Gate-Level Minimization

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

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 · over +
$f = x'z + x'yz' + xz$	Simplify (7 literals)
$f = x'z + xz + x'yz'$	Reorder
$f = (x' + x)z + x'yz'$	Distributive · over +
$f = z + x'yz'$	Simplify (4 literals)
$f = (z + x'y)(z + z')$	Distributive + over ·
$f = z + x'y$	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)

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 \ y	0	1
0	m_0	m_1
1	m_2	m_3

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

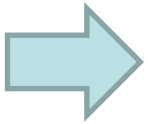
Note: adjacent squares horizontally and vertically ***NOT*** *diagonally*

From a Truth Table to Karnaugh Map

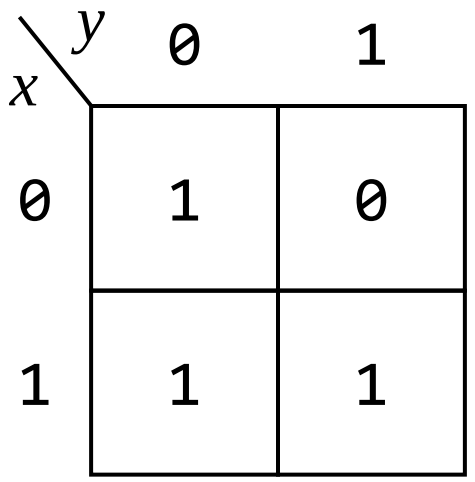
- Given a truth table, construct the corresponding K-map
- Copy the function values from the truth table into the K-map
- Make sure to copy each value into the proper K-map square

Truth Table

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



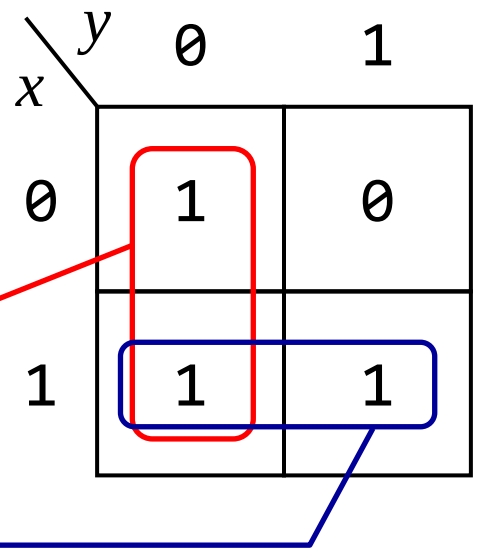
K-map



Two-variable Map Minimization- Example 1

- 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

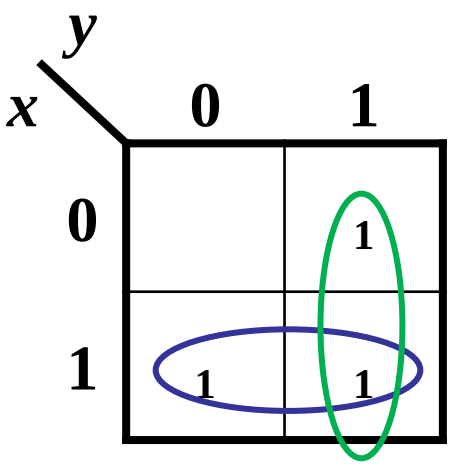


Two-variable Map - Example 2

❖ Example

	x y	F	Minterm	
0	0 0	0	m_0	$x'y'$
1	0 1	1	m_1	$x'y$
2	1 0	1	m_2	$x\,y'$
3	1 1	1	m_3	xy

m_0	m_1
m_2	m_3



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

$$(x' + x)y \qquad x(y + y')$$

$$F = y + x$$

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

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

	x \ 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'

Three-variable Map Minimization – Example 1

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

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

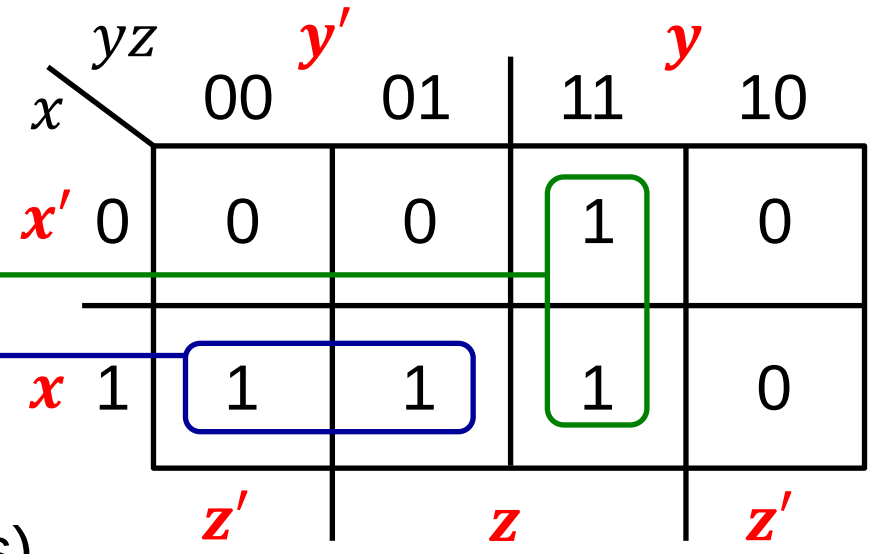
1. Mark '**1**' all the K-map squares that represent function f

2. Find possible adjacent squares

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

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

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



Three-variable Map Minimization – Example 2

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

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

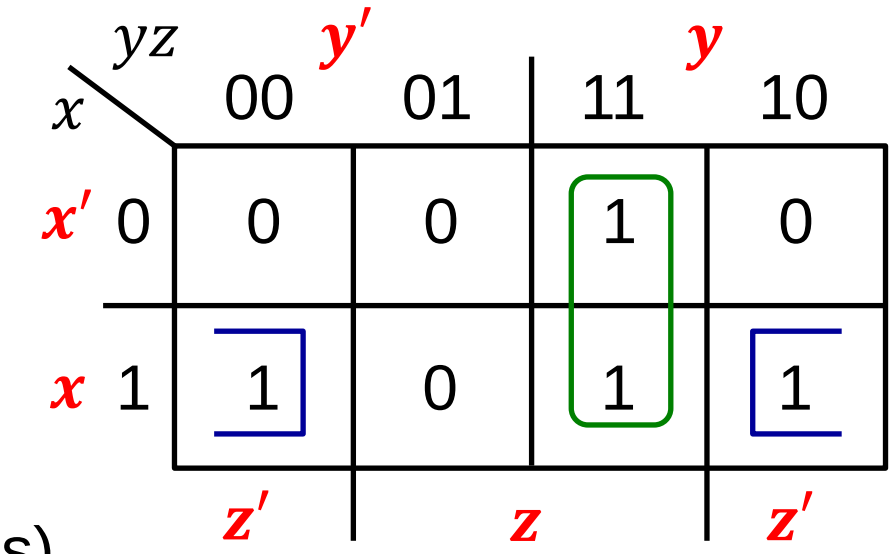
Learn the locations of the 8 indices based on the variable order

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

Corner squares can be combined

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

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



Combining Squares on a 3-Variable K-Map

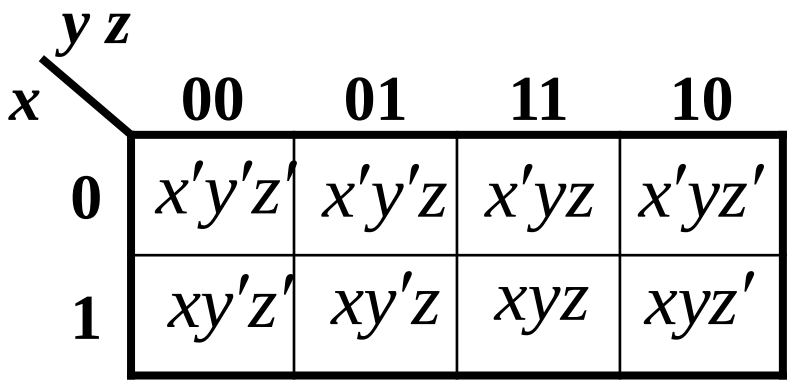
- By combining squares, we reduce number of literals in a product term, thereby reducing the cost
- On a 3-variable K-Map:
  - One square represents a minterm with 3 variables
  - Two adjacent squares represent a term with 2 variables
  - Four adjacent squares represent a term with 1 variable
  - Eight adjacent square is the constant '**1**' (no variables)

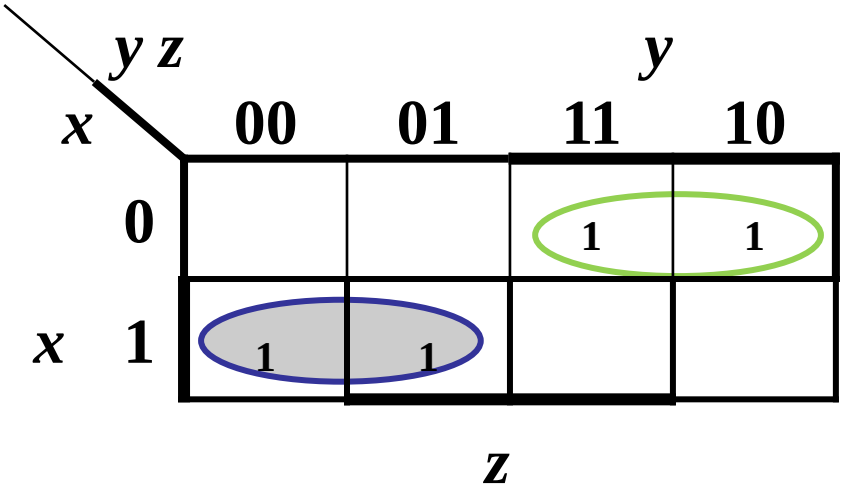
Three-variable Map Minimization – Example 3

❖ **Example:** Simplify the Boolean function

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

	x y z	F	Minterm	
0	0 0 0	0	m_0	$x'y'z'$
1	0 0 1	0	m_1	$x'y'z$
2	0 1 0	1	m_2	$x'yz'$
3	0 1 1	1	m_3	$x'yz$
4	1 0 0	1	m_4	$xy'z'$
5	1 0 1	1	m_5	$xy'z$
6	1 1 0	0	m_6	xyz'
7	1 1 1	0	m_7	xyz

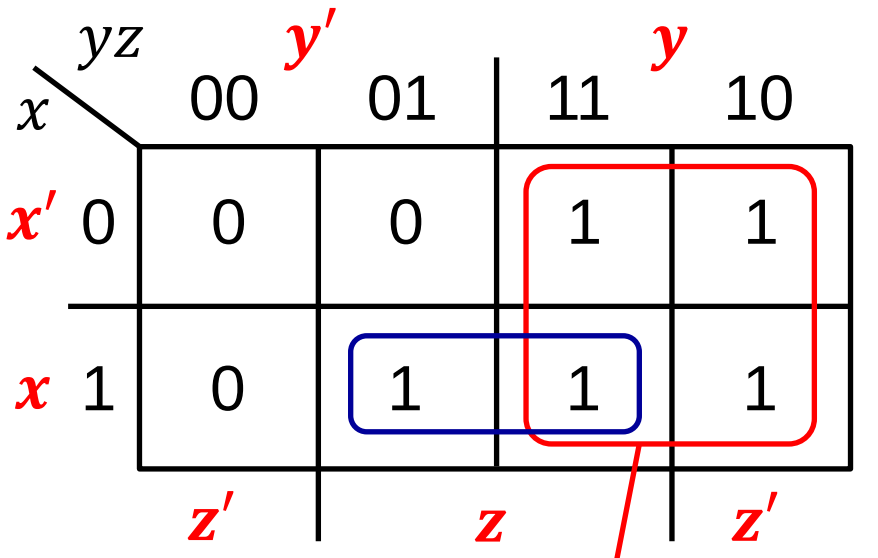




$F = xy' + x'y$

Three-variable Map Minimization – Example 4

- 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: $f = y + xz$



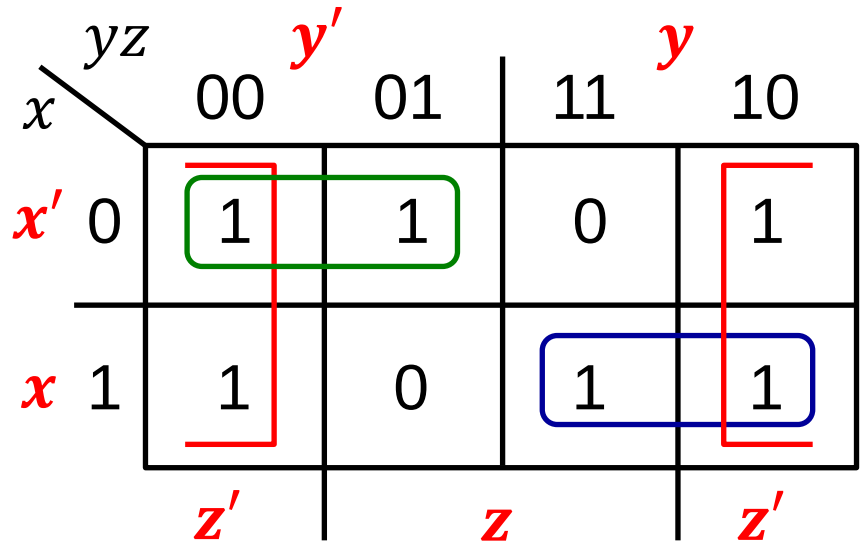
$$x'yz + x'yz' + xyz + xyz'$$
$$= x'y(z + z') + xy(z + z')$$
$$= x'y + xy = (x' + x)y = y$$

Three-variable Map Minimization – Example 5

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)

Three-variable Map Minimization – Example 6

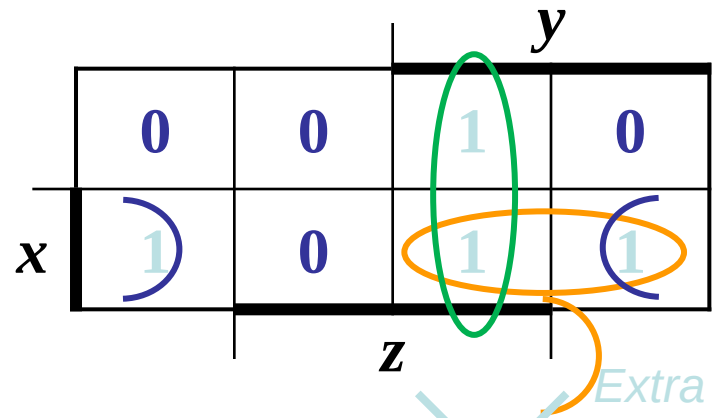
❖ **Example**

$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

	x y z	F	Minterm	
0	0 0 0	0	m_0	$x'y'z'$
1	0 0 1	0	m_1	$x'y'z$
2	0 1 0	0	m_2	$x'yz'$
3	0 1 1	1	m_3	$x'yz$
4	1 0 0	1	m_4	$xy'z'$
5	1 0 1	0	m_5	$xy'z$
6	1 1 0	1	m_6	xyz'
7	1 1 1	1	m_7	xyz

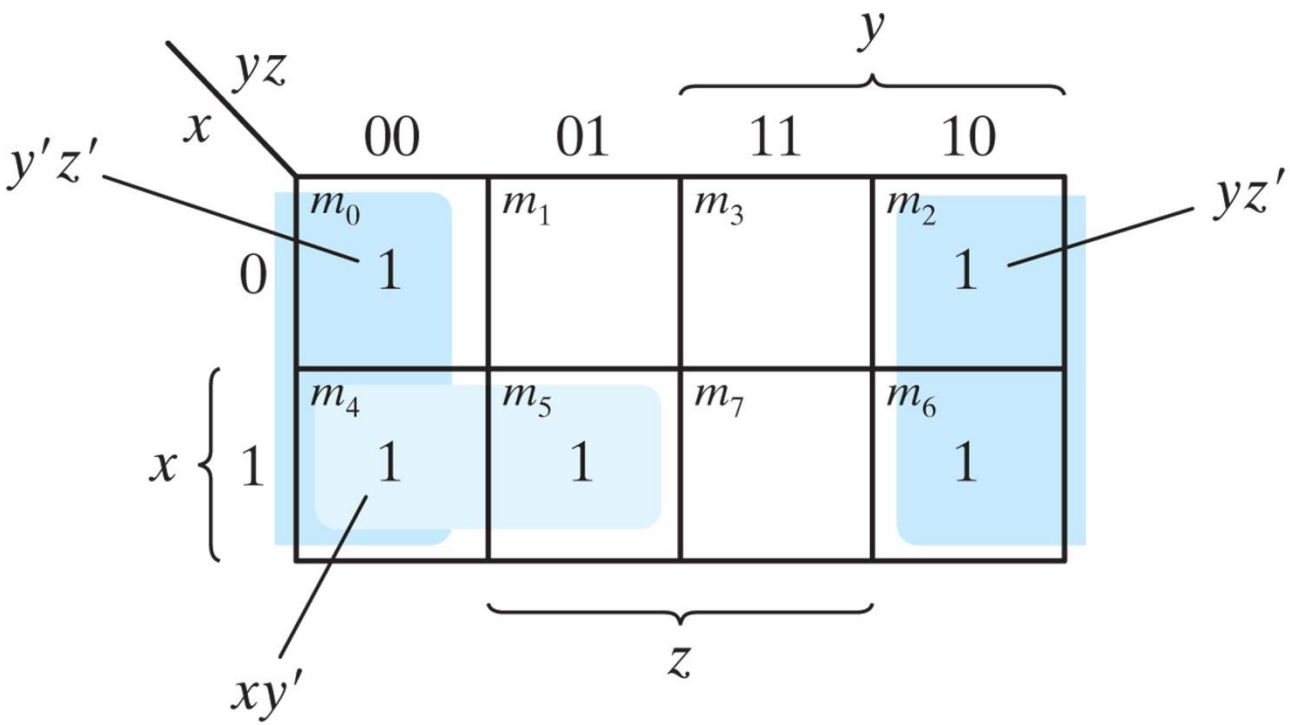
x \ $y\,z$	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'



