a function has multiple simplified expressions

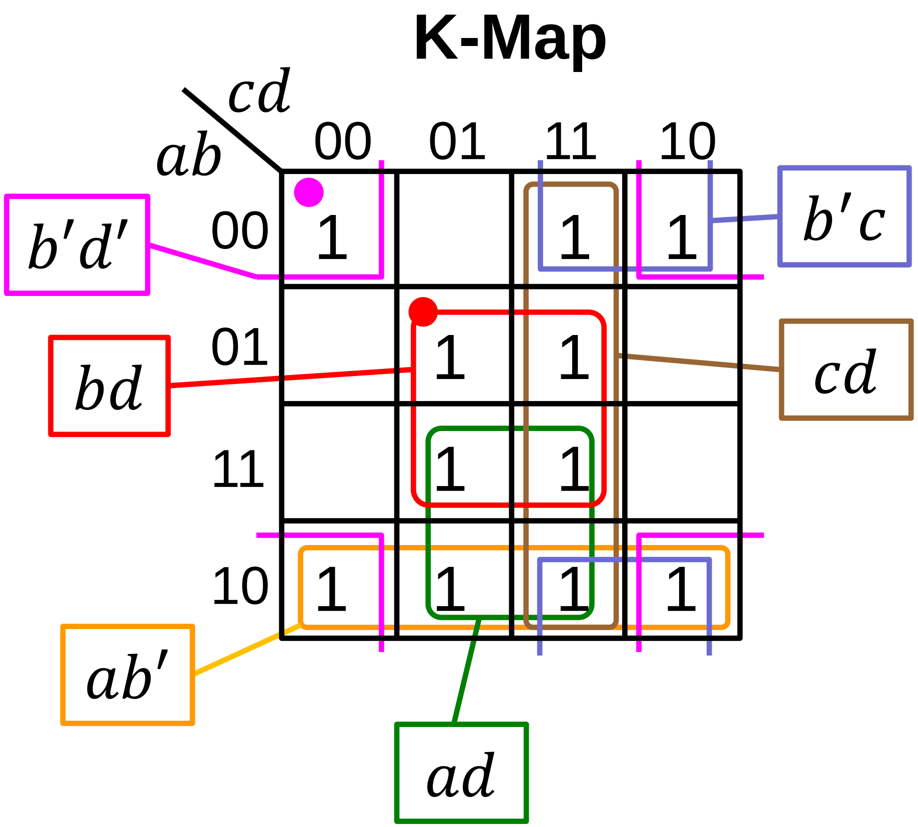
- ✧ You may be asked to list all the simplified sum-of-product expressions

Obtaining All Minimal SOP Expressions

Consider again: $f(a,b,c,d) = \sum(0,2,3,5,7,8,9,10,11,13,15)$

Obtain all minimal sum-of-products (SOP) expressions

K-Map



Two essential Prime Implicants: bd and $b'd'$

Four possible solutions:

$f = bd + b'd' + cd + ad$

$f = bd + b'd' + cd + ab'$

$f = bd + b'd' + b'c + ab'$

$f = bd + b'd' + b'c + ad$

Product-of-Sums (POS) Simplification

- All previous examples were expressed in Sum-of-Products form
- With a minor modification, the Product-of-Sums can be obtained
- Example: $f(a, b, c, d) = \sum(1, 2, 3, 9, 10, 11, 13, 14, 15)$

K-Map of f

$ab \backslash cd$	00	01	11	10
00		1	1	1
01				
11		1	1	1
10		1	1	1

$$f = ad + ac + b'd + b'c$$

Minimal Sum-of-Products = 8 literals

All prime implicants are essential

K-Map of f'

$ab \backslash cd$	00	01	11	10
00	1			
01	1	1	1	1
11	1			
10	1			

$$f' = c'd' + a'b$$

$$f = (c + d)(a + b') \rightarrow$$

Minimal Product-of-Sums = 4 literals

Simplification Procedure

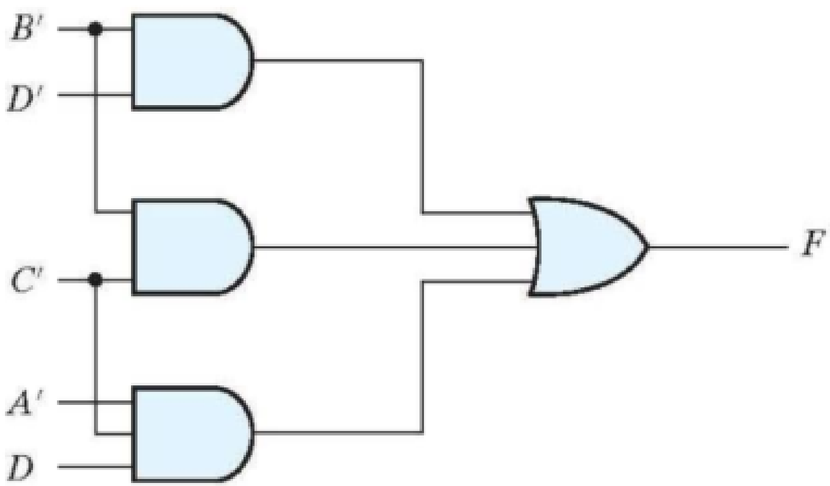
1. Draw the K-map for the function f
   - ✧ Obtain a minimal Sum-of-Products (SOP) expression for f
2. Draw the K-map for f', replacing the 0's of f with 1's in f'
3. Obtain a minimal Sum-of-Products (SOP) expression for f'
4. Use DeMorgan's theorem to obtain $f = (f')'$
   - ✧ The result is a minimal Product-of-Sums (POS) expression for f
5. Compare the cost of the minimal SOP and POS expressions
   - ✧ Count the number of literals to find which expression is minimal

Example

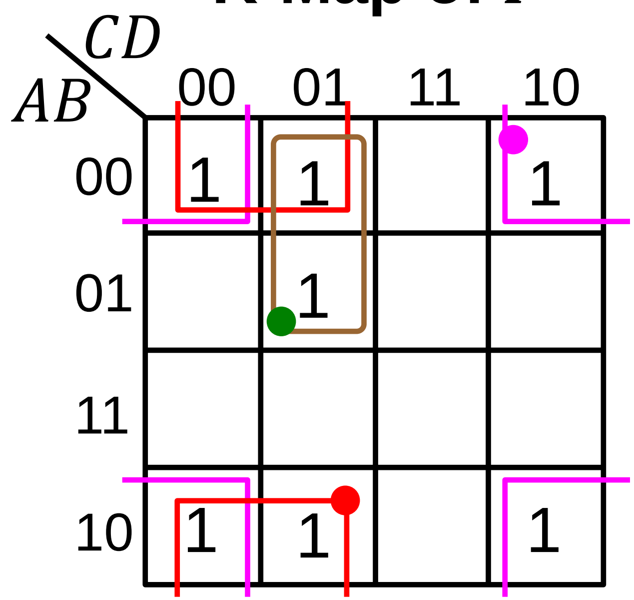
❖ Express the Boolean function f in standard form using the minimal number of literals

$$F(A,B,C,D) = \prod(3,4,6,7,11,12,13,14,15)$$

a) Simplify the function in sum-of-products form



K-Map of *F*



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

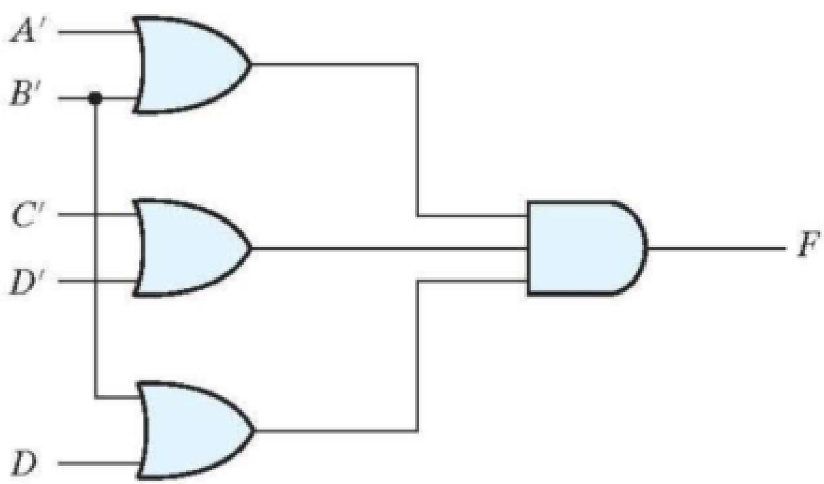
Minimal Sum-of-Products = 7 literals

Example (Cont.)