$F = xz' + yz + xy$

Three-variable Map Minimization – Example 7

Map for $F(x,y,z) = \Sigma(0,2,4,5,6) = z' + xy'$

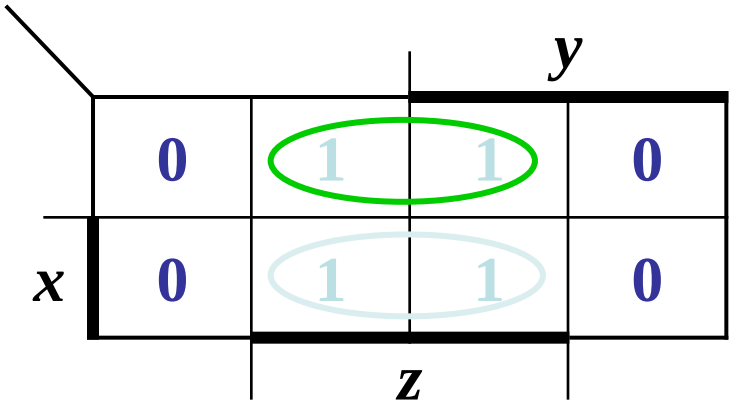


Note: $y'z' + yz' = z'$

Three-variable Map Minimization – Example 8

❖ **Example:**

	x y z	F	Minterm	
0	0 0 0	0	m_0	$x'y'z'$
1	0 0 1	1	m_1	$x'y'z$
2	0 1 0	0	m_2	$x'yz'$
3	0 1 1	1	m_3	$x'yz$
4	1 0 0	0	m_4	$xy'z'$
5	1 0 1	1	m_5	$xy'z$
6	1 1 0	0	m_6	xyz'
7	1 1 1	1	m_7	xyz

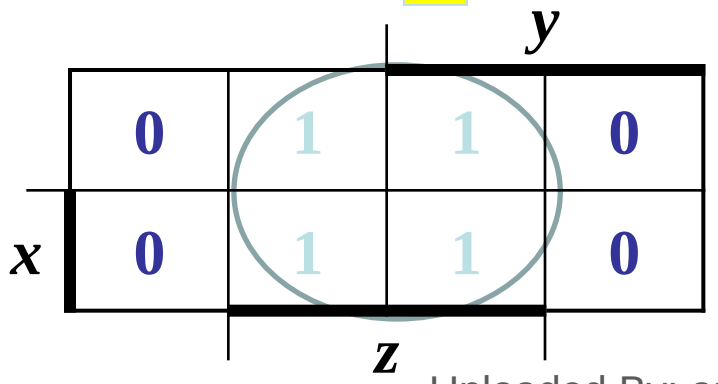


$$F(x,y,z) = x'y'z + x'yz + xy'z + xyz$$

$$\bar{x}\,z(\bar{y}+y) \qquad x\,z(\bar{y}+y)$$

$$\bar{x}\,z \qquad x\,z$$

$$z$$



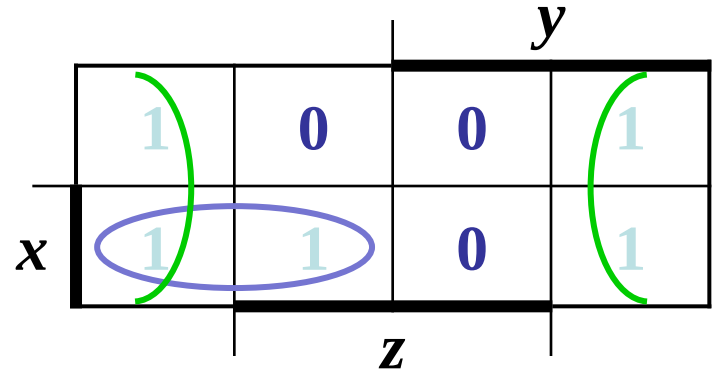
Three-variable Map Minimization – Example 9

$$F(x,y,z) = \sum(0,2,4,5,6)$$

	x y z	F	Minterm	
0	0 0 0	1	m_0	$x'y'z'$
1	0 0 1	0	m_1	$x'y'z$
2	0 1 0	1	m_2	$x'yz'$
3	0 1 1	0	m_3	$x'yz$
4	1 0 0	1	m_4	$xy'z'$
5	1 0 1	1	m_5	$xy'z$
6	1 1 0	1	m_6	xyz'
7	1 1 1	0	m_7	xyz

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

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



$$F = z' + xy'$$

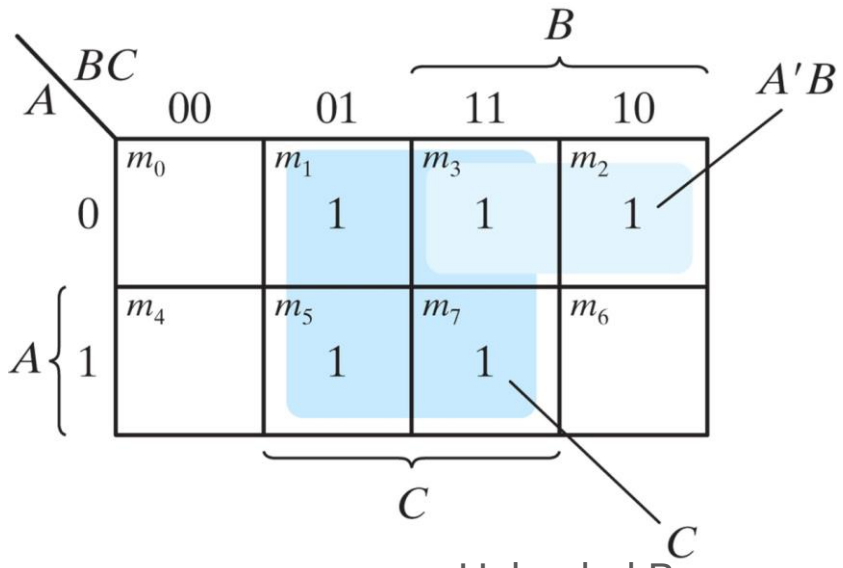
Three-variable Map Minimization – Example 10

Let the Boolean Function

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

(a) Express this function as a sum of minterms

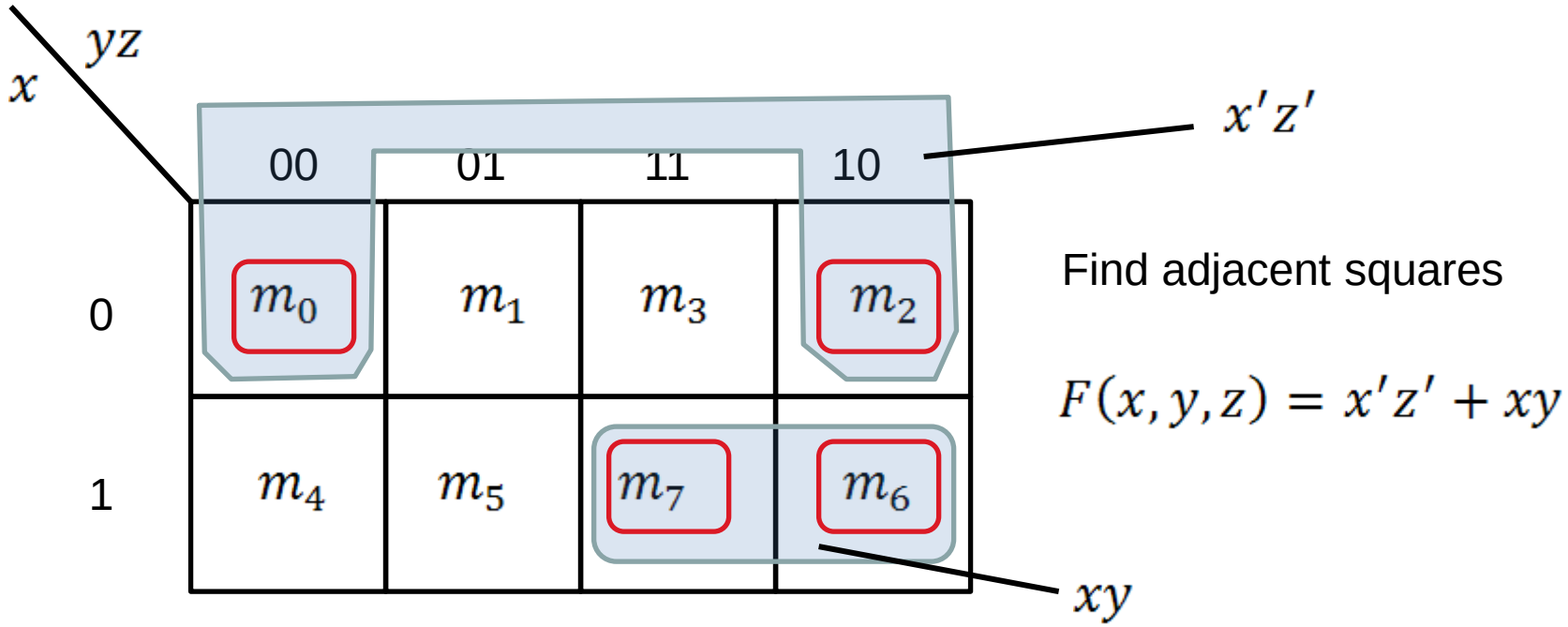
$$= A'(B + B')C + A'B(C + C') + AB'C$$
$$+(A + A')BC$$
$$= A'BC + A'B'C + \cancel{A'BC} + A'BC' + AB'C$$
$$+ABC + \cancel{A'BC}$$
$$= A'BC + A'B'C + A'BC' + AB'C + ABC$$
$$= \sum(1,2,3,5,7)$$



Three-variable Map Minimization – Example 11

Simplify the following Boolean function, using three-variable maps

$$F(x,y,z) = xy + x'y'z' + x'yz'$$
$$= xy(z + z') + x'y'z' + x'yz'$$
$$= xyz + xyz' + x'y'z' + x'yz'$$
$$= \sum(0,2,6,7)$$



Find adjacent squares

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

Four-Variable Karnaugh Map

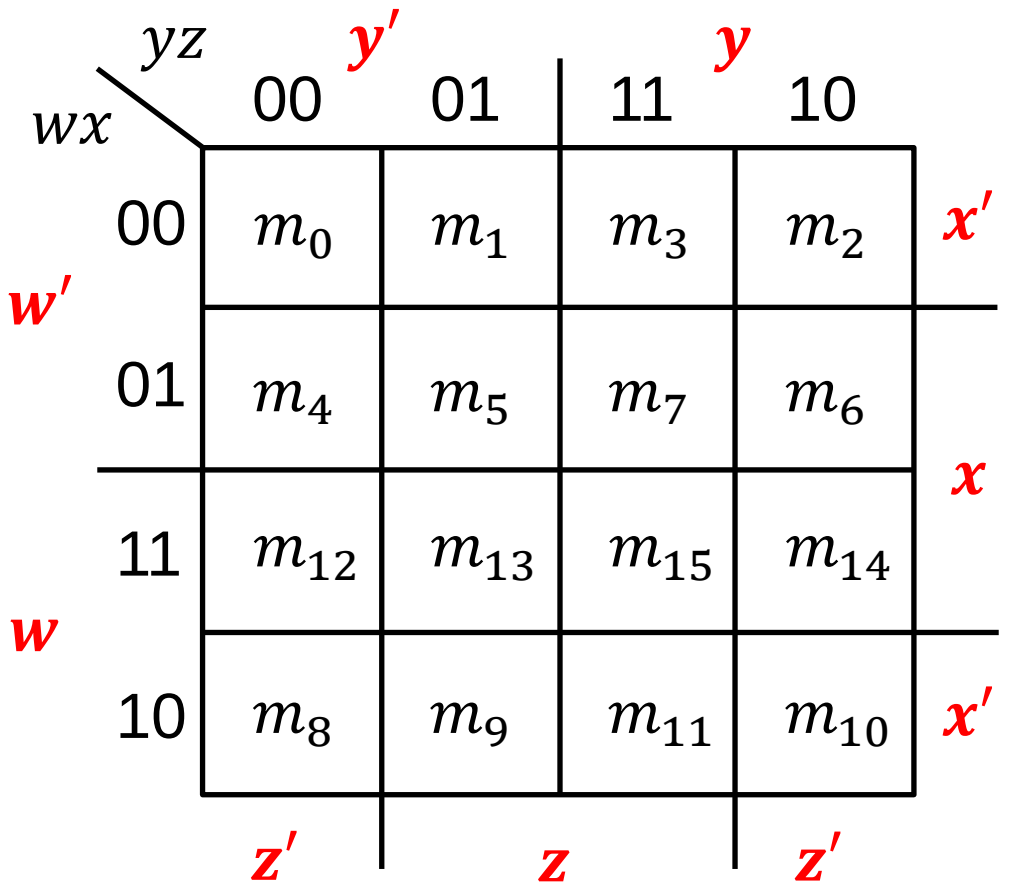
4 variables ➔ 16 squares

Remember the numbering of the squares in the K-map

Each square is adjacent to four other squares

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

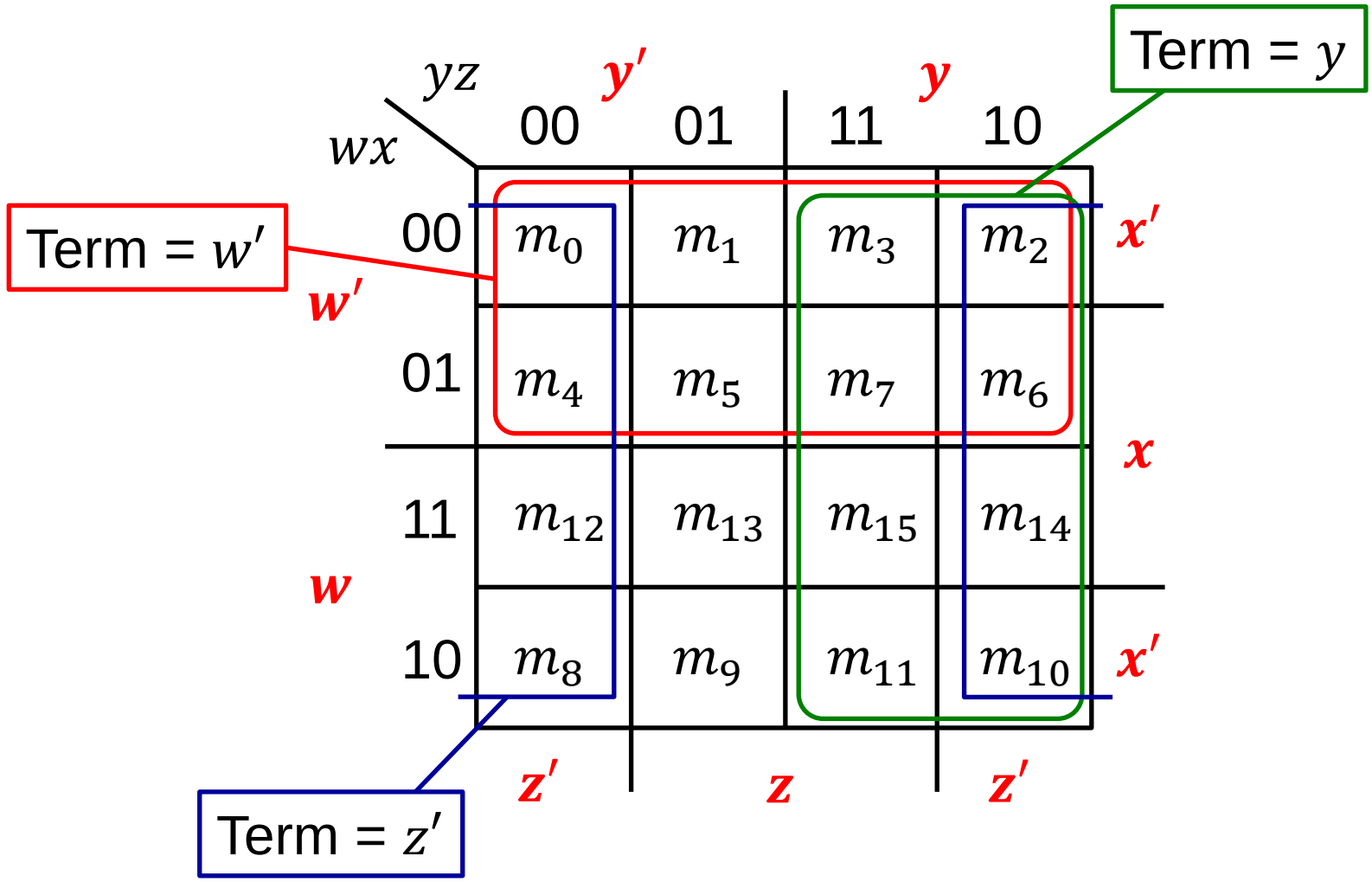
Notice the order of Rows 11 and 10 and the order of columns 11 and 10



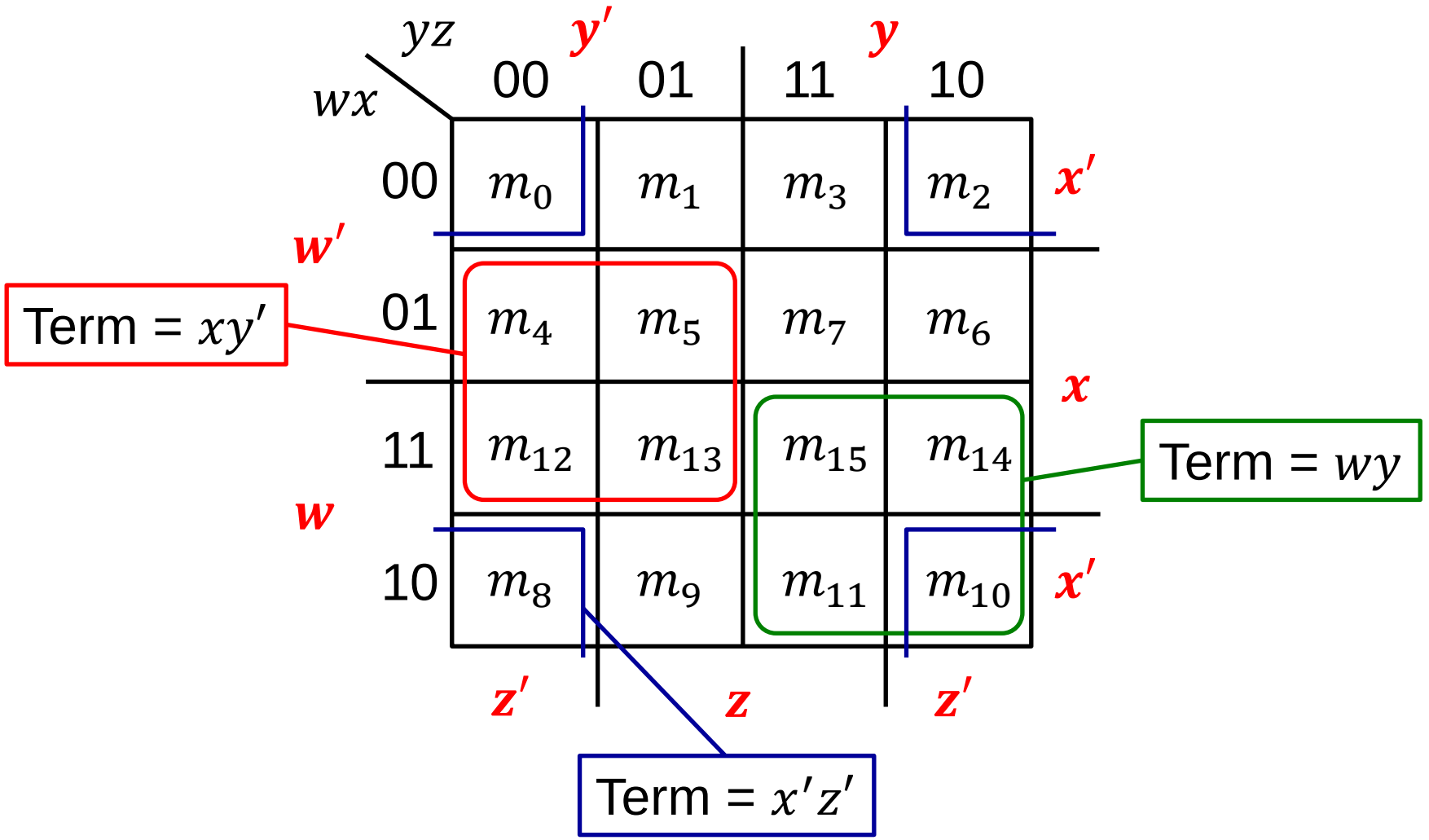
Combining Squares on a 4-Variable K-Map

- 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)

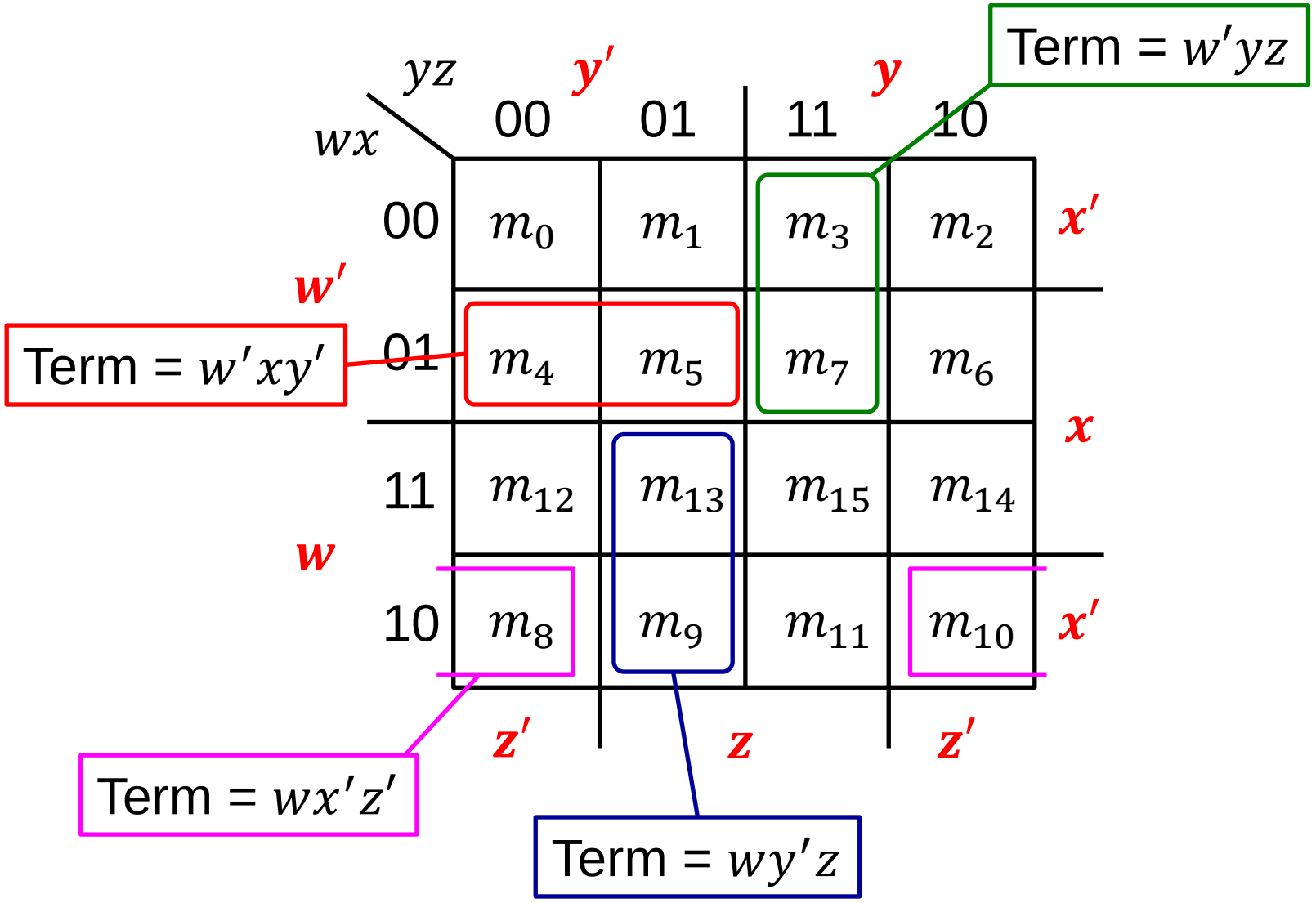
Combining Eight Squares



Combining Four Squares



Combining Two Squares



Four-variable Map Minimization – Example 1

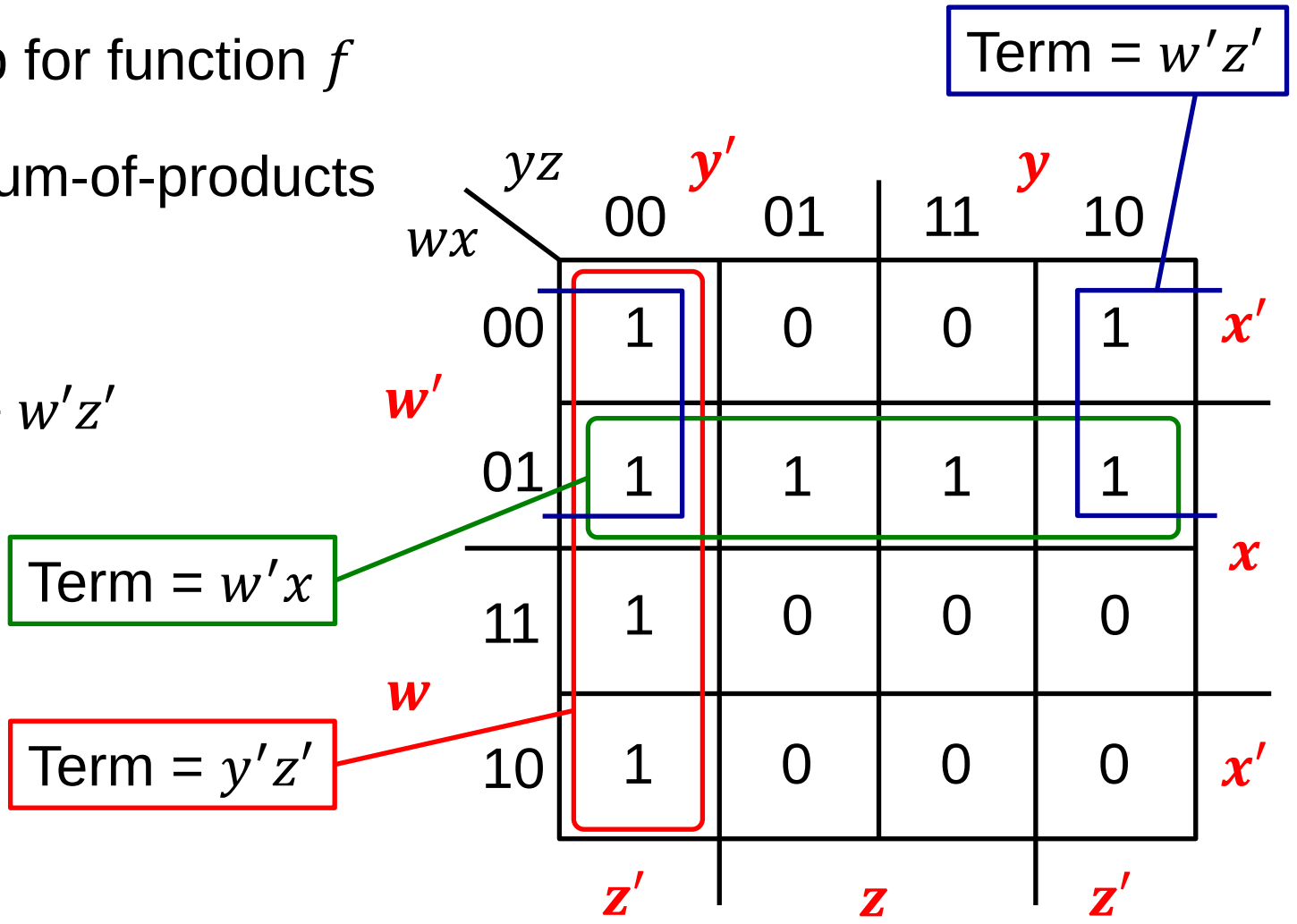
Given $f(w,x,y,z) = \sum(0,2,4,5,6,7,8,12)$

Draw the K-map for function f

Minimize f as sum-of-products

Solution:

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

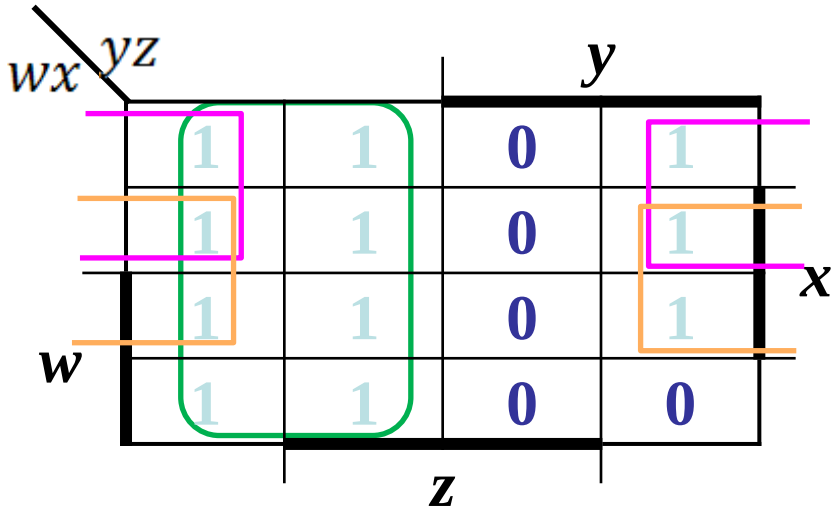


Four-variable Map Minimization – Example 2

$$F(w,x,y,z) = \sum(0,1,2,4,5,6,8,9,12,13,14)$$

	w x y z	F	Minterm	
0	0 0 0 0	1	m_0	$w'x'y'z'$
1	0 0 0 1	1	m_1	$w'x'y'z$
2	0 0 1 0	1	m_2	$w'x'yz'$
3	0 0 1 1	0	m_3	$w'x'yz$
4	0 1 0 0	1	m_4	$w'xy'z'$
5	0 1 0 1	1	m_5	$w'xy'z$
6	0 1 1 0	1	m_6	$w'xyz'$
7	0 1 1 1	0	m_7	$w'xyz$
8	1 0 0 0	1	m_8	$wx'y'z'$
9	1 0 0 1	1	m_9	$wx'y'z$
10	1 0 1 0	0	m_{10}	$wx'yz'$
11	1 0 1 1	0	m_{11}	$wx'yz$
12	1 1 0 0	1	m_{12}	$wxy'z'$
13	1 1 0 1	1	m_{13}	$wxy'z$
14	1 1 1 0	1	m_{14}	$wxyz'$
15	1 1 1 1	0	m_{15}	$wxyz$

wx \ yz	00	01	11	10
00	$\bar{w}\bar{x}\bar{y}\bar{z}$	$\bar{w}\bar{x}\bar{y}z$	$\bar{w}\bar{x}yz$	$\bar{w}\bar{x}y\bar{z}$
01	$\bar{w}x\bar{y}\bar{z}$	$\bar{w}x\bar{y}z$	$\bar{w}xyz$	$\bar{w}xy\bar{z}$
11	$wx\bar{y}\bar{z}$	$wx\bar{y}z$	$wxyz$	$wxy\bar{z}$
10	$w\bar{x}\bar{y}\bar{z}$	$w\bar{x}\bar{y}z$	$w\bar{x}yz$	$w\bar{x}y\bar{z}$

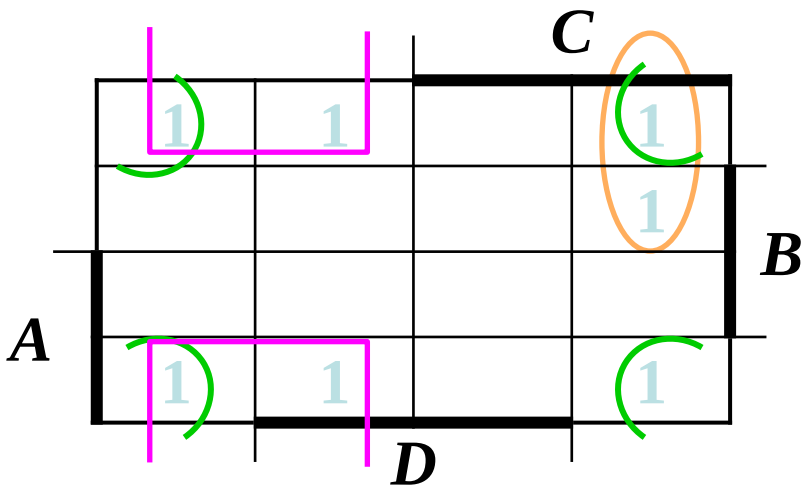


$$F = y' + w'z' + xz'$$

Four-variable Map Minimization – Example 3

❖Example

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



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

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

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

Prime implicants

When choosing adjacent squares in a map, make sure that:

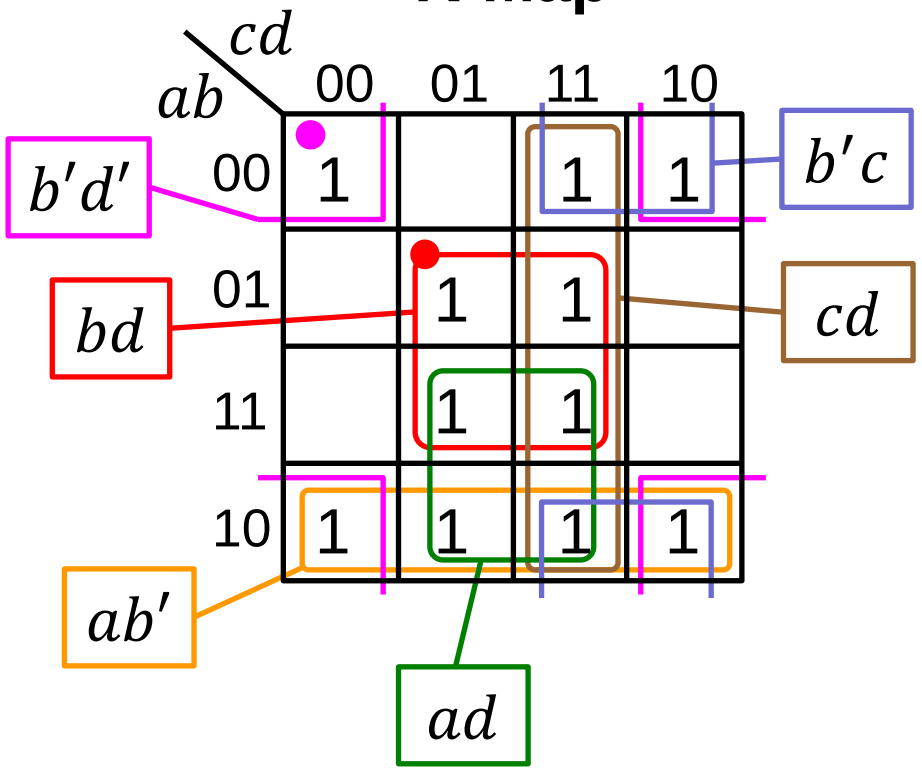
1. All the minterms are covered when combining the squares
2. The number of terms in the expression is minimized
3. There are no redundant terms (minterms already covered by other terms)