❖ Express the Boolean function f in standard form using the minimal number of literals

$$F(A,B,C,D) = \prod(3,4,6,7,11,12,13,14,15)$$

b) Simplify the function in product-of-sums form



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

Minimal Product-of-Sums = 6 literals

K-Map of F'

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

$$F' = CD + BD' + AB$$

Next . . .

- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- **Don't Cares**
- Five and Six-Variable K-Maps
- Multiple Outputs
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- Odd and Even functions
- Parity Generators and Checkers

Don't Cares

- Sometimes, a function table may contain entries for which:
  - The input values of the variables will never occur, or
  - The output value of the function is never used
- In this case, the output value of the function is not defined
- The output value of the function is called a **don't care**
- A don't care is an **X** value that appears in the function table
- The **X** value can be later chosen to be **0 or 1**
  - To minimize the function implementation

Example of a Function with Don't Cares

❖ Consider a function f defined over BCD inputs

❖ The function input is a BCD digit from 0 to 9

❖ The function output is 0 if the BCD input is 0 to 4

❖ The function output is 1 if the BCD input is 5 to 9

❖ The function output is X (don't care) if the input is 10 to 15 (not BCD)

❖ $f = \underbrace{\sum_m (5, 6, 7, 8, 9)}_{\text{Minterms}} + \underbrace{\sum_d (10, 11, 12, 13, 14, 15)}_{\text{Don't Cares}}$

Truth Table

a b c d	f
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1
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

Minimizing Functions with Don't Cares

Consider: $f = \sum_m(5,6,7,8,9) + \sum_d(10,11,12,13,14,15)$

If the don't cares were treated as 0's we get:

$f = a'bd + a'bc + ab'c'$ (9 literals)

If the don't cares were treated as 1's we get:

$f = a + bd + bc$ (5 literals)

The don't care values can be selected to be either 0 or 1, to produce a minimal expression

K-Map of f

$ab \backslash cd$	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

Simplification Procedure with Don't Cares

1. Find all the essential prime implicants
   - Covering maximum number (power of 2) of 1's and X's (don't cares)
   - Mark the 1's that make the prime implicants essential
2. Add prime implicants to cover the function
   - Choose a minimal subset of prime implicants that cover all remaining 1's
   - Make sure to cover all 1's not covered by the essential prime implicants
   - Minimize the overlap among the additional prime implicants
   - You need not cover all the don't cares (some can remain uncovered)

- Sometimes, a function has multiple simplified expressions

Minimizing Functions with Don't Cares (2)

Simplify the function $g(a,b,c,d) = \sum_m(1,3,7,11,15)$ which has the don't care conditions $d(a,b,c,d) = \sum_d(0,2,5)$

Solution 1: $g = cd + a'b'$ (4 literals)

Solution 2: $g = cd + a'd$ (4 literals)

Prime Implicant cd is essential

Not all don't cares need be covered

K-Map of g

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

K-Map of g

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

Minimal Product-of-Sums with Don't Cares

Simplify: $g = \sum_m(1, 3, 7, 11, 15) + \sum_d (0, 2, 5)$

Obtain a minimal product-of-sums expression

Solution: $g' = \sum_m(4, 6, 8, 9, 10, 12, 13, 14) + \sum_d (0, 2, 5)$

Minimal $g' = d' + ac'$ (3 literals)

Minimal product-of-sums:

$g = d(a' + c)$ (3 literals)

The minimal sum-of-products expression for g had 4 literals

K-Map of g'

$ab \backslash cd$	00	01	11	10
00	X	0	0	X
01	1	X	0	1
11	1	1	0	1
10	1	1	0	1

Next . . .

- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- Don't Cares
- **Five and Six-Variable K-Maps**
- Multiple Outputs
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- Odd and Even functions
- Parity Generators and Checkers

Five-Variable Karnaugh Map

- Consists of $2^5 = 32$ squares, numbered 0 to 31
  - Remember the numbering of squares in the K-map
- Can be visualized as two layers of 16 squares each
- Top layer contains the squares of the first 16 minterms ($a = 0$)
- Bottom layer contains the squares of the last 16 minterms ($a = 1$)

$a = 0$

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

$a = 1$

$bc \backslash de$	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

Each square is adjacent to **5** other squares:
4 in the same layer and
1 in the other layer:
m_0 is adjacent to m_{16}
m_1 is adjacent to m_{17}
m_4 is adjacent to m_{20} …

Example of a Five-Variable K-Map

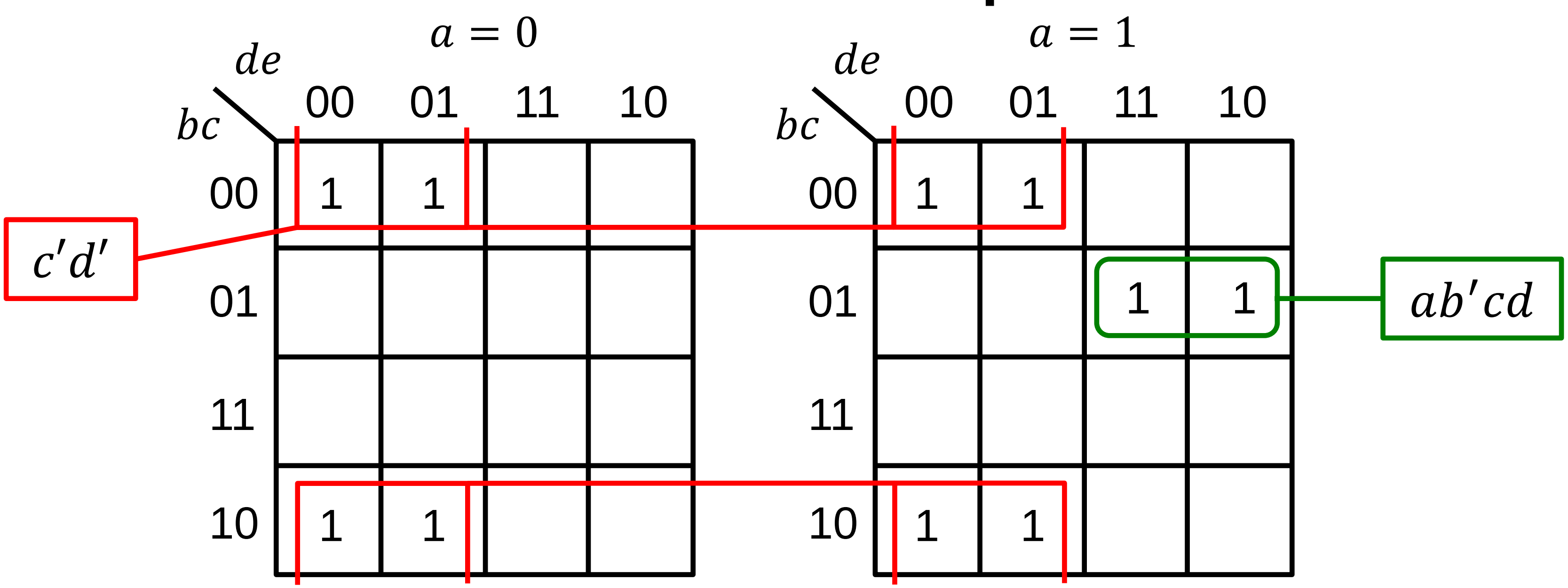
Given: $f(a,b,c,d,e) = \sum(0,1,8,9,16,17,22,23,24,25)$

Draw the 5-Variable K-Map

Obtain a minimal Sum-of-Products expression for f

Solution: $f = c'd' + ab'cd$ (6 literals)

5-Variable K-Map



Five-Variable K-Map with Don't Cares

$g(a,b,c,d,e) = \sum_m(3,6,7,11,24,25,27,28,29) + \sum_d(2,8,9,12,13,26)$

Draw the 5-Variable K-Map

Obtain a minimal Sum-of-Products expression for g

Solution: $g = bd' + a'b'd + bc'e$ (8 literals)

5-Variable K-Map

$a = 0$

$bc \backslash de$	00	01	11	10
00			1	X
01			1	1
11	X	X		
10	X	X	1	

$a = 1$

$bc \backslash de$	00	01	11	10
00				
01				
11	1	1		
10	1	1	1	X

$a'b'd$

bd'