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 $b'd'$

Prime Implicant - Example 2

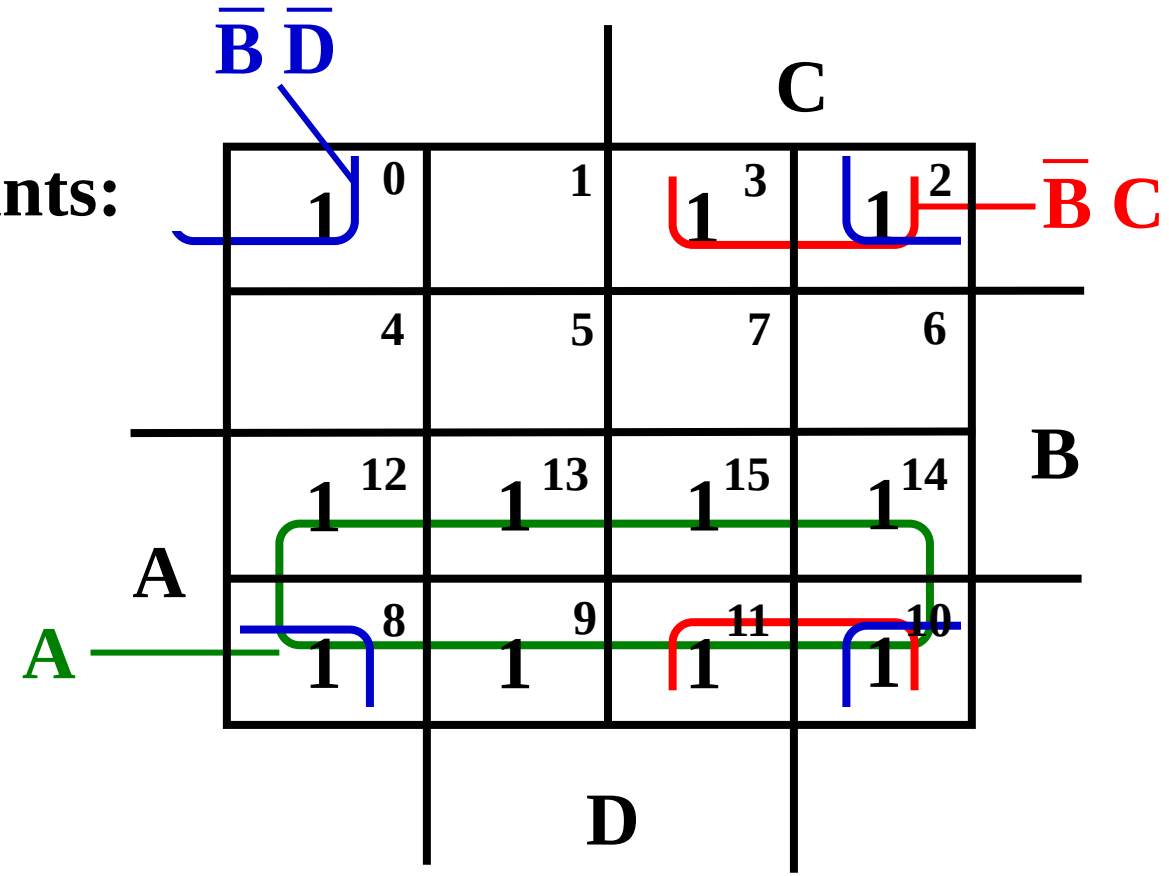
❖ **Find all prime implicants for**

$$F(A, B, C, D) = \sum (0,2,3,8,9,10,11,12,13,14,15)$$

3 prime implicants:

A, $\overline{B}$ C, $\overline{B}$ $\overline{D}$

All 3 prime implicants are essential



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

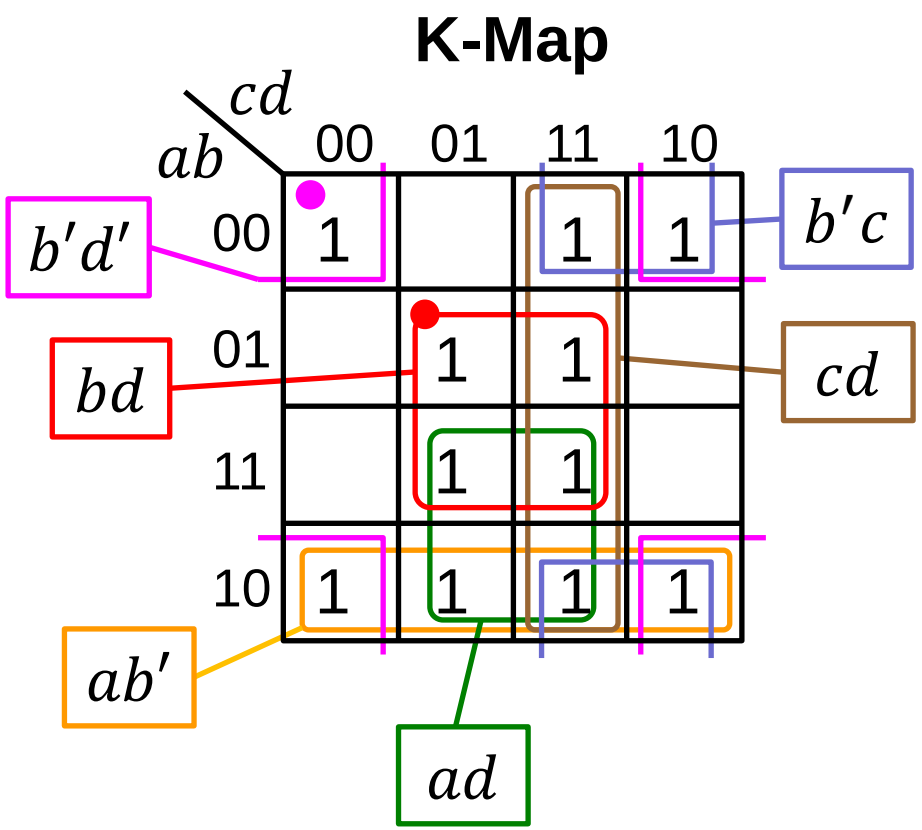
❖ Sometimes, a function has multiple simplified expressions

- ✧ 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



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

Four possible solutions:

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

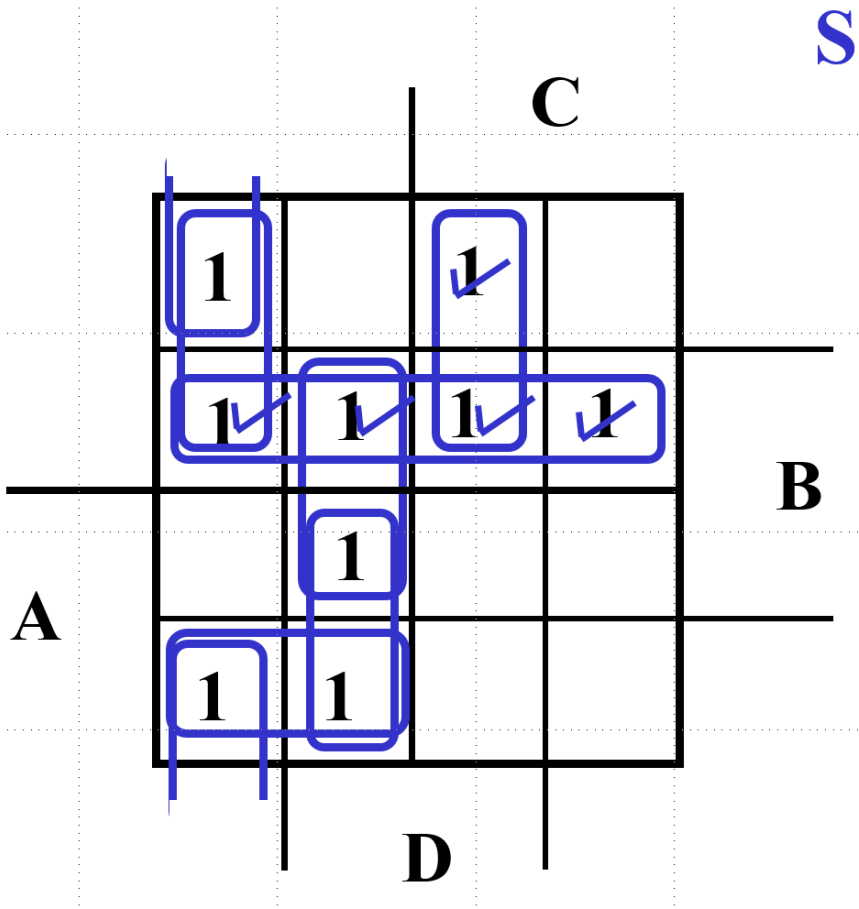
$f = bd + b'd' + cd + ab'$

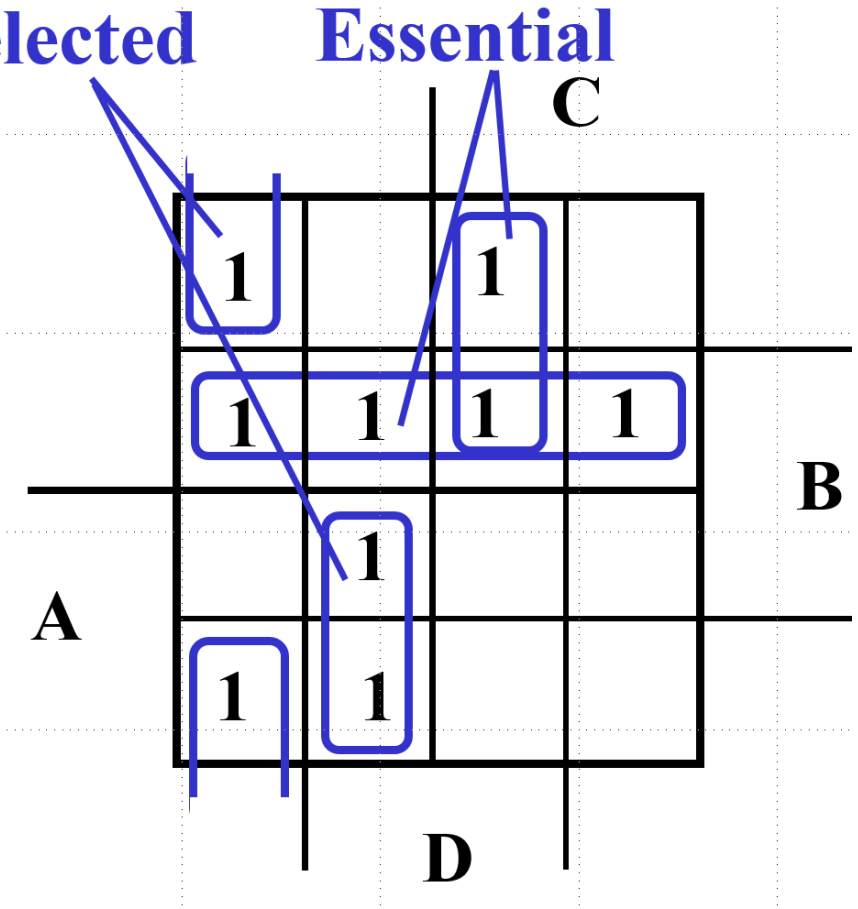
$f = bd + b'd' + b'c + ab'$

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

Simplification Example 2

- **Simplify F(A, B, C, D) given on the K-map**

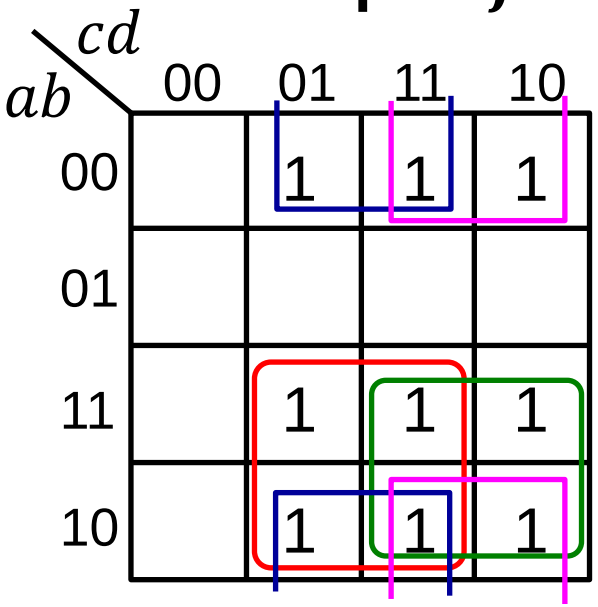




Product-of-Sums (POS) Simplification

- 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

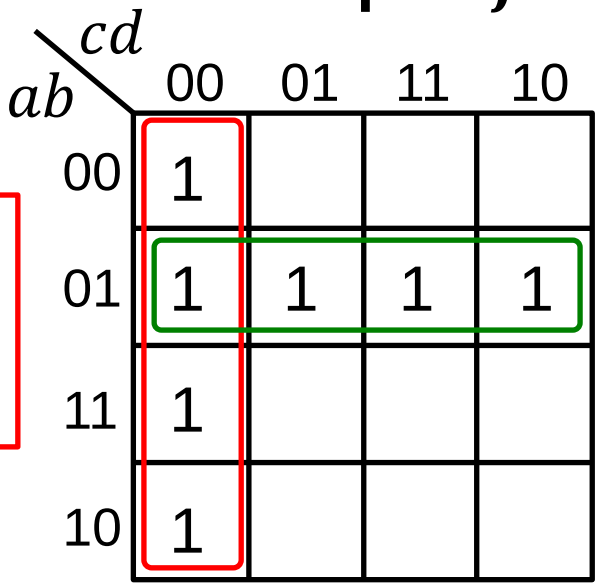


$f = ad + ac + b'd + b'c$

Minimal Sum-of-Products = 8 literals

All prime implicants are essential

K-Map of f'



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

$f = (c + d)(a + b')$ →

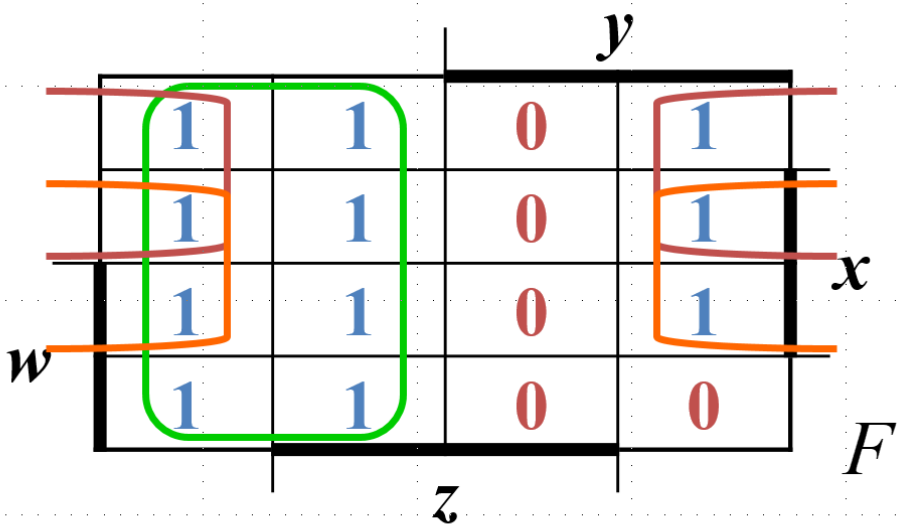
Minimal Product-of-Sums = 4 literals

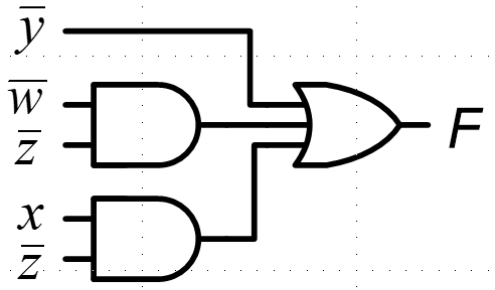
Product-of-Sums Simplification Procedure

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

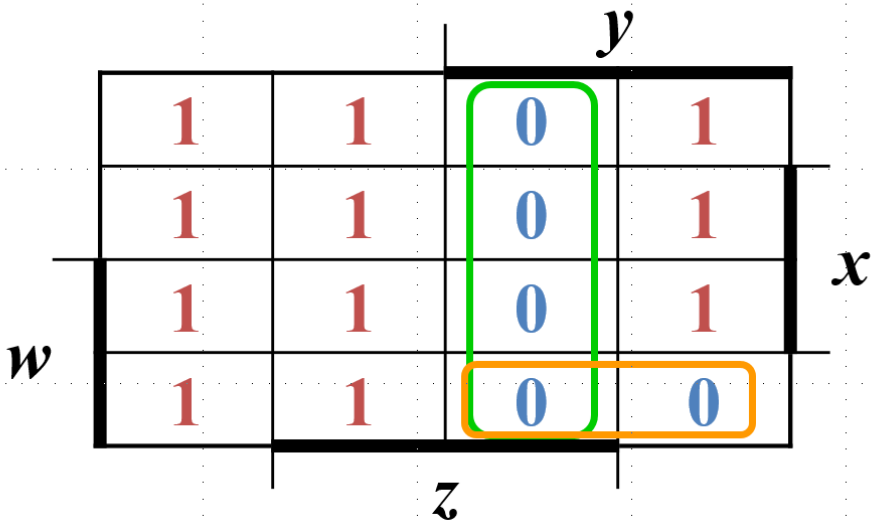
Product of Sums Simplification-example 2

	w x y z	F	$\overline{F}$
0	0 0 0 0	1	0
1	0 0 0 1	1	0
2	0 0 1 0	1	0
3	0 0 1 1	0	1
4	0 1 0 0	1	0
5	0 1 0 1	1	0
6	0 1 1 0	1	0
7	0 1 1 1	0	1
8	1 0 0 0	1	0
9	1 0 0 1	1	0
10	1 0 1 0	0	1
11	1 0 1 1	0	1
12	1 1 0 0	1	0
13	1 1 0 1	1	0
14	1 1 1 0	1	0
15	1 1 1 1	0	1



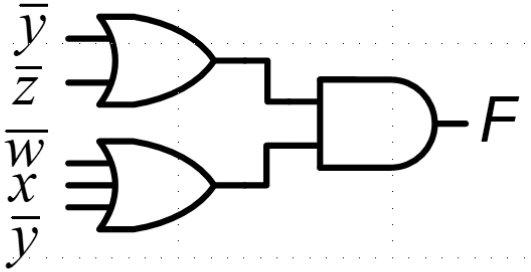


$$F = \overline{y} + \overline{w}\,\overline{z} + x\,\overline{z}$$



$$\overline{F} = y\,z + w\,\overline{x}\,y$$

$$\overline{\overline{F}} = \overline{y\,z + w\,\overline{x}\,y}$$



$$F = (\overline{y} + \overline{z}) \bullet (\overline{w} + x + \overline{y})$$

Product-Of-Sums Simplification – Example 3

$$F(A,B,C,D) = \sum(0,1,2,5,8,9,10)$$

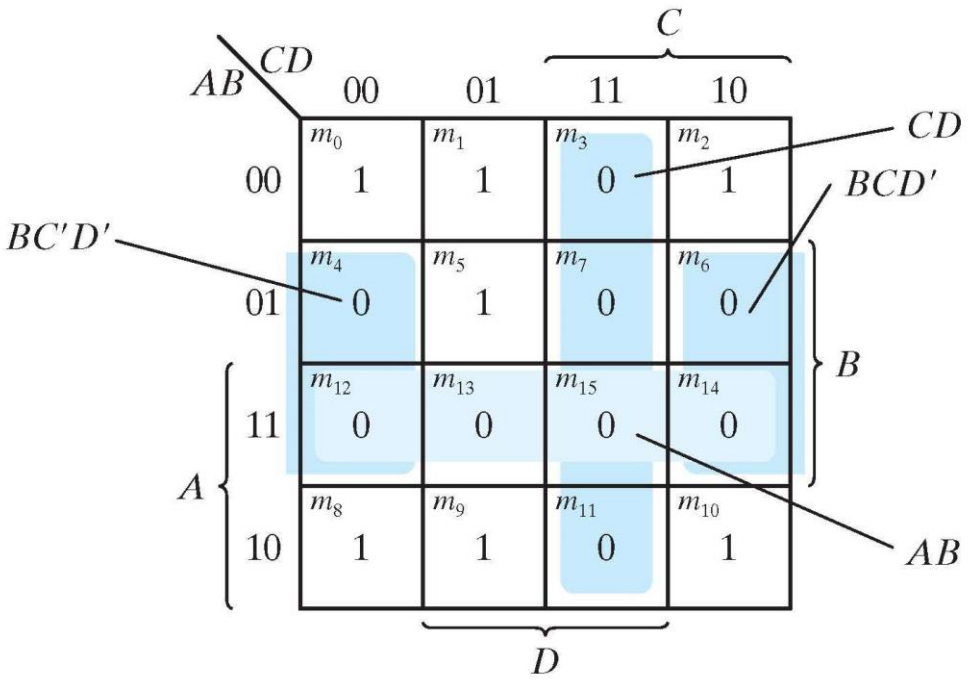
$$F = B'D' + B'C' + A'C'D$$

Take the squares with zeros and obtain the simplified complemented function

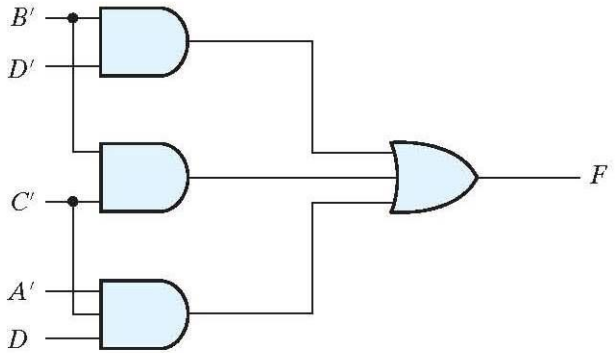
$$F' = AB + CD + BD'$$

Complement the above expression and use DeMorgan's

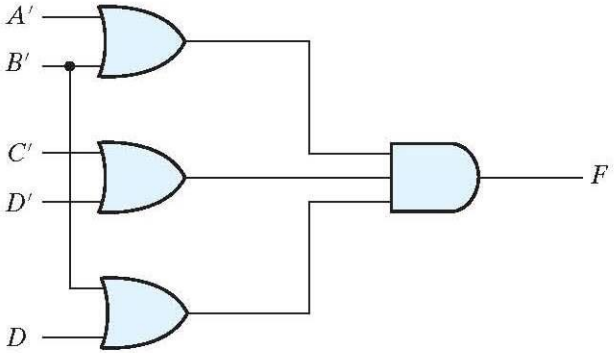
$$F = (A' + B')(C' + D')(B' + D)$$



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



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



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

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

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

Don't-Care Condition

❖Example

$$A = \begin{cases} 1 & \text{if a 5 NIS is deposited} \\ 0 & \textit{otherwise} \end{cases}$$

$$B = \begin{cases} 1 & \text{if a 2 NIS is deposited} \\ 0 & \textit{otherwise} \end{cases}$$

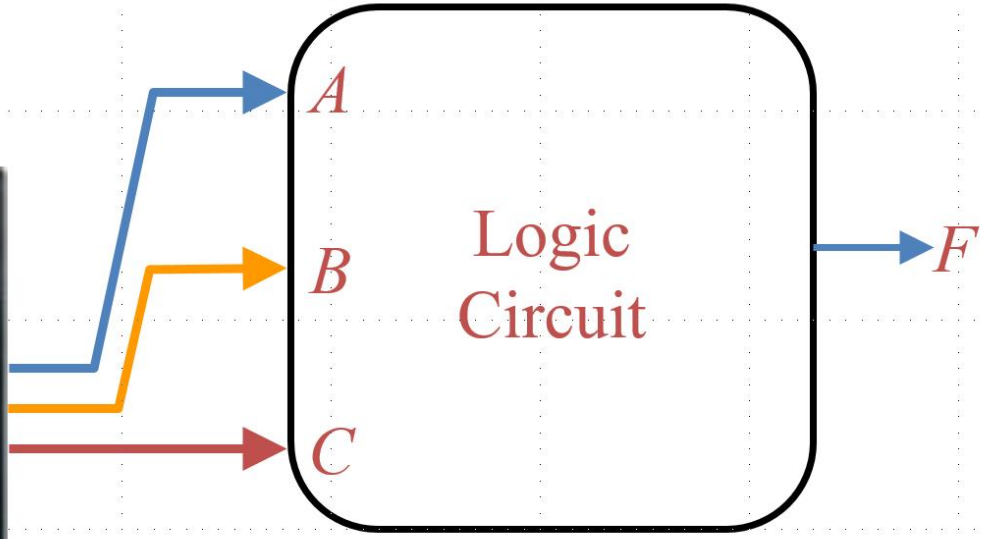
$$C = \begin{cases} 1 & \text{if a 1 NIS is deposited} \\ 0 & \textit{otherwise} \end{cases}$$

A	B	C	NIS Value
0	0	0	0.00
0	0	1	1 NIS
0	1	0	2 NIS
0	1	1	Not possible
1	0	0	5 NIS
1	0	1	Not possible
1	1	0	Not possible
1	1	1	Not possible

You can only drop one coin at a time.

Used as "don't care"

Don't-Care Condition



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

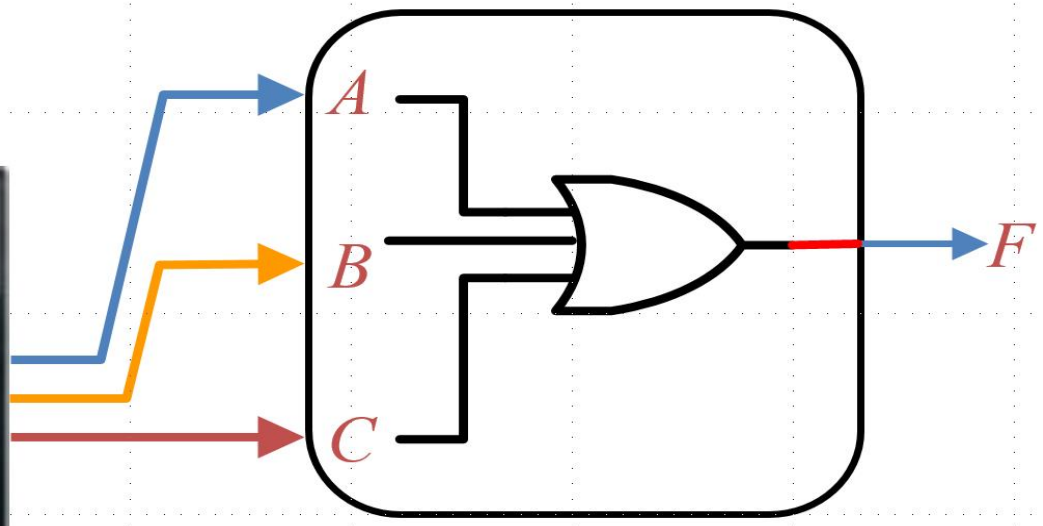
$$F(A,B,C) = \sum(1, 2, 4)$$

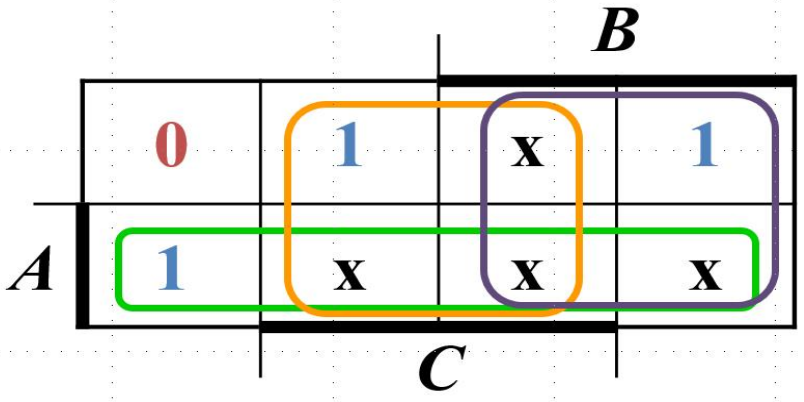
$$d(A,B,C) = \sum(3, 5, 6, 7)$$

Don't care what value F may take

Don't-Care Condition – Example 1

- Example





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

$$F = A + B + C$$

Minimizing Functions with Don't Cares – Example 2

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)

K-Map of f

ab \ 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

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

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 – Example 3

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

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

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

Prime Implicant cd is essential

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

Not all don't cares need be covered

Minimal Product-of-Sums with Don't Cares

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

Obtain a product-of-sums minimal 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 \ 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

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 \ 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 \ 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} …

Five-Variable K-Map – Example 1

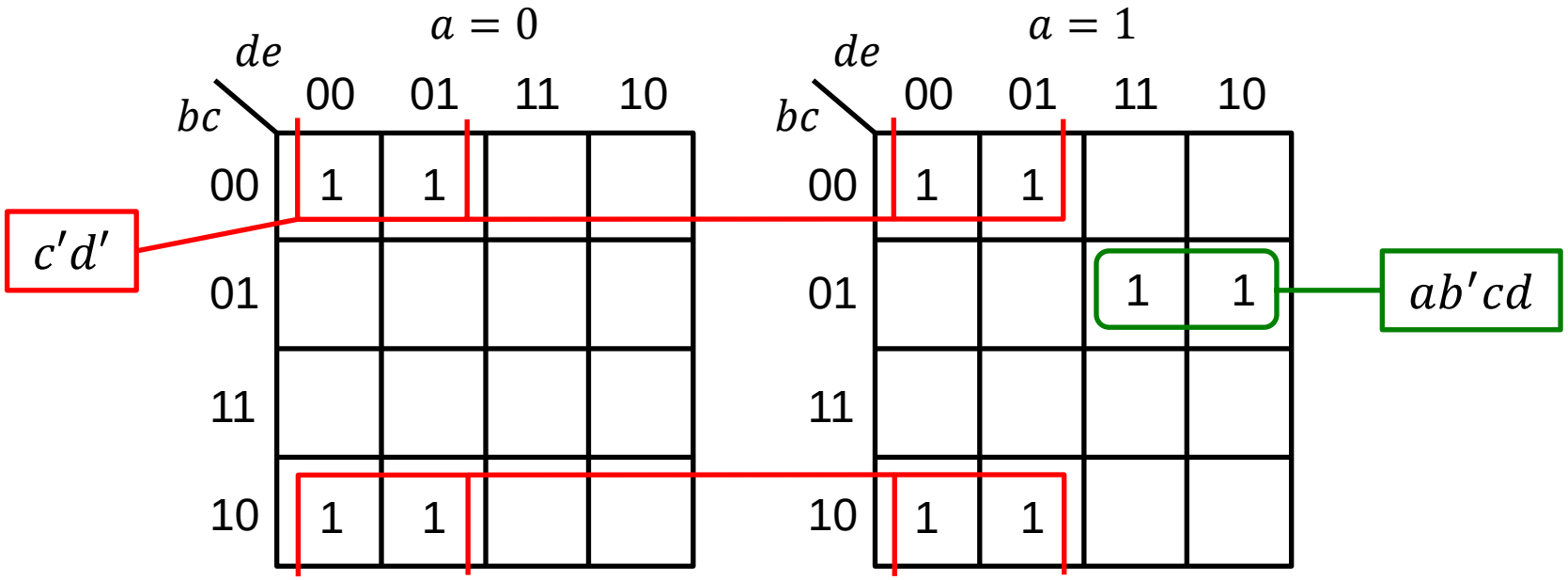
Given: $f(a,b,c,d,e) = \sum(0,1,8,9,16,17,22,23,24,25)$

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 – Example 3

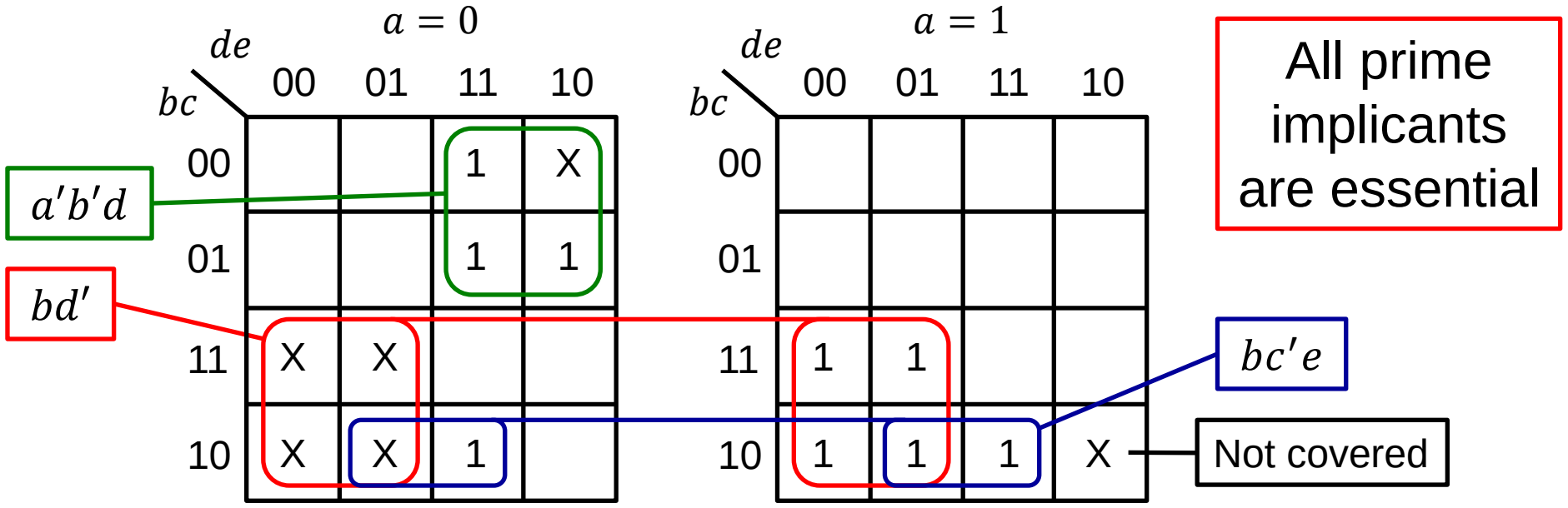
$g(a,b,c,d,e) = \sum_m(3,6,7,11,24,25,27,28,29) + \sum_d(2,8,9,12,13,26)$

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

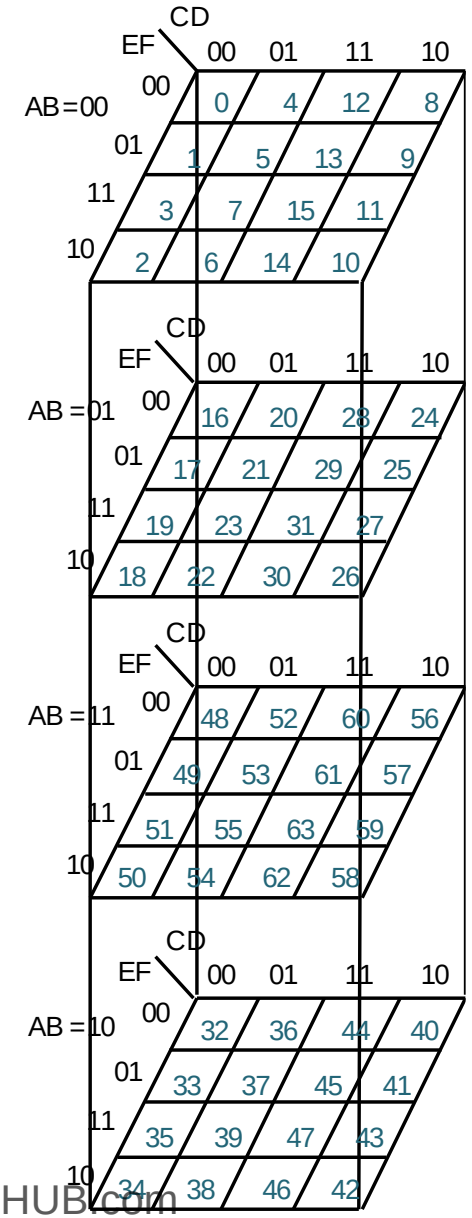


Six-Variable Karnaugh Map

- 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

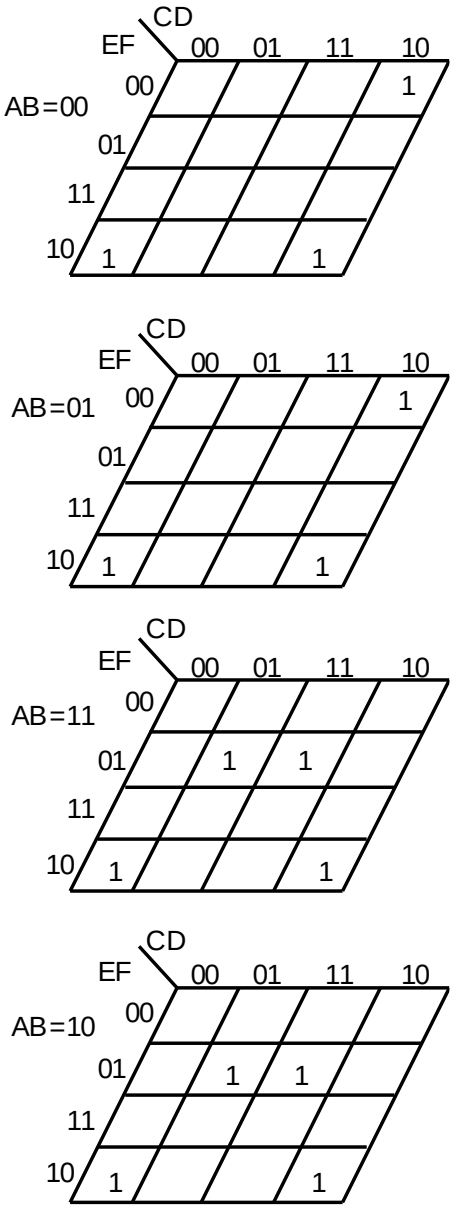
	$ab = 00$				$ab = 01$				$ab = 11$				$ab = 10$			
cd \ ef	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	m_0	m_1	m_3	m_2	m_{16}	m_{17}	m_{19}	m_{18}	m_{48}	m_{49}	m_{51}	m_{50}	m_{32}	m_{33}	m_{35}	m_{34}
01	m_4	m_5	m_7	m_6	m_{20}	m_{21}	m_{23}	m_{22}	m_{52}	m_{53}	m_{55}	m_{54}	m_{36}	m_{37}	m_{39}	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}