$bc'e$

Not covered

All prime implicants are essential

Six-Variable Karnaugh Map

- Consists of $2^6 = 64$ squares, numbered 0 to 63
- Can be visualized as four layers of 16 squares each
  - Four layers: $ab = 00, 01, 11, 10$ (Notice that layer 11 comes before 10)
- Each square is adjacent to 6 other squares:
  - 4 squares in the same layer and 2 squares in the above and below layers

	$ab = 00$				$ab = 01$				$ab = 11$				$ab = 10$			
$cd \backslash ef$	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	m_0	m_1	m_3	m_2	m_{16}	m_{17}	m_{19}	m_{18}	m_{48}	m_{49}	m_{51}	m_{50}	m_{32}	m_{33}	m_{35}	m_{34}
01	m_4	m_5	m_7	m_6	m_{20}	m_{21}	m_{23}	m_{22}	m_{52}	m_{53}	m_{55}	m_{54}	m_{36}	m_{37}	m_{39}	m_{38}
11	m_{12}	m_{13}	m_{15}	m_{14}	m_{28}	m_{29}	m_{31}	m_{30}	m_{60}	m_{61}	m_{63}	m_{62}	m_{44}	m_{45}	m_{47}	m_{46}
10	m_8	m_9	m_{11}	m_{10}	m_{24}	m_{25}	m_{27}	m_{26}	m_{56}	m_{57}	m_{59}	m_{58}	m_{40}	m_{41}	m_{43}	m_{42}

Example of a Six-Variable K-Map

$h(a, b, c, d, e, f) = \sum(2, 10, 11, 18, 21, 23, 29, 31, 34, 41, 50, 53, 55, 61, 63)$

Draw the 6-Variable K-Map

Obtain a minimal Sum-of-Products expression for h

Solution: $h = c'd'ef' + b\,d\,f + a'b'c\,d'e + a\,b'\,c\,d'e'f$ (18 literals)

$cd \backslash ef$	$ab = 00$: 00	01	11	10	$ab = 01$: 00	01	11	10	$ab = 11$: 00	01	11	10	$ab = 10$: 00	01	11	10
00				1				1				1				1
01						1	1			1	1					
11						1	1			1	1					
10			1	1										1		

$c'd'ef'$

bdf

$a'b'c\,d'e$

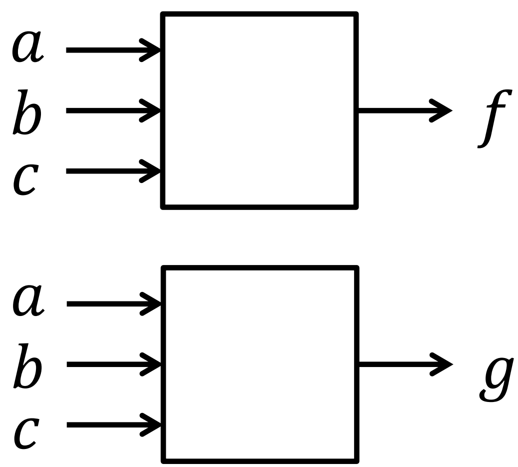
$a\,b'\,c\,d'e'f$

Next . . .

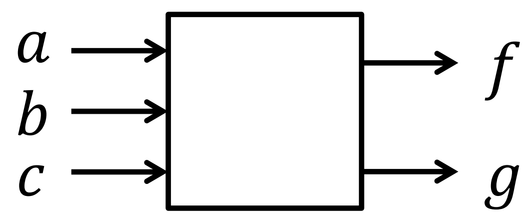
- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- Don't Cares
- Five and Six-Variable K-Maps
- **Multiple Outputs**
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- Odd and Even functions
- Parity Generators and Checkers

Multiple Outputs

- Suppose we have two functions: $f(a, b, c)$ and $g(a, b, c)$
- Same inputs: a, b, c, but two outputs: f and g
- We can minimize each function separately, or
- Minimize f and g as one circuit with two outputs
- The idea is to share terms (gates) among f and g



Two separate circuits



One circuit with Two Outputs

Multiple Outputs: Example 1

Given: $f(a,b,c) = \sum(0,2,6,7)$ and $g(a,b,c) = \sum(1,3,6,7)$

Minimize each function separately

Minimize both functions as one circuit

K-Map of f

$a \backslash bc$	00	01	11	10
0	1	0	0	1
1	0	0	1	1

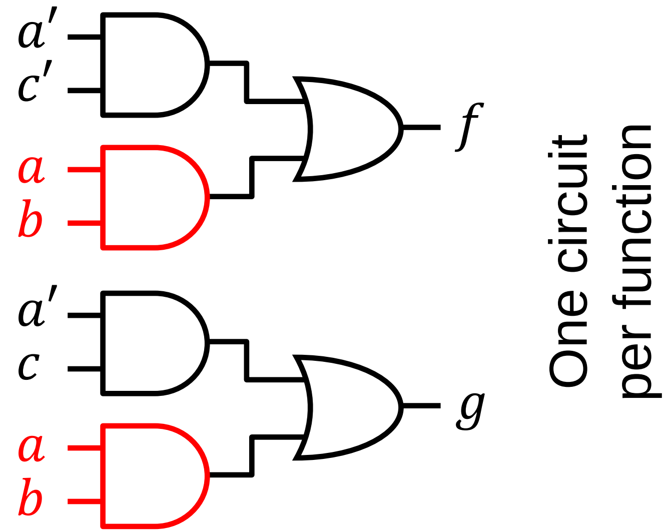
$$f = a'c' + ab$$

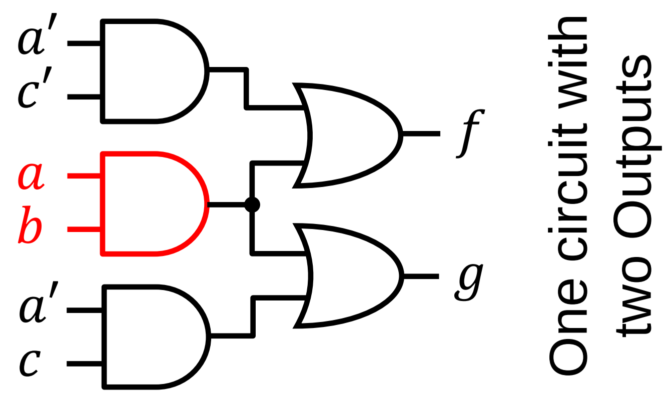
K-Map of g

$a \backslash bc$	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$g = a'c + ab$$