6- Variable K-Maps - Example 1

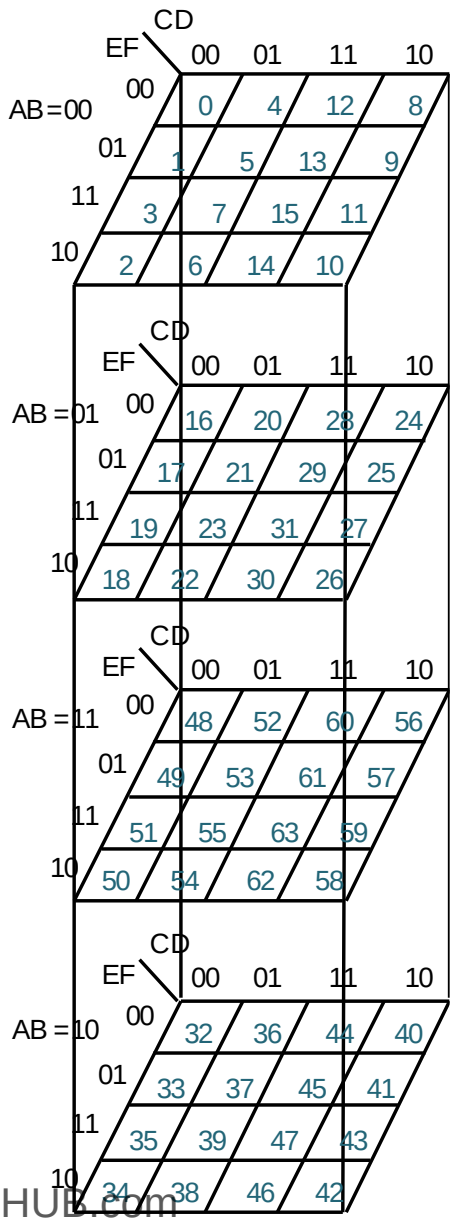


$f(A,B,C,D,E,F) = \Sigma m(2,8,10,18,24,26,34,37,42,45,50,53,58,61)$

=

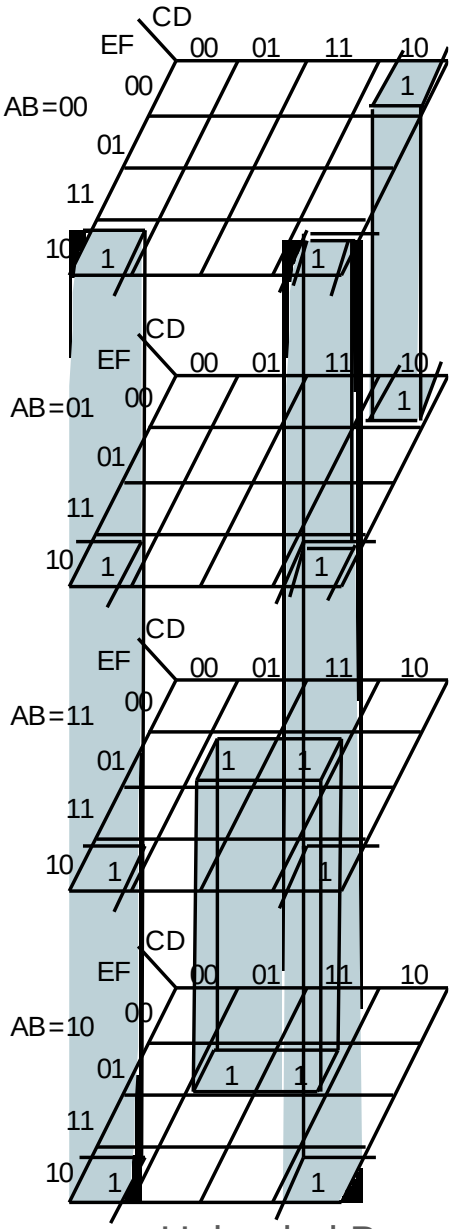


6- Variable K-Maps - Example 1



$f(A,B,C,D,E,F) = \Sigma m(2,8,10,18,24,26,34,37,42,45,50,53,58,61)$

$= D'\,E\,F' + A\,D\,E'\,F + A'\,C\,D'\,F'$



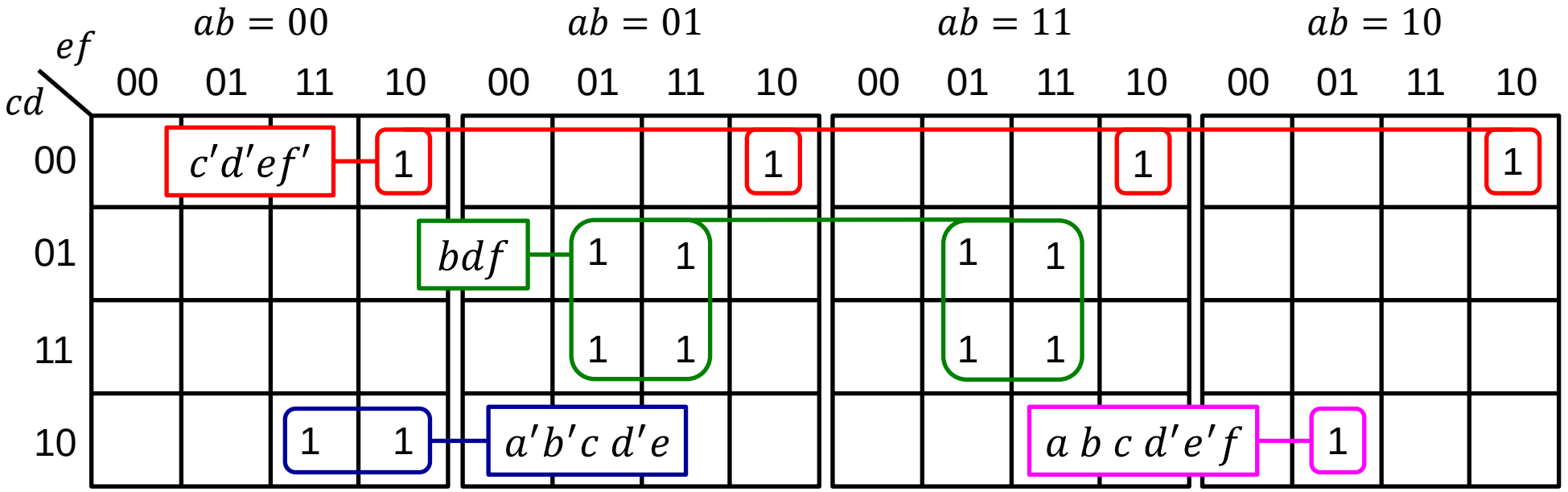
6 Variable K-Maps - Example 2

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

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)

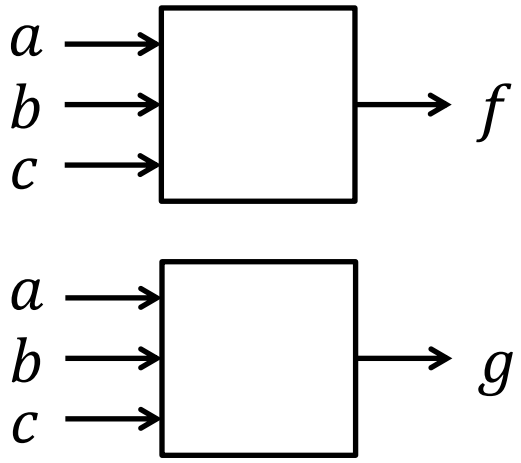


Next . . .

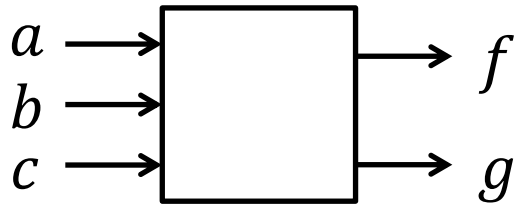
- Boolean Function Minimization
- The Karnaugh Map (K-Map)
- Two, Three, and Four-Variable K-Maps
- Prime and Essential Prime Implicants
- Minimal Sum-of-Products and Product-of-Sums
- Don't Cares
- Five and Six-Variable K-Maps
- Multiple Outputs

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 \ bc	00	01	11	10
0	1	0	0	1
1	0	0	1	1

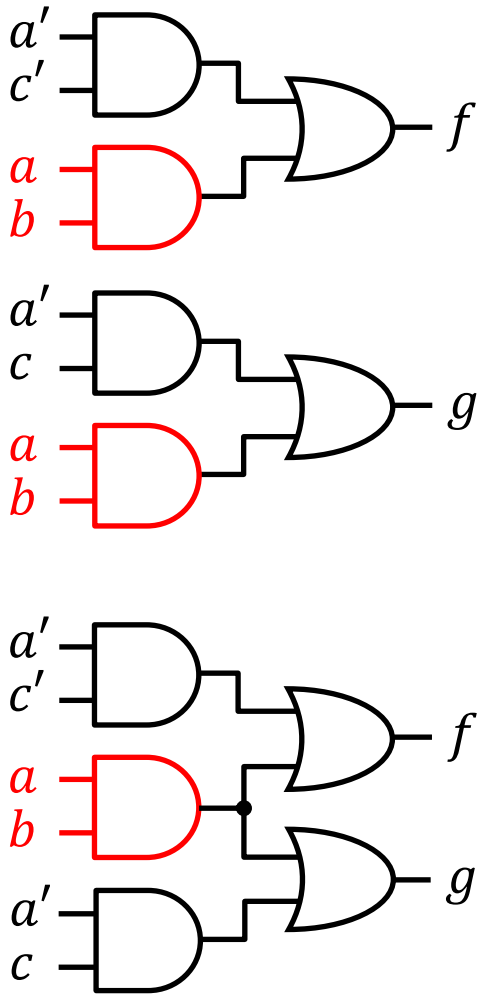
$f = a'c' + ab$

K-Map of g

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

$g = a'c + ab$

Common Term = ab



One circuit per function

One circuit with two Outputs

Multiple Outputs: Example 2

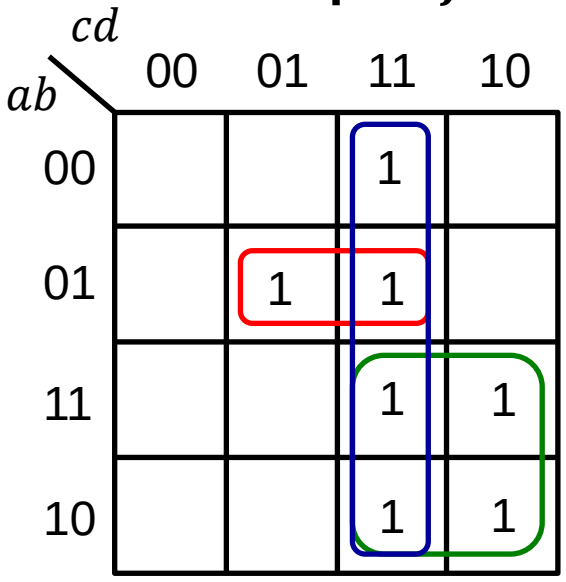
$f(a,b,c,d) = \sum(3,5,7,10,11,14,15)$, $g(a,b,c,d) = \sum(1,3,5,7,10,14)$

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

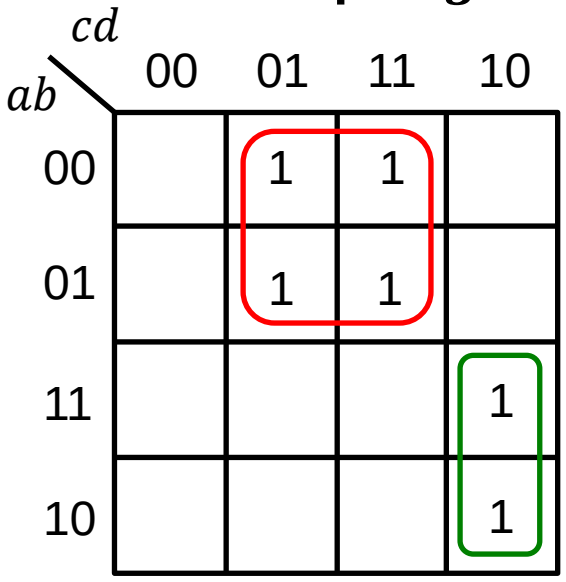
$f = a'bd + ac + cd$ $\quad\quad$ $g = a'd + acd'$

Extract the common terms of f and g

K-Map of f



K-Map of g



Common Terms

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

Minimal f and g

$f = T_1 b + T_2 + cd$

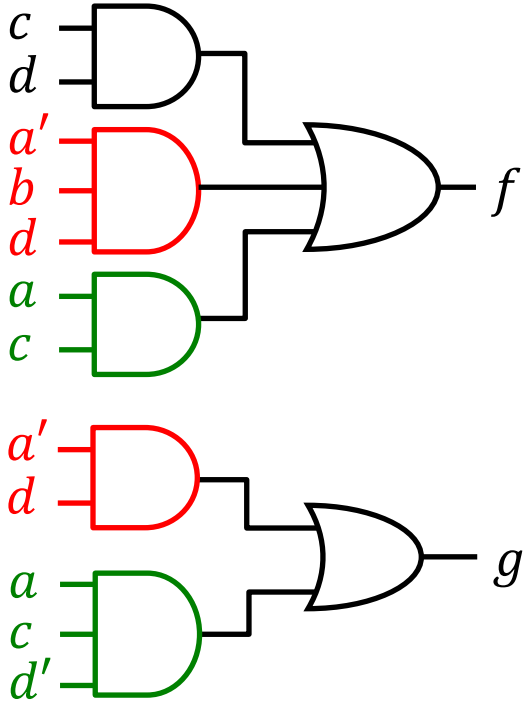
$g = T_1 + T_2 d'$

Common Terms ➔ Shared Gates

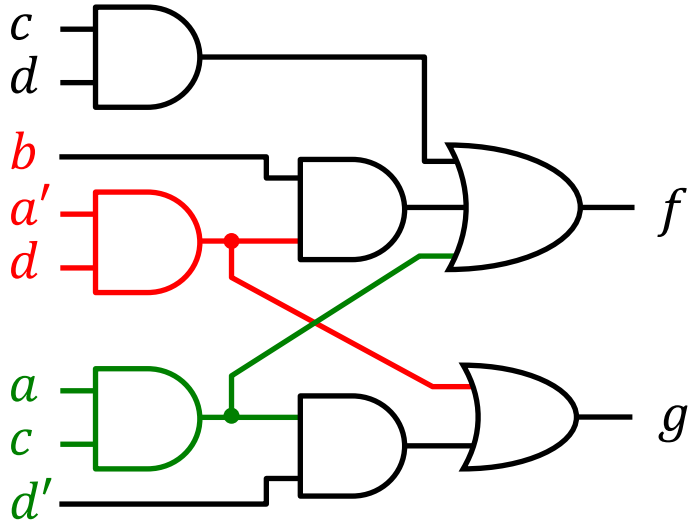
Minimal $f = a'bd + ac + cd$ Minimal $g = a'd + acd'$

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

Quine-McCluskey Method

- The Karnaugh map method described in Unit 5 is an effective way to simplify switching functions which have a small number of variables
- A systematic solution to K-Map when more complex function with more literals is given
- In principle, can be applied to an arbitrary large number of inputs
- One can translate Quine-McCluskey method into a computer program to perform minimization

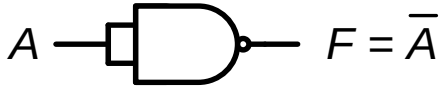
Universal Gates

- NAND and NOR gates are said to be *universal* gate because any logic circuit can be implemented with it.
  - Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.
- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.
- Because of the prominence of NAND and NOR gates in the design of digital circuits, rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR, and NOT into equivalent NAND and NOR logic diagrams

Universal Gates

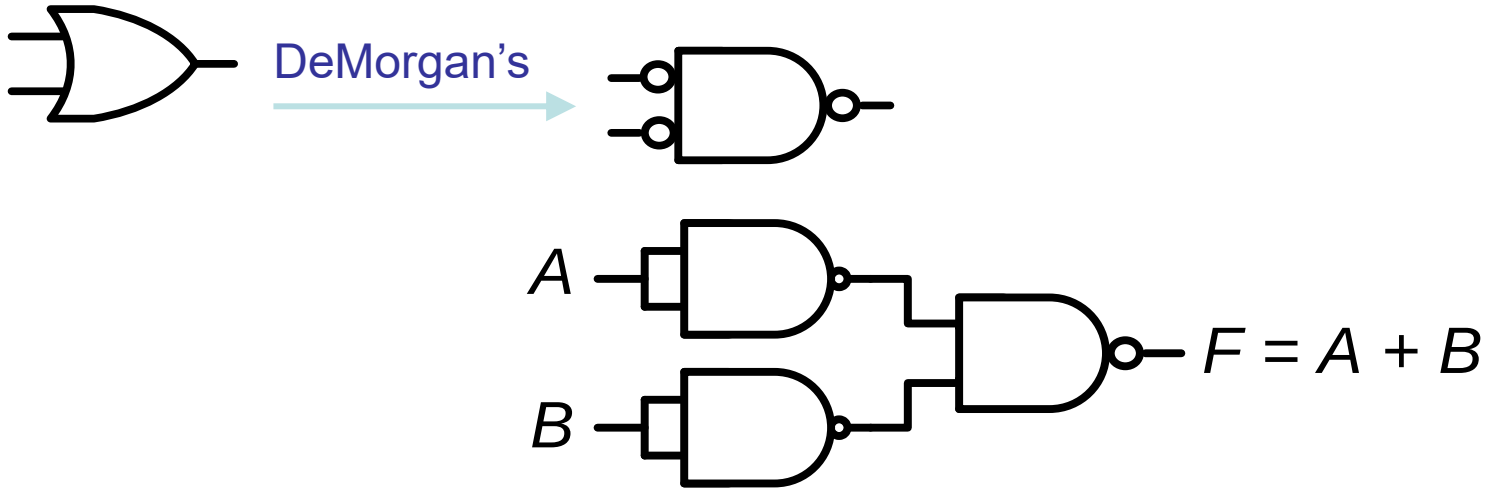
❖ **NAND** Gate

✧ NOT:



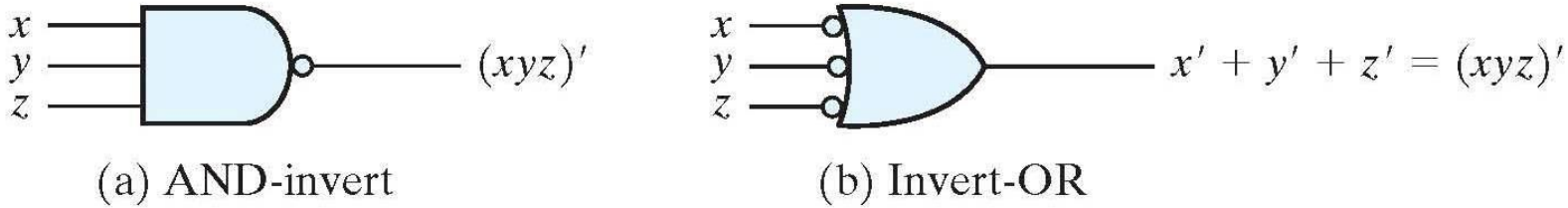
✧ AND:

✧ OR:



NAND Gate Symbol

- Two equivalent graphic symbols for the NAND gate
  - The AND-invert symbol has been defined previously and consists of an AND graphic symbol followed by a small circle negation indicator referred to as a bubble.
  - Alternatively, it is possible to represent a NAND gate by an OR graphic symbol that is preceded by a bubble in each input.



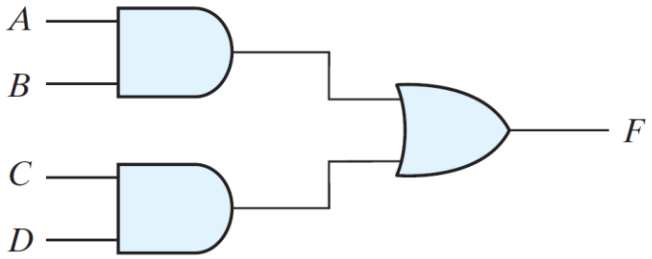
(a) AND-invert (b) Invert-OR

Two-Level NAND Implementation

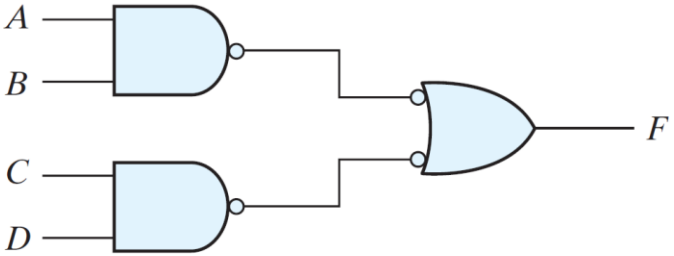
- **A convenient way to implement a Boolean function with NAND gates is to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic.**
- **The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form.**
- Procedure:
  - Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of first-level gates.
  - Draw a single gate using the AND-invert or the invert-OR graphic symbol in the second level, with inputs coming from outputs of first-level gates.
  - A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate.

Two-Level NAND Implementation

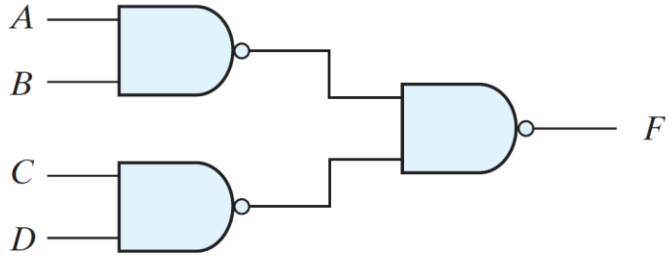
$$F = AB + CD$$



(a)



(b)



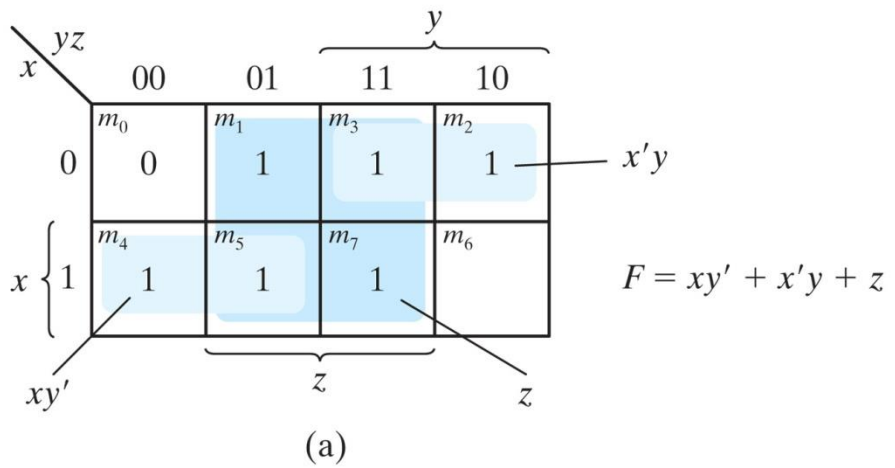
(c)

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

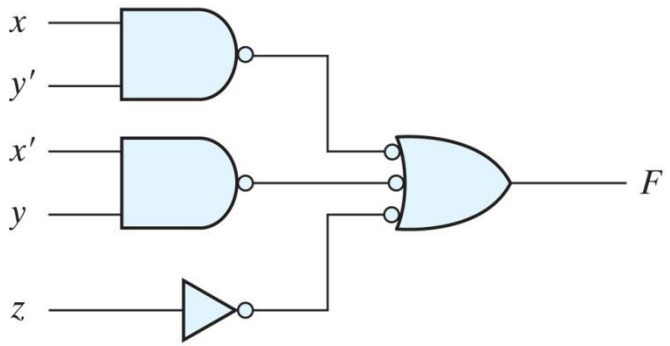
Boolean function with NAND gates

Implement the following Boolean function with NAND gates:

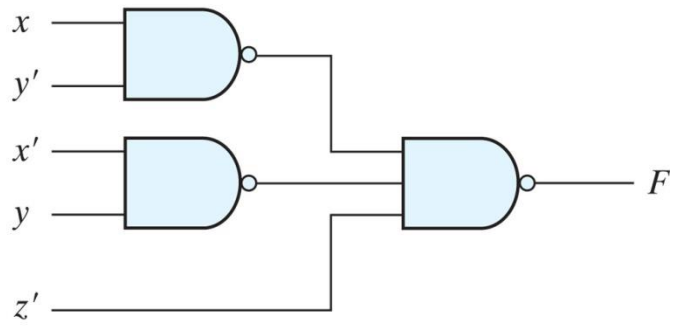
$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$



(a)



(b)



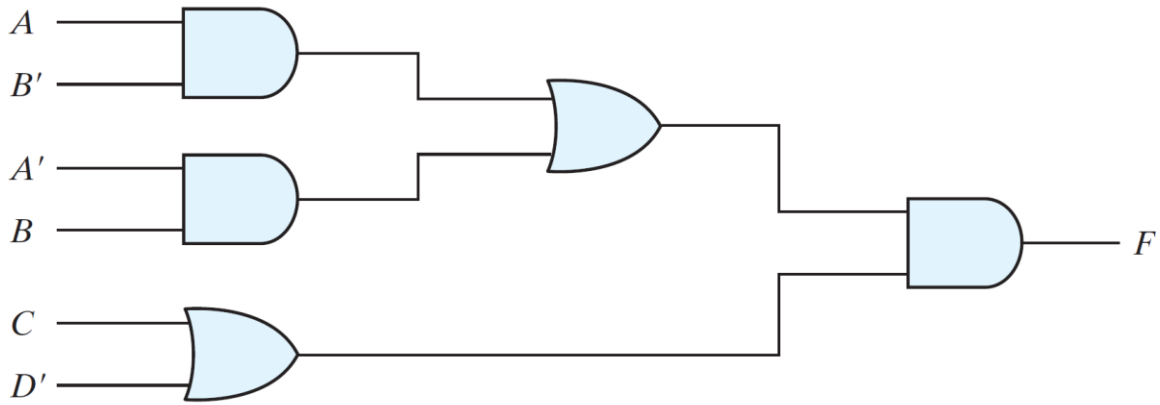
(c)

Multilevel NAND Implementation

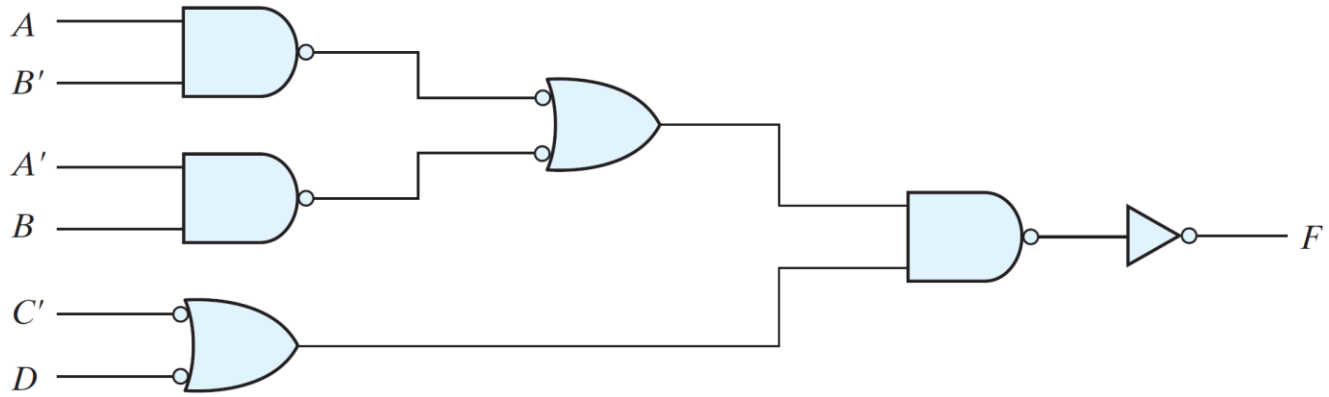
- General Procedure for converting a multilevel AND–OR diagram into an all-NAND diagram using mixed notation is as follows:
  - Convert all AND gates to NAND gates with AND-invert graphic symbols.
  - Convert all OR gates to NAND gates with invert-OR graphic symbols.
  - Check all the bubbles in the 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 NAND Implementation

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



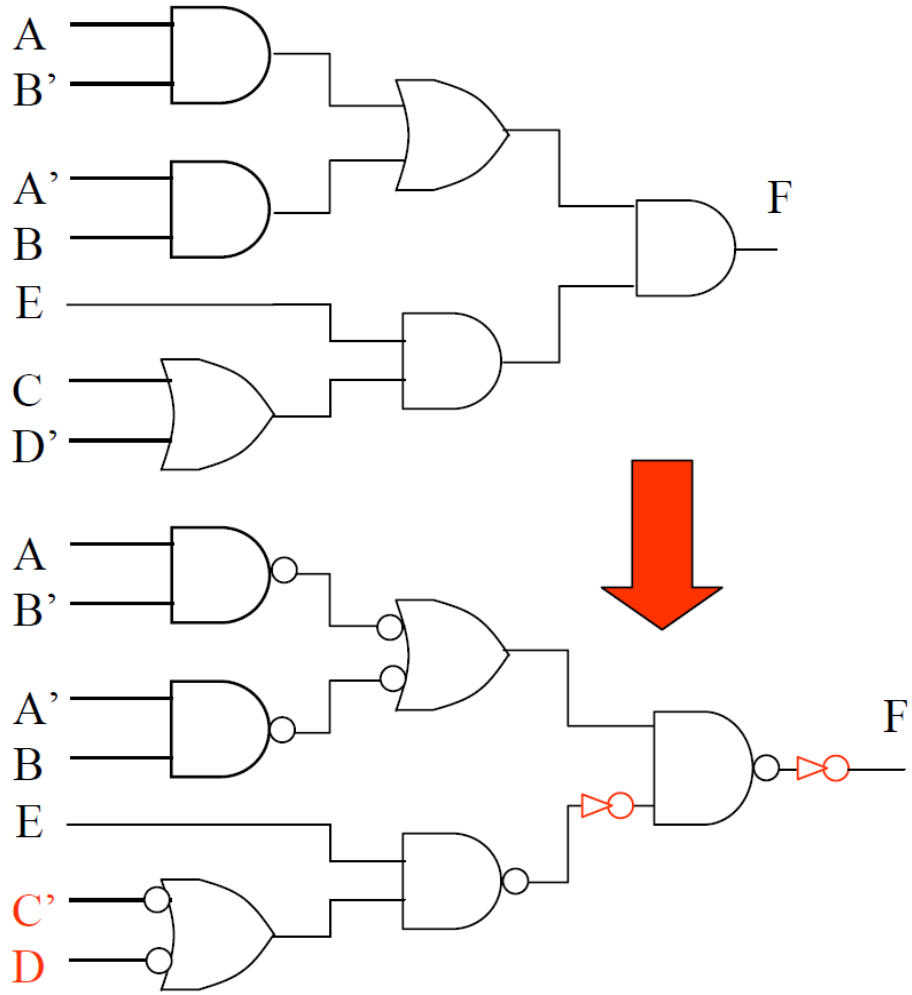
(a) AND–OR gates



(b) NAND gates

Multilevel NAND Implementation

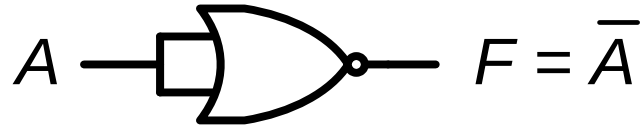
- Consider the function F = (AB'+A'B)E(C+D')



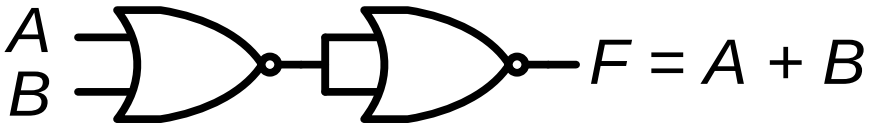
Note: an inverter is added to three paths

NOR Gates

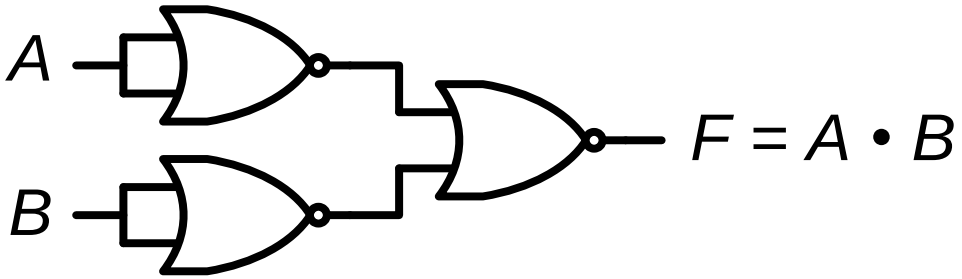
- The NOR operation is the dual of the NAND operation. Therefore, all procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic.
  - NOT:



  - OR:

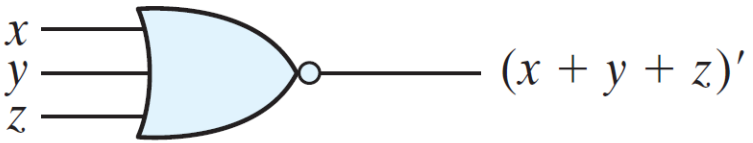


  - AND:

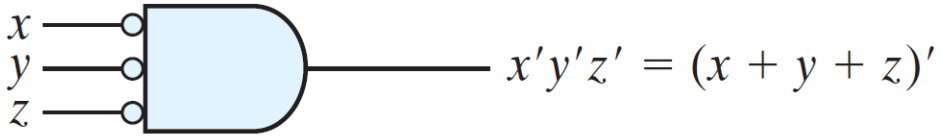


NOR Gate Symbol

- The two graphic symbols for the mixed notation.
  - The OR-invert symbol defines the NOR operation as an OR followed by a complement.
  - The invert-AND symbol complements each input and then performs an AND operation.