Common Term = ab





Multiple Outputs: Example 2

$f(a,b,c,d) = \sum(3,5,7,10,11,14,15)$, $g(a,b,c,d) = \sum(1,3,5,7,10,14)$

Draw the K-map and write minimal SOP expressions of f and g

$f = a'bd + ac + cd$ $\qquad$ $g = a'd + acd'$

Extract the common terms of f and g

K-Map of *f*

$ab \backslash cd$	00	01	11	10
00			1	
01		1	1	
11			1	1
10			1	1

K-Map of *g*

$ab \backslash cd$	00	01	11	10
00		1	1	
01		1	1	
11				1
10				1

Common Terms

$T_1 = a'd$ and $T_2 = ac$

Minimal f and g

$$f = T_1 b + T_2 + cd$$

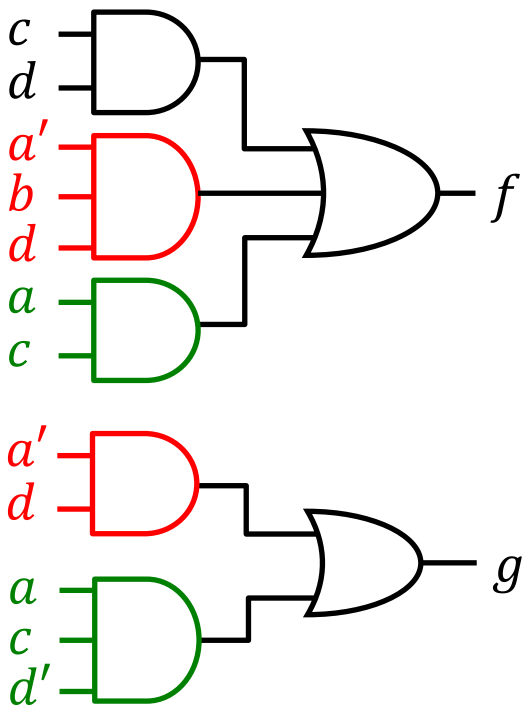
$$g = T_1 + T_2 d'$$

Common Terms ➔ Shared Gates

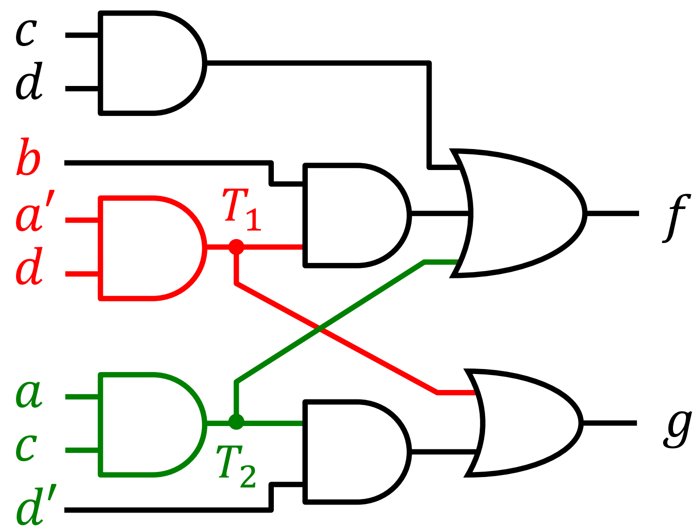
Minimal $f = a'bd + ac + cd$ Minimal $g = a'd + acd'$

Let $T_1 = a'd$ and $T_2 = ac$ (shared by f and g)

Minimal $f = T_1 b + T_2 + cd$, Minimal $g = T_1 + T_2 d'$



NO Shared Gates



One Circuit

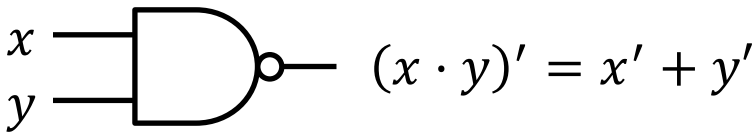
Two Shared Gates

Next . . .

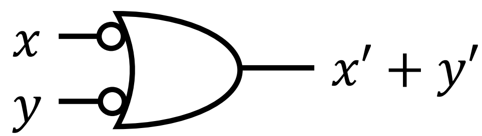
- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- Don't Cares
- Five and Six-Variable K-Maps
- Multiple Outputs
- **Universality of NAND and NOR gates**
- **NAND-NAND and NOR-NOR implementations**
- Odd and Even functions
- Parity Generators and Checkers

NAND Gate

- The NAND gate has the following symbol and truth table
- NAND represents **N**OT **AND**
- The small bubble circle represents the invert function



NAND gate



Another symbol for NAND

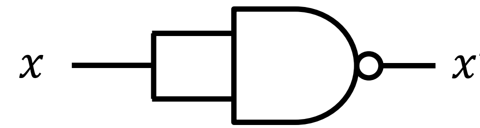
x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

- NAND gate is implemented efficiently in CMOS technology
  - In terms of chip area and speed

The NAND Gate is Universal

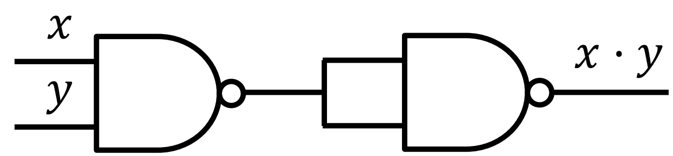
- NAND gates can implement any Boolean function
- NAND gates can be used as inverters, or to implement AND/OR
- A single-input NAND gate is an inverter

x NAND $x = (x \cdot x)' = x'$



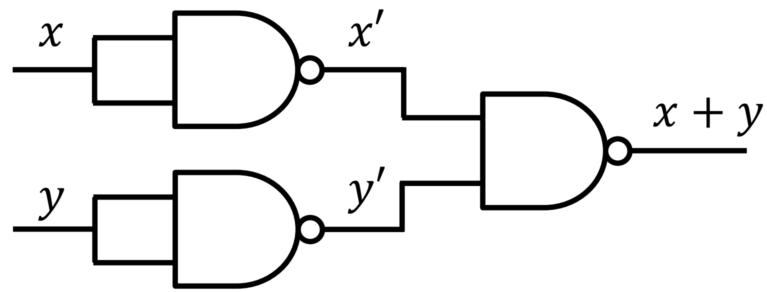
- AND is equivalent to NAND with **inverted output**

$(x$ NAND $y)' = ((x \cdot y)')' = x \cdot y$ (AND)



- OR is equivalent to NAND with **inverted inputs**

$(x'$ NAND $y') = (x' \cdot y')' = x + y$ (OR)

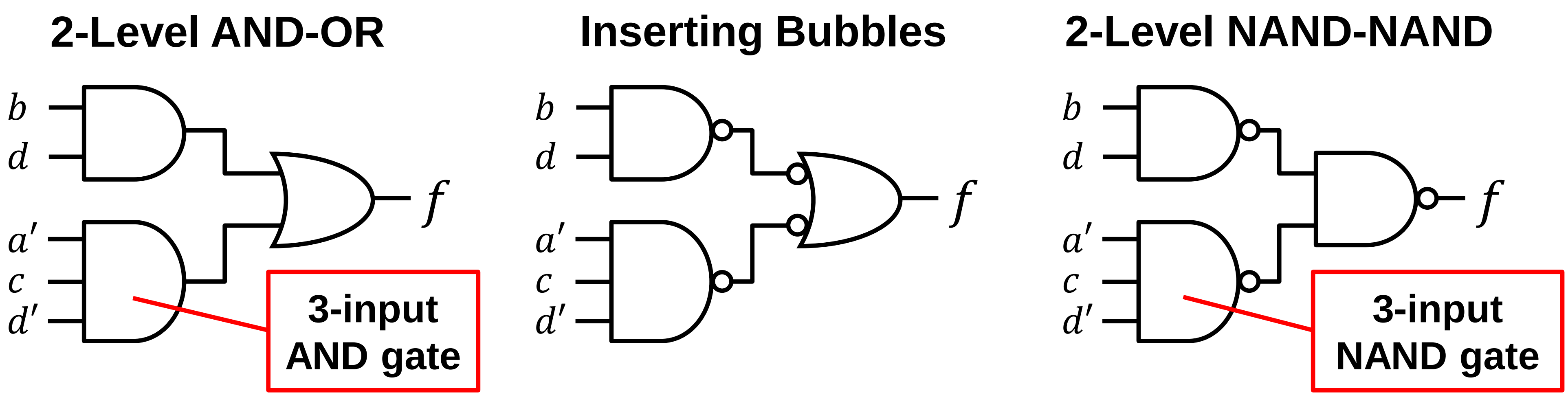


NAND - NAND Implementation

❖ Consider the following **sum-of-products** expression:

$$f = bd + a'cd'$$

❖ A 2-level **AND-OR** circuit can be converted easily to a 2-level **NAND-NAND implementation**



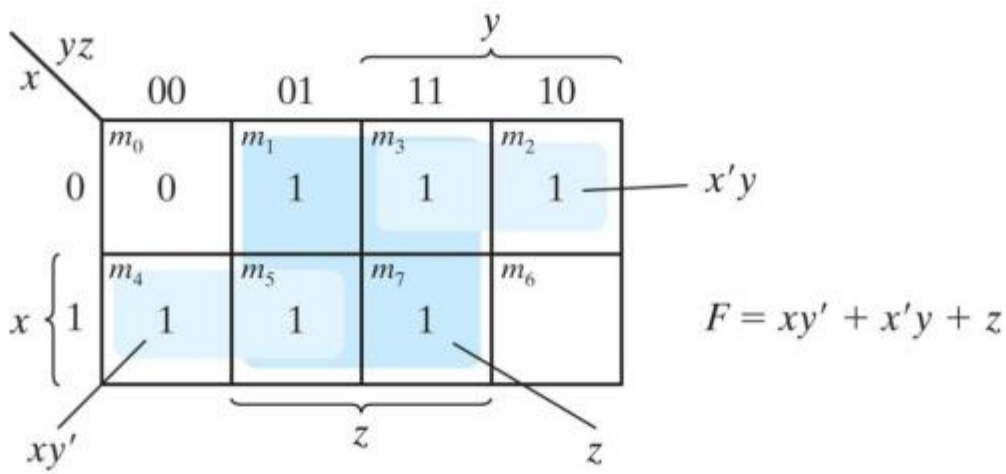
Two successive bubbles on same line cancel each other

Boolean Function with NAND Gates

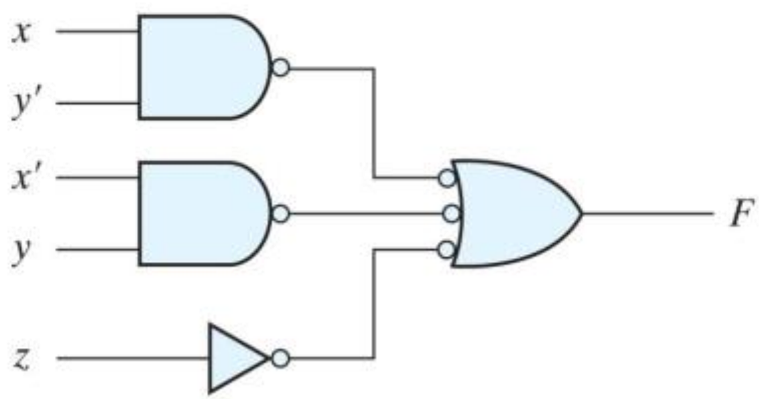
- Example: Implement the Boolean function

  $f(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ using <u>only</u> **NAND** gates

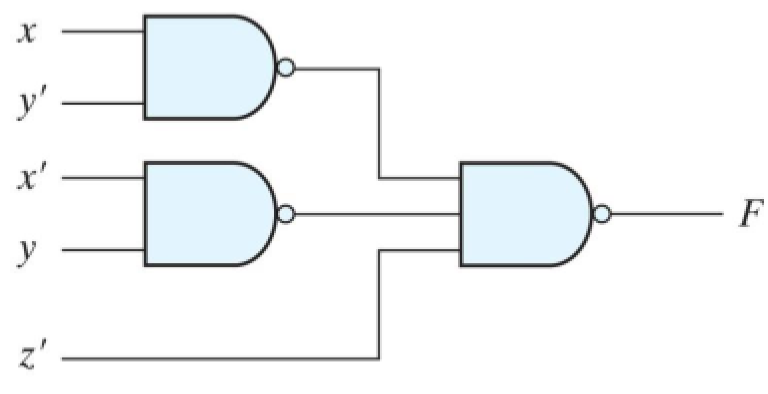
- Solution:



(a)



(b)



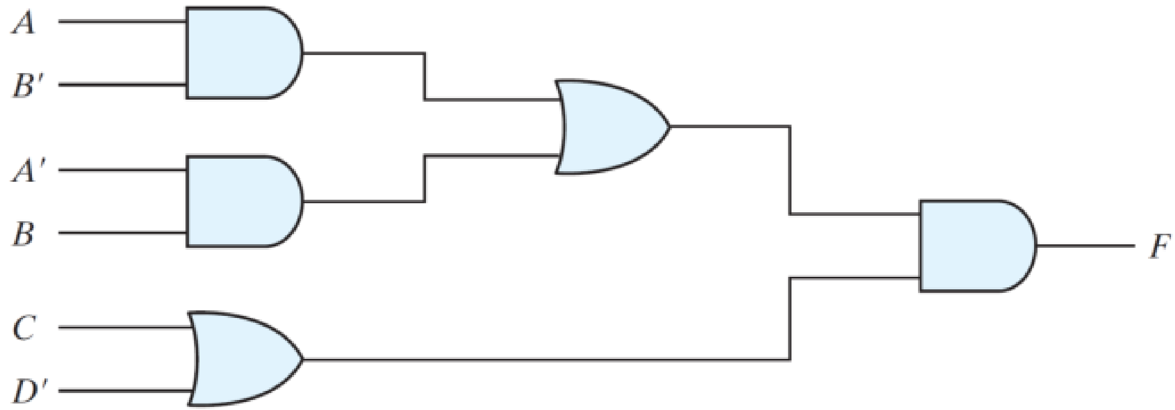
(c)

Multilevel Circuits using NAND Gates

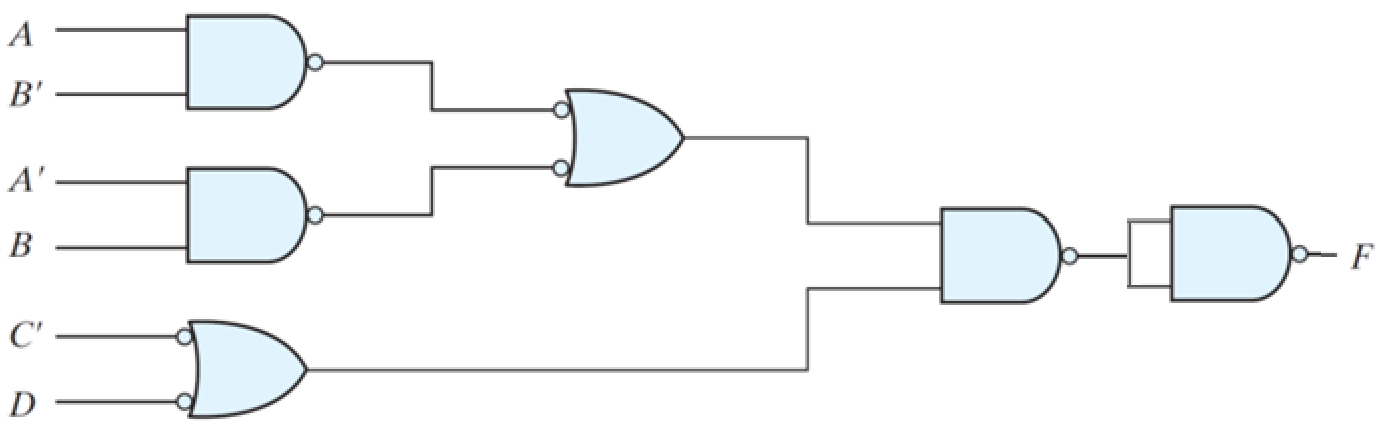
- General Procedure for converting a multilevel AND–OR diagram into an all-NAND diagram using mixed notation is as follows:
  - Convert all AND gates to NAND gates with AND-invert graphic symbols.
  - Convert all OR gates to NAND gates with invert-OR graphic symbols.
  - Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.

Multilevel Circuits using NAND Gates

❖ Example: Implement the given circuit using <u>only</u> **NAND** gates



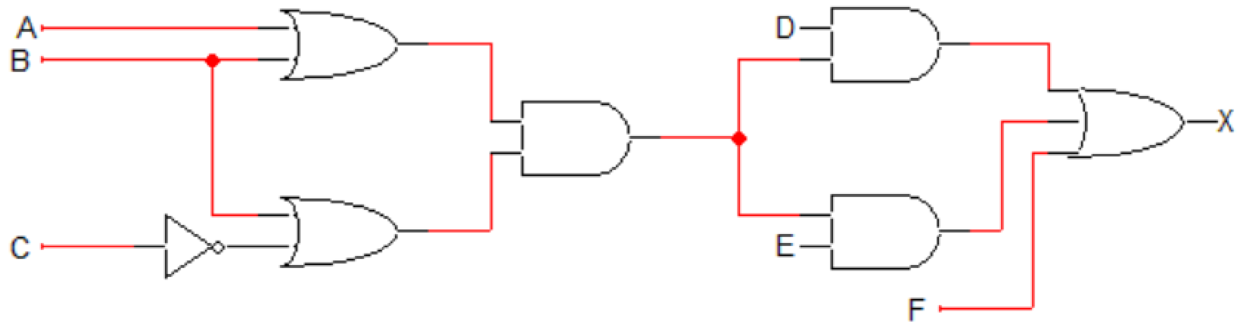
❖ Solution:



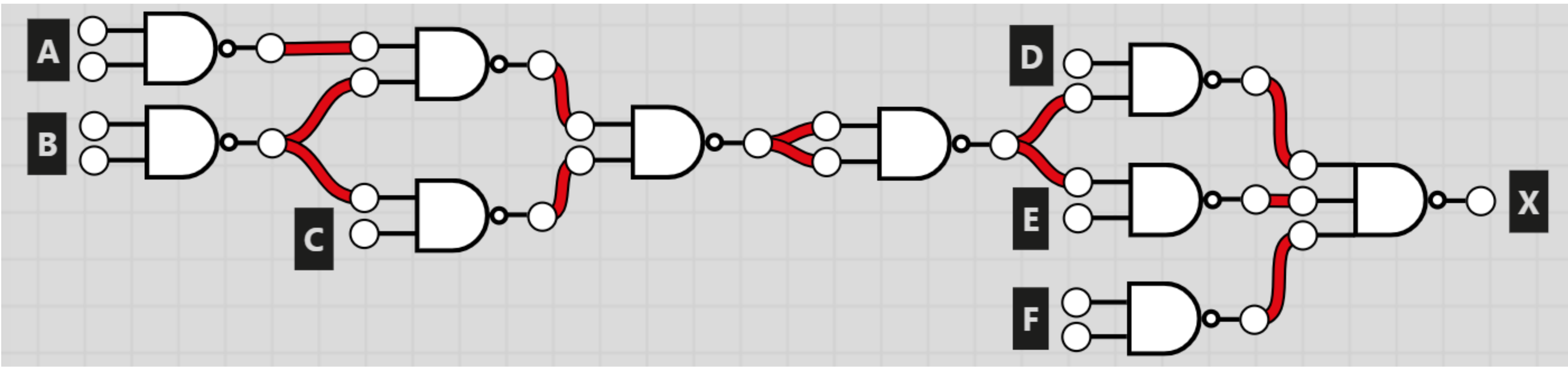
Start from output toward inputs converting gate by gate

Multilevel Circuits using NAND Gates

❖ Example: Implement the given circuit using <u>only</u> **NAND** gates



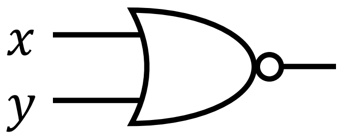
❖ Solution:



Start from output toward inputs converting gate by gate

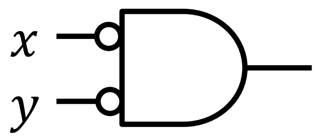
NOR Gate

- The NOR gate has the following symbol and truth table
- NOR represents **N**OT **OR**
- The small bubble circle represents the invert function



$(x + y)' = x' \cdot y'$

NOR gate



$x' \cdot y'$

Another symbol for NOR

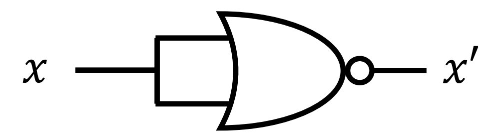
x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- NOR gate is implemented efficiently in CMOS technology
  - In terms of chip area and speed

The NOR Gate is also Universal

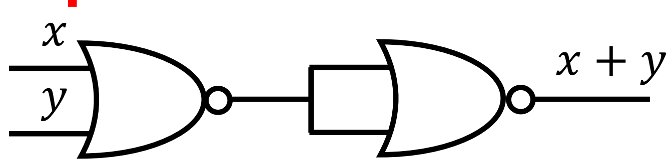
- NOR gates can implement any Boolean function
- NOR gates can be used as inverters, or to implement AND/OR
- A single-input NOR gate is an inverter

x NOR x = $(x + x)'$ = x'



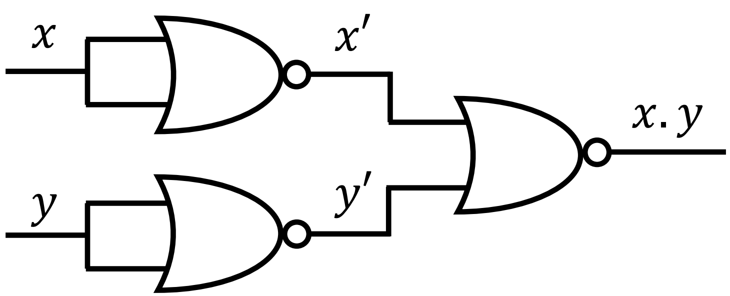
- OR is equivalent to NOR with **inverted output**

$(x$ NOR $y)'$ = $((x + y)')'$ = $x + y$ (OR)



- AND is equivalent to NOR with **inverted inputs**

$(x'$ NOR $y')$ = $(x' + y')'$ = $x \cdot y$ (AND)

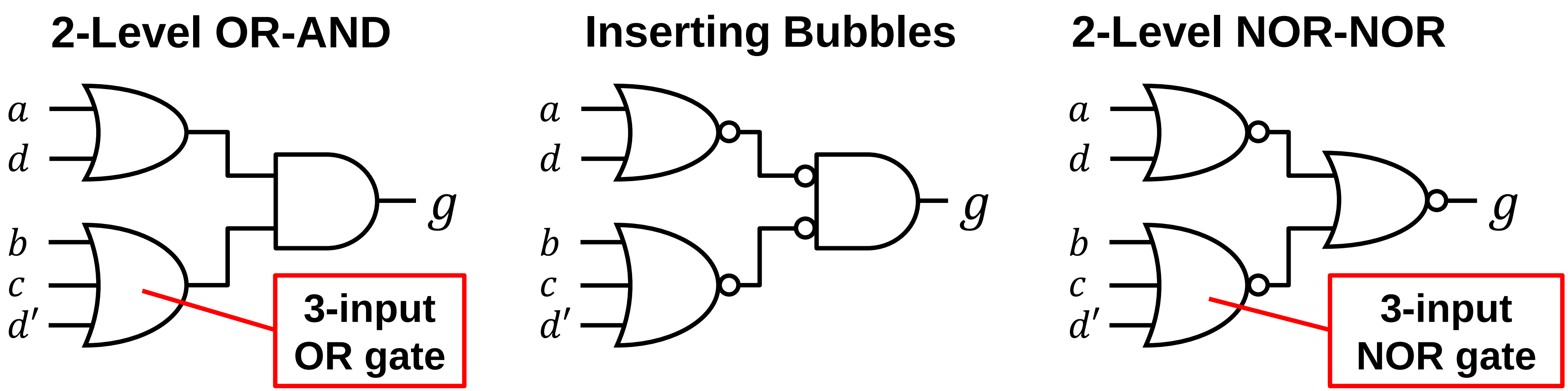


NOR - NOR Implementation

❖ Consider the following **product-of-sums** expression:

$$g = (a + d)(b + c + d')$$

❖ A 2-level **OR-AND** circuit can be converted easily to a 2-level **NOR-NOR implementation**



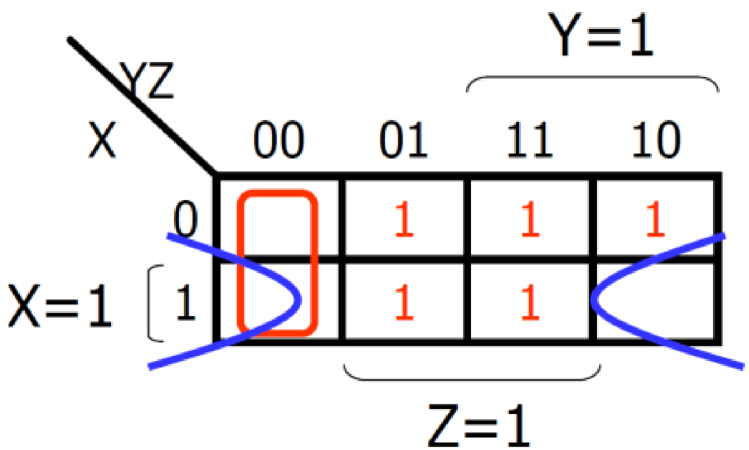
Two successive bubbles on same line cancel each other

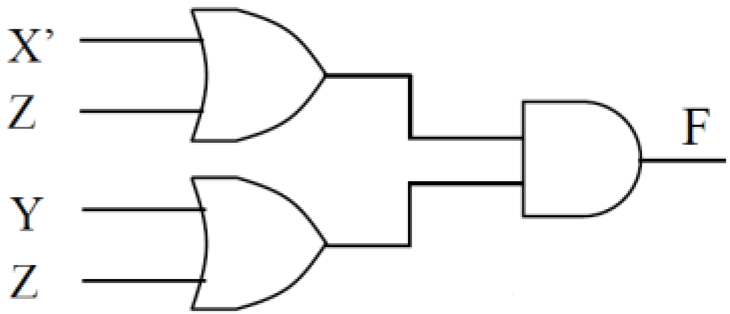
Boolean Function with NOR Gates

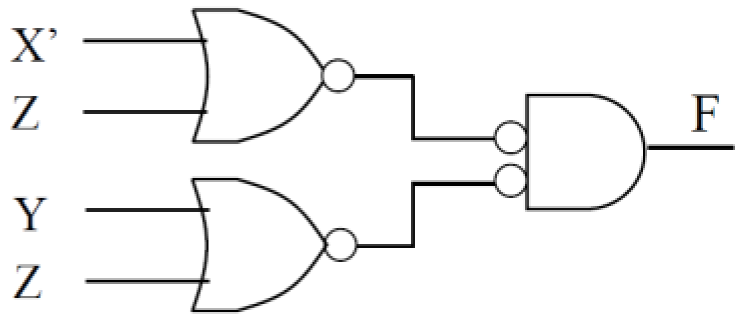
- Example: Implement the Boolean function