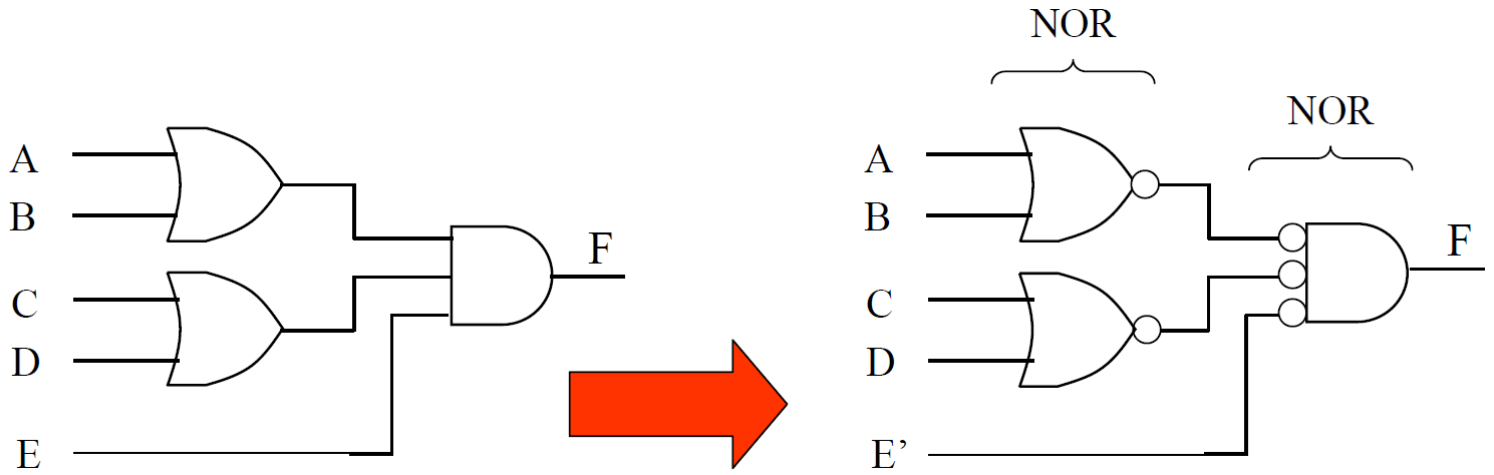
(a) OR-invert



(b) Invert-AND

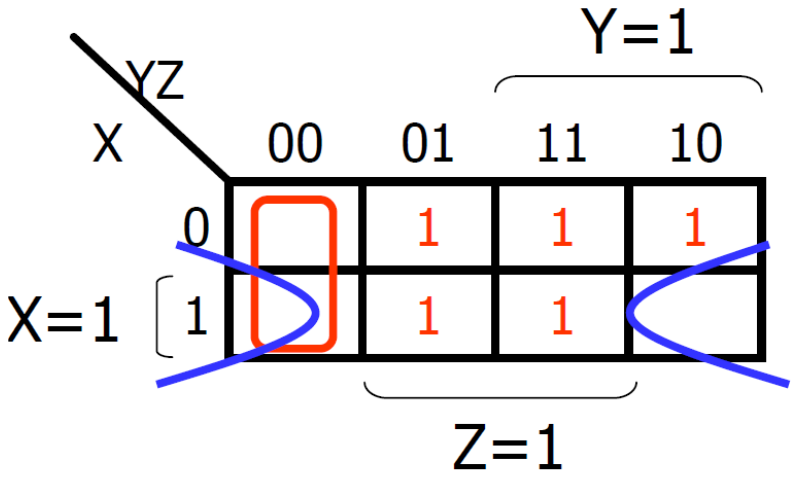
Two-Level NOR Implementation

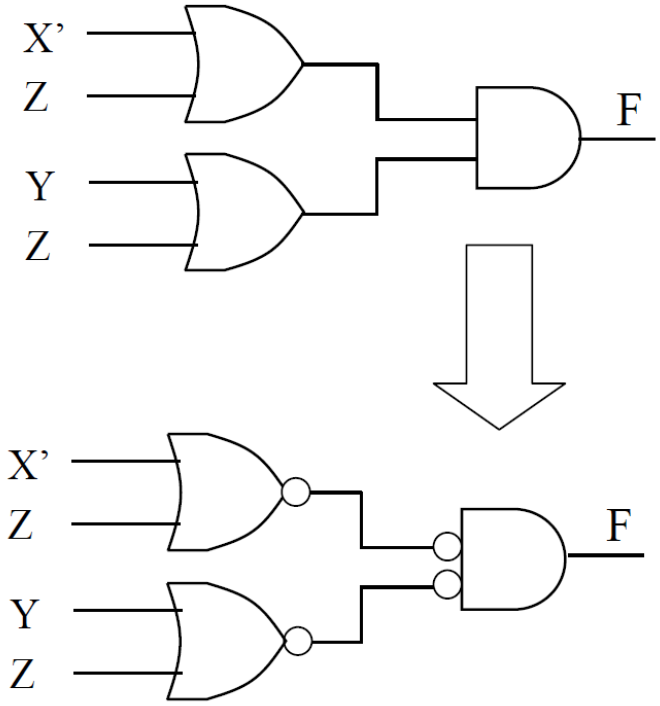
- Rules for 2-Level NOR Implementations
  - Simplify the function and express it in product of sums form
  - Draw a NOR gate (using OR-NOT symbol) for each sum term (with 2 literals or more)
  - Draw a single NOR gate (using NOT-AND symbol) the 2nd level (in place of the AND gate)
  - A term with single literal requires a NOT



Two-Level NOR Implementation

- Consider F =Σm(1,2,3,5,7) – Implement using NOR gates



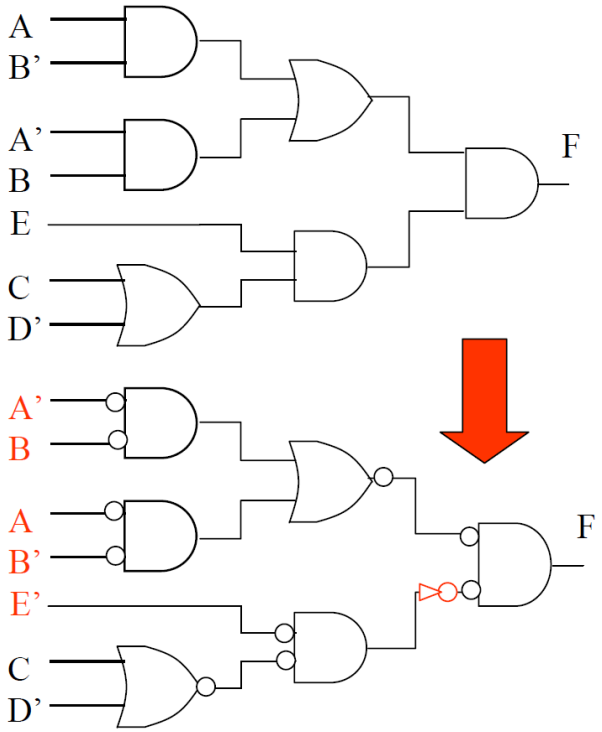


F′(X,Y) = Y′Z′+XZ′, or
F(X,Y) = (Y+Z)(X′+Z)

Rules for Multi-Level NOR Implementations

- NOTE: the function is NOT in the standard form – WHY?
- Steps:
  - Draw a NOR (OR-NOT) gate for each OR gate
  - Draw a NOR (NOT-AND) gate for each AND gate
  - Check paths – add inverters to make even number of bubbles

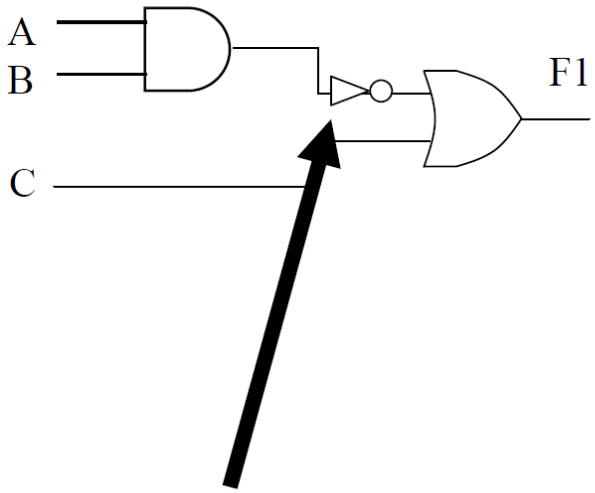
- Consider the function F = (AB'+A'B)E(C+D')



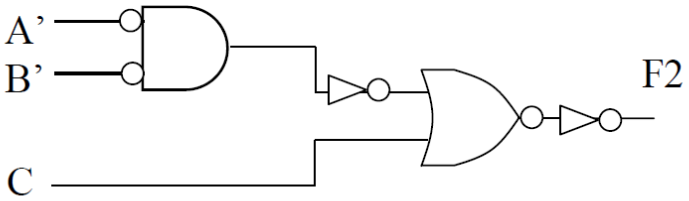
Note: an inverter is added to six paths

Multilevel NOR Implementation

- Consider the function F = (AB)'+C



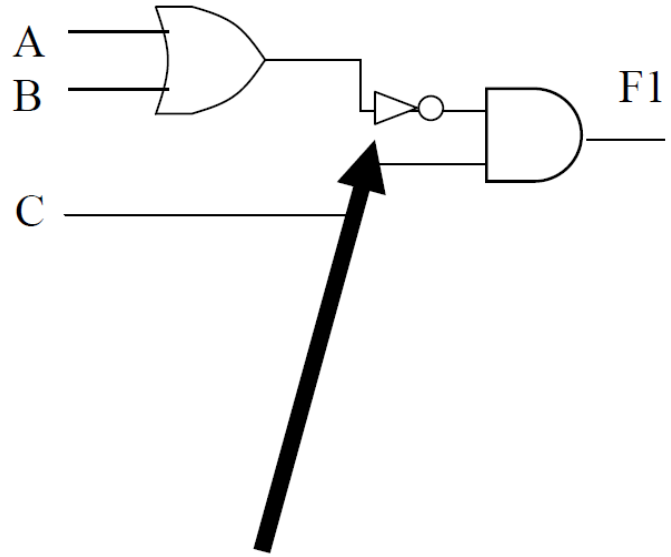
•Hence this bubble need not be complemented
•Only the ones added!!

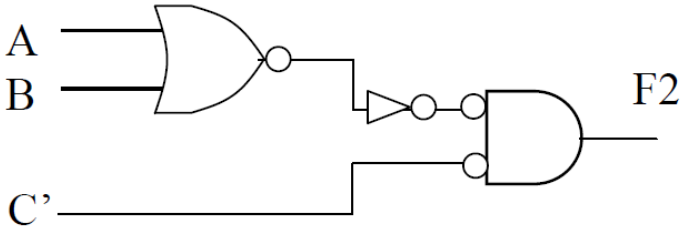


F2 = ((A''B'')'+C)''
= (AB)'+C
= F1

Multilevel NOR Implementation

- Consider the function F = (A+B)'C





- Hence this bubble need not be complemented
- Only the ones added!!

$$F2 = (A+B)'''C'' = (A+B)'C' = (A+B)'\ C = F1$$

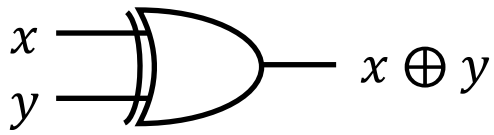
Exclusive OR / Exclusive NOR

- Exclusive OR (XOR) is an important Boolean operation used extensively in logic circuits
- Exclusive NOR (XNOR) is the complement of XOR

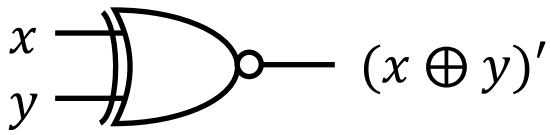
x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

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

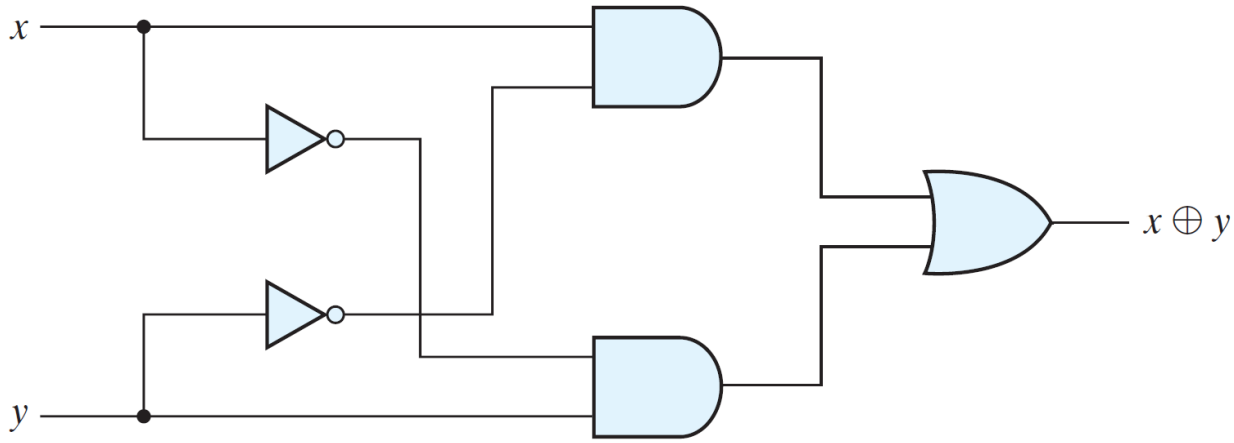
XOR / XNOR Functions

- The XOR function is: $x \oplus y = xy' + x'y$
- The XNOR function is: $(x \oplus y)' = xy + x'y'$
- XOR and XNOR gates are complex
  - Can be implemented as a true gate, or by
  - Interconnecting other gate types
- XOR and XNOR gates do not exist for more than two inputs
  - For 3 inputs, use two XOR gates
  - The cost of a 3-input XOR gate is greater than the cost of two XOR gates
- Uses for XOR and XNOR gates include:
  - Adders, subtractors, multipliers, counters, incrementers, decrementers
  - Parity generators and checkers

XOR Implementations

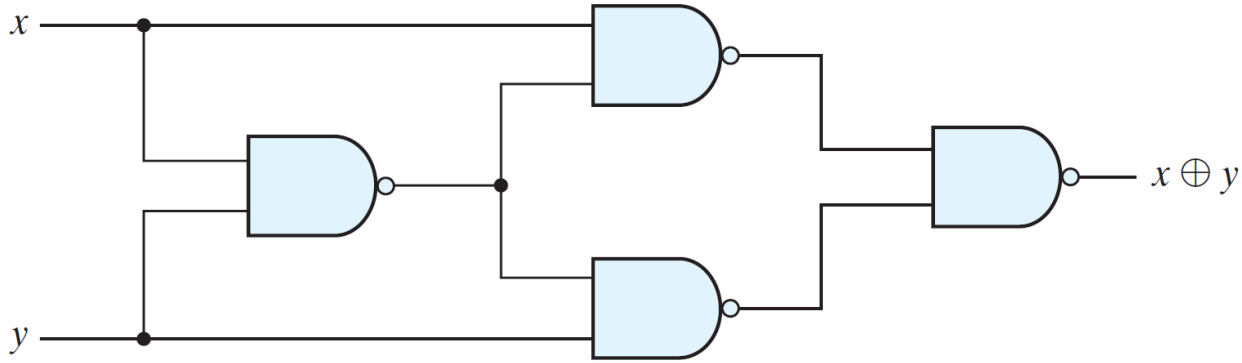
SOP implementation for XOR:

$$X \oplus Y = \bar{X}\,Y + X\,\bar{Y}$$



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

NAND only implementation for XOR:



(b) Exclusive-OR with NAND gates

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

XOR and XNOR Properties

- $x \oplus 0 = x$
- $x \oplus 1 = x'$
- $x \oplus x = 0$
- $x \oplus x' = 1$
- $x \oplus y = y \oplus x$
- $x' \oplus y' = x \oplus y$
- $(x \oplus y)' = x' \oplus y = x \oplus y'$

XOR and XNOR are **associative** operations

- $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$
- $\big((x \oplus y)' \oplus z\big)' = (x \oplus (y \oplus z)')' = x \oplus y \oplus z$

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

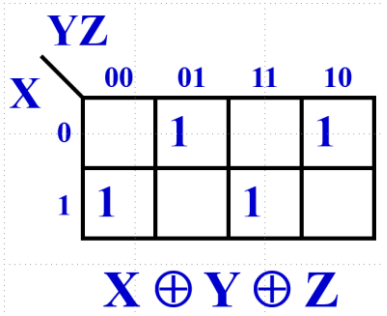
Odd Function with 3 inputs

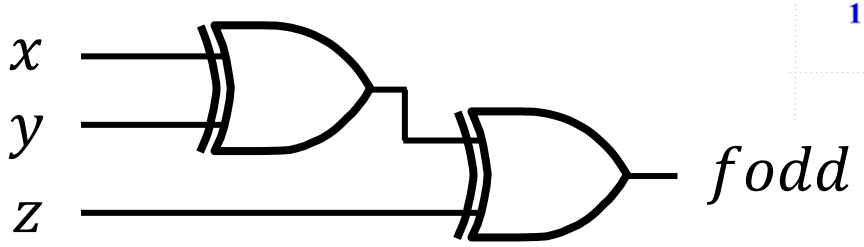
x y z	fodd
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

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

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

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





Implementation using two XOR gates

Even Function

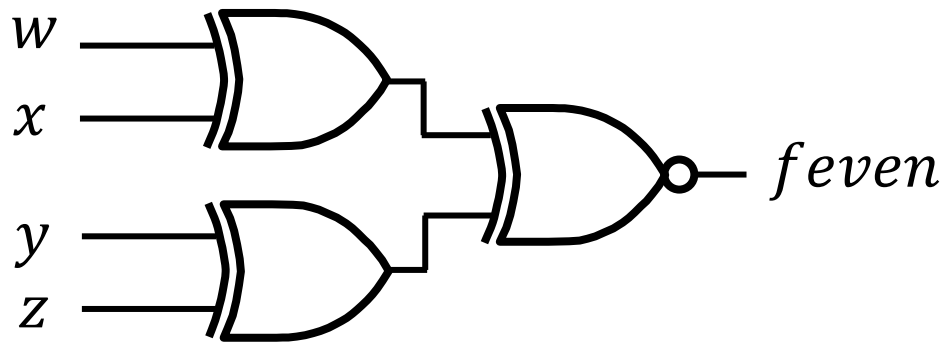
Even Function with 4 inputs

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

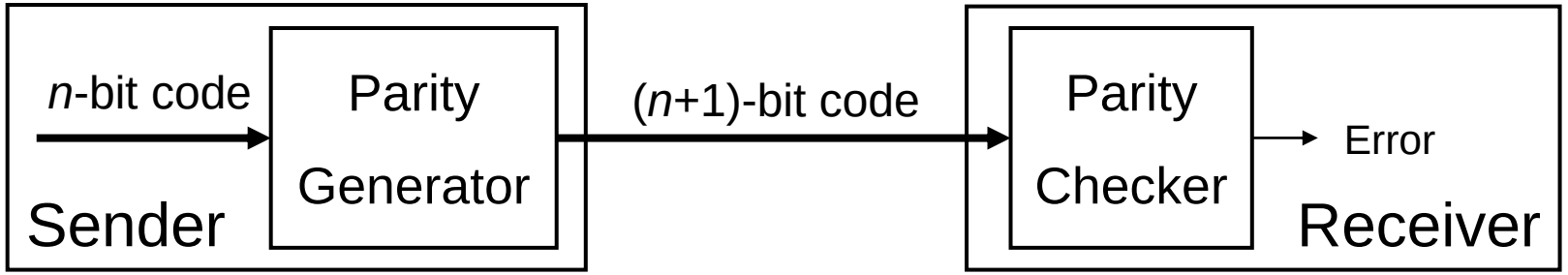
$$feven = \sum(0, 3, 5, 6, 9, 10, 12, 15)$$

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



Implementation using two XOR gates and one XNOR

Parity Generators and Checkers



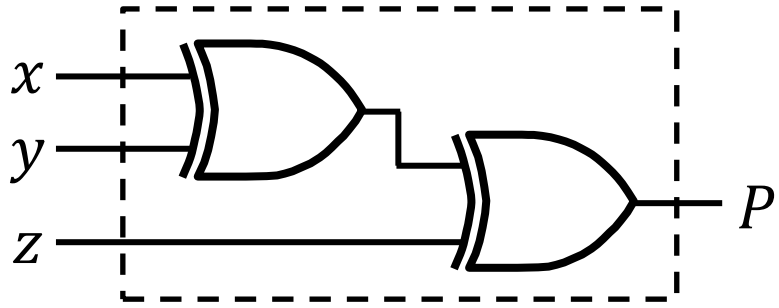
- A parity bit is added to the *n*-bit code
  - Produces (*n*+1)-bit code with an odd (or even) count of 1's
- **Odd parity:** count of 1's in the (*n*+1)-bit code is **odd**
  - Use an **even function** to generate the **odd parity bit**
  - Use an **even function** to check the (*n*+1)-bit code
- **Even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