  $f(x, y, z) = \sum(1, 2, 3, 5, 7)$ using only **NOR** gates

- Solution:







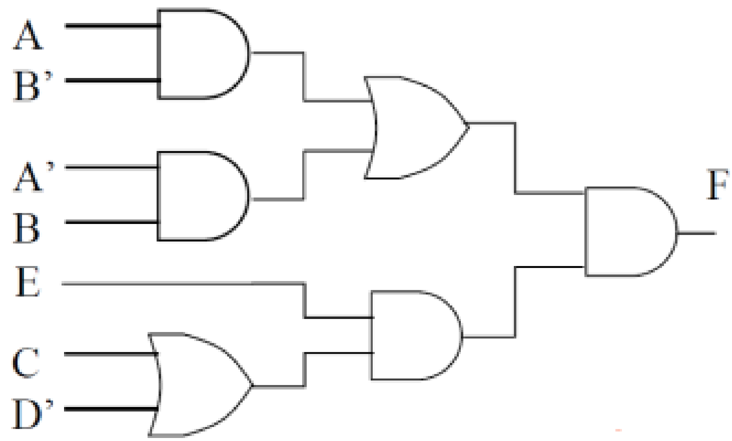
Multilevel Circuits using NOR Gates

- General Procedure for converting a multilevel OR–AND diagram into an all-NOR diagram using mixed notation is as follows:
  - Convert all OR gates to NOR gates with OR-invert graphic symbols.
  - Convert all AND gates to NOR gates with invert-AND graphic symbols.
  - Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NOR gate) or complement the input literal.

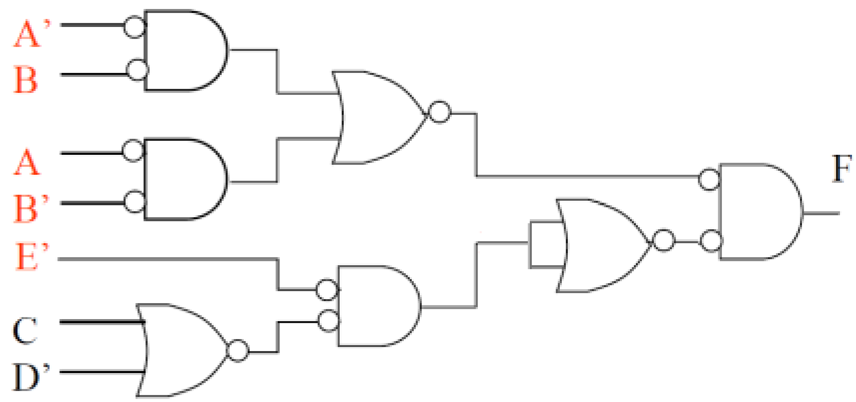
Multilevel Circuits using NOR Gates

❖ Example: Implement the Boolean function

$f(A, B, C, D, E) = (AB' + A'B)E(C + D')$ using <u>only</u> **NOR** gates



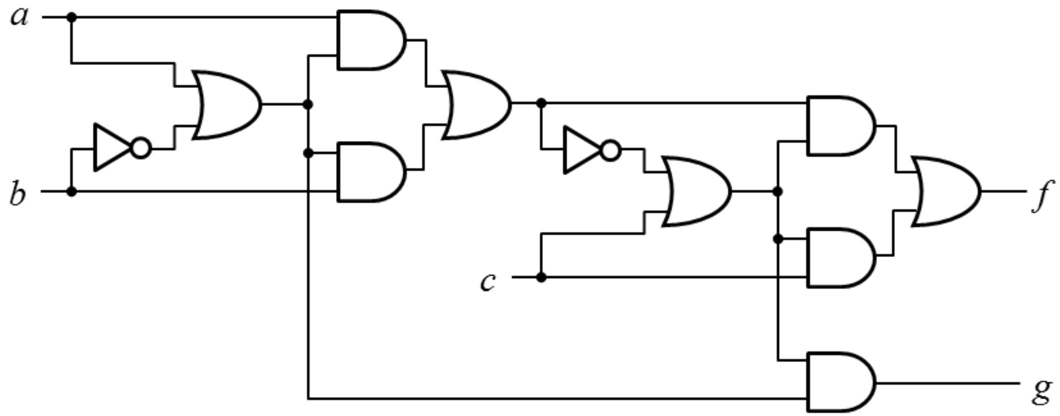
❖ Solution:



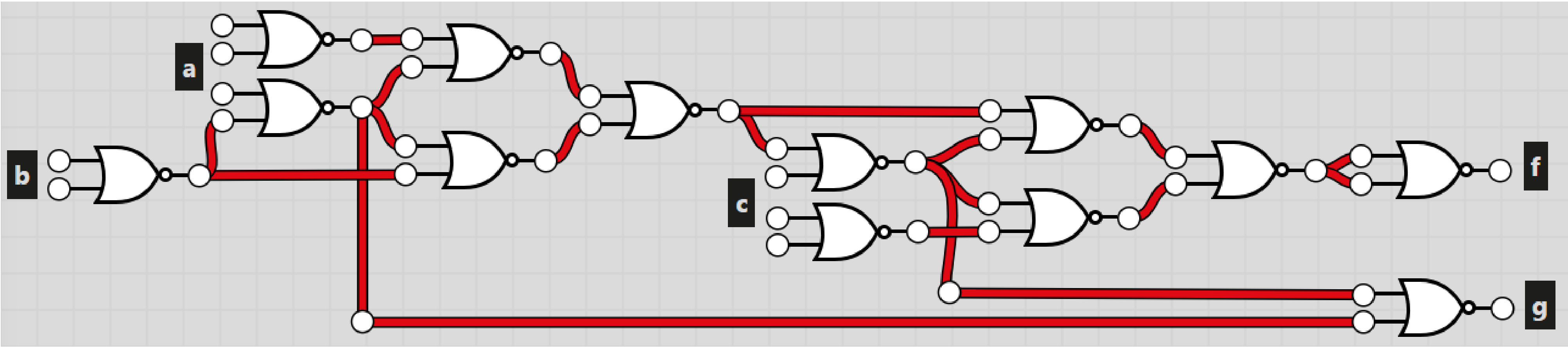
Start from output toward inputs converting gate by gate

Multilevel Circuits using NOR Gates

❖ Example: Implement the given circuit using <u>only</u> **NOR** gates



❖ Solution:



Start from output toward inputs converting gate by gate

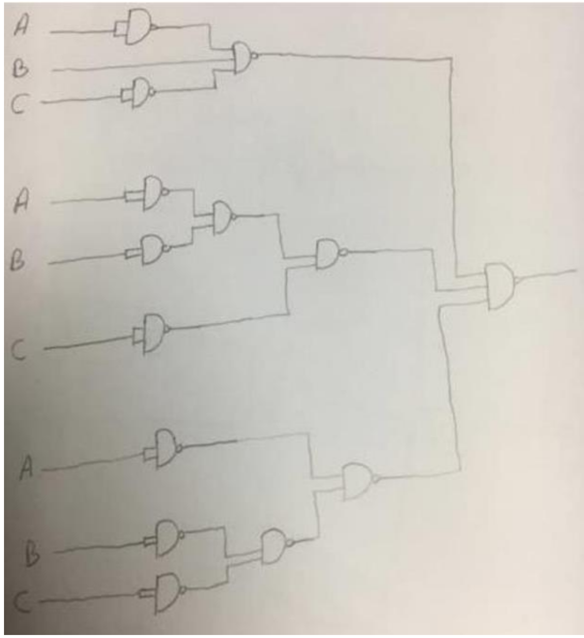
Multilevel Circuits using NAND/NOR Gates

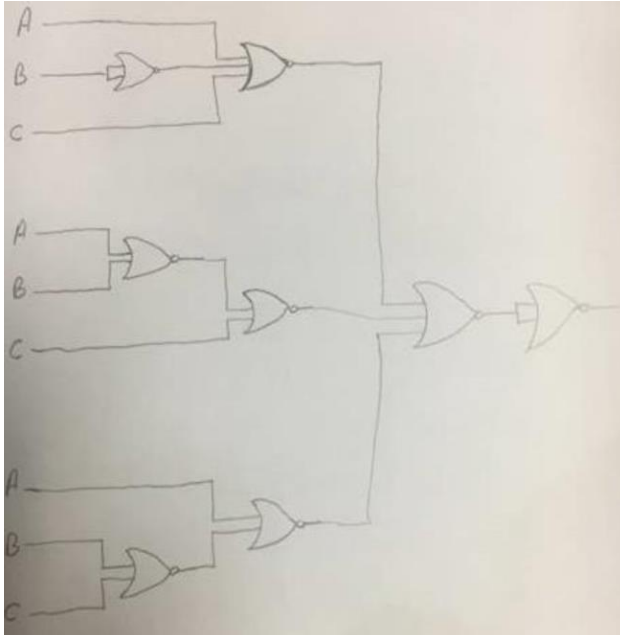
- Example: Find the complement of the following expression and implement it using (1) NAND gates, and (2) NOR gates:

$$G(A, B, C) = (A + B' + C)(A'B' + C)(A + B'C')$$

- Solution:

$$G' = ((A + B' + C)(A'B' + C)(A + B'C'))' = A'BC' + C'(A + B) + A'(B + C)$$





Next . . .

- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- Don't Cares
- Five and Six-Variable K-Maps
- Multiple Outputs
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- **Odd and Even functions**
- **Parity Generators and Checkers**

Exclusive OR / Exclusive NOR

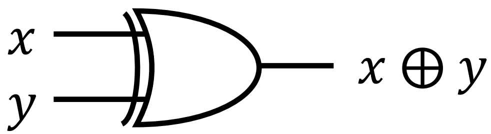
❖ Exclusive OR (XOR) is an important Boolean operation used extensively in logic circuits

❖ Exclusive NOR (XNOR) is the complement of XOR

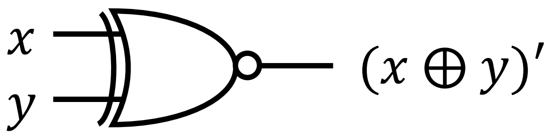
x y	XOR
0 0	0
0 1	1
1 0	1
1 1	0

x y	XNOR
0 0	1
0 1	0
1 0	0
1 1	1

XNOR is also known as **equivalence**



XOR gate



XNOR gate

Odd Function

- Output is 1 if the **number of 1's is odd in the inputs**
- Output is the XOR operation on all input variables

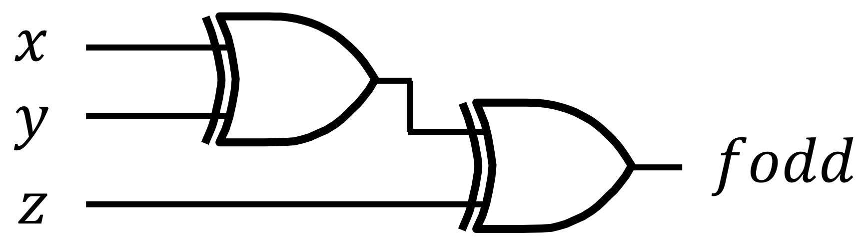
Odd Function with 3 inputs

x y z	fodd
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

$$fodd = \sum(1,2,4,7)$$

$$fodd = x'y'z + x'yz' + xy'z' + xyz$$

$$fodd = x \oplus y \oplus z$$



Implementation using two XOR gates

Even Function

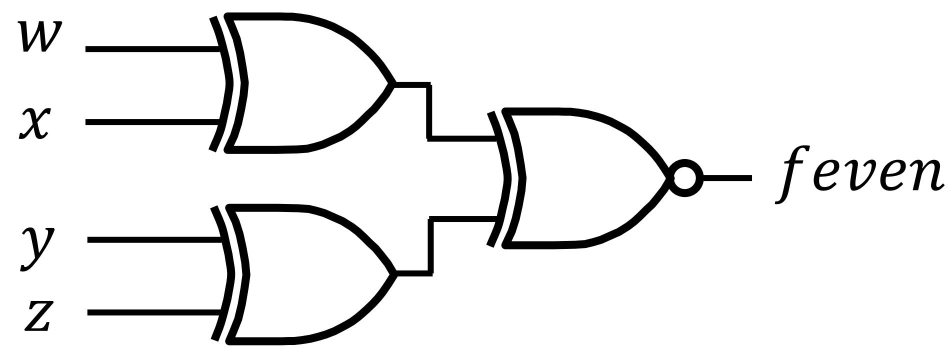
Even Function with 4 inputs

w x y z	feven
0 0 0 0	1
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	1

- Output is 1 if the **number of 1's is even** in the inputs (complement of odd function)
- Output is the XNOR operation on all inputs

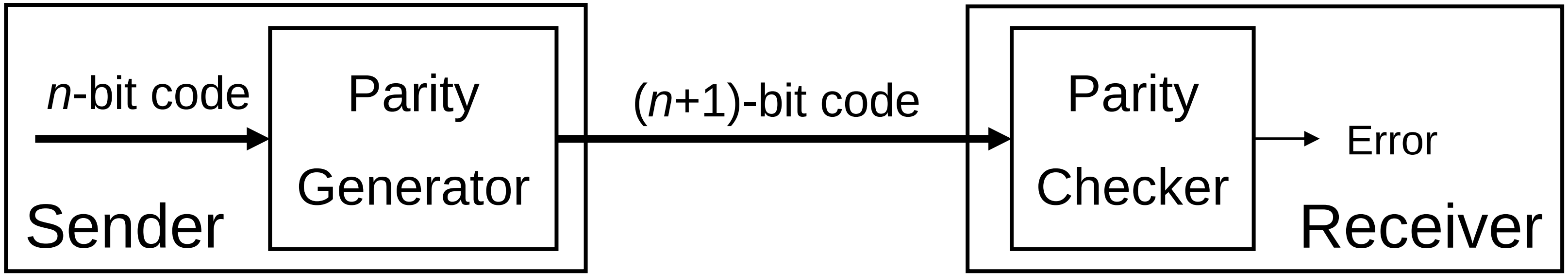
$$feven = \sum(0, 3, 5, 6, 9, 10, 12, 15)$$

$$feven = (w \oplus x \oplus y \oplus z)'$$



Implementation using two XOR gates and one XNOR

Parity Generators and Checkers



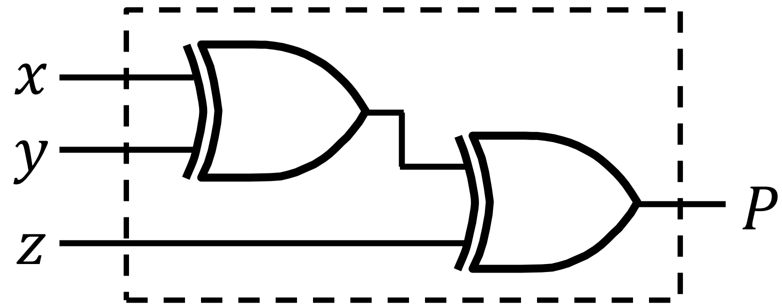
- A parity bit is added to the *n*-bit code
  - Produces (*n*+1)-bit code with an odd (or even) count of 1's
- **Odd parity:** count of 1's in the (*n*+1)-bit code is **odd**
  - Use an **even function** to generate the **odd parity bit**
  - Use an **even function** to check the (*n*+1)-bit code
- **Even parity:** count of 1's in the (*n*+1)-bit code is **even**
  - Use an **odd function** to generate the **even parity bit**
  - Use an **odd function** to check the (*n*+1)-bit code

Example of Parity Generator and Checker

- Design **even** parity generator & checker for **3-bit** codes

- Solution:
  - Use **3-bit odd function** to generate even parity bit P.
  - Use **4-bit odd function** to check if there is an error E in even parity.
  - Given that: xyz = 001 then P = 1. The sender transmits $Pxyz$ = 1001.
  - If y changes from 0 to 1 between generator and checker, the parity checker receives $Pxyz$ = 1011 and produces E = 1, indicating an error.

Parity Generator



Parity Checker