ITI1100 Section z
Digital Systems I

Chapter 3:

Gate–Level minimization (2)

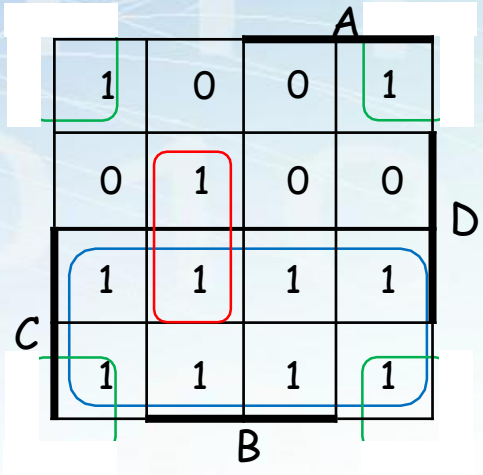
Prof. Mohammad Alja'Afreh

Summer 2019

K-map: 4-variable Examples

F(A,B,C,D) = Σ(0,2,3,5,6,7,8,10,11,14,15)

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



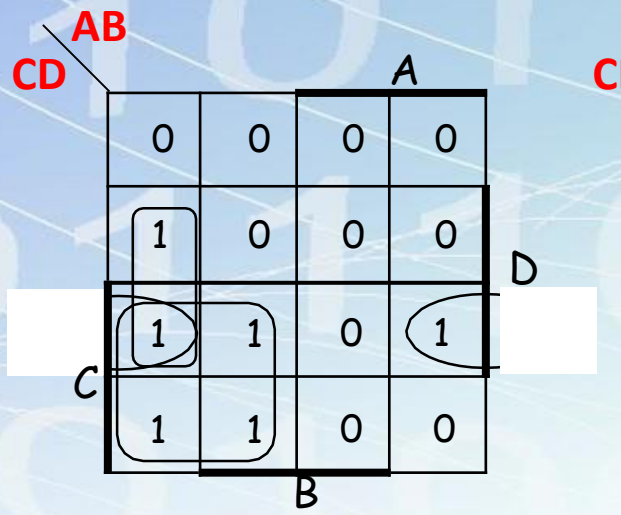
CD / AB	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

=

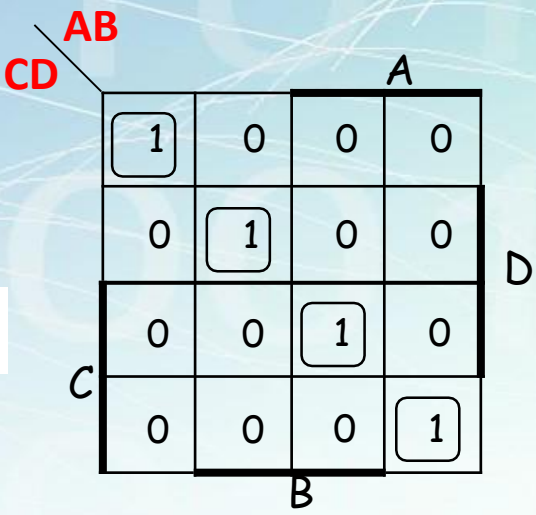
AB / CD	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

Note: around squares (loops) are adjacent squares

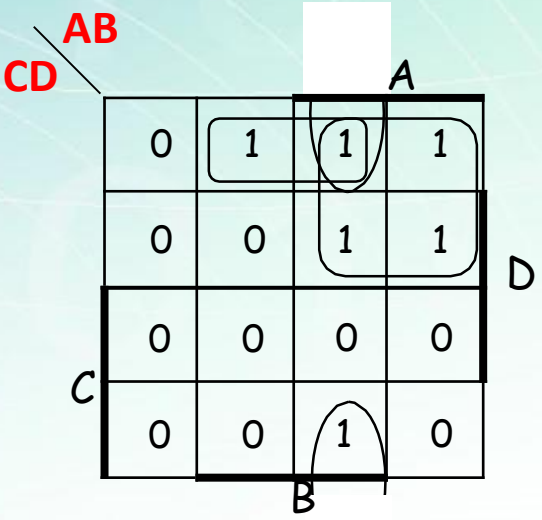
K-map: 4-variable Examples (cont'd)



K-map for G



K-map for Z



K-map for Y

G = A' B' D + A' C + B' C D

Z = A'B'C'D' + A'BC'D + ABCD + AB'CD' (can not be simplified)

Y = B C' D' + A C' + A B D'

Prime Implicants

Any single 1 or group of 1s in the Karnaugh map of a function **F** is an ***implicant*** of **F**.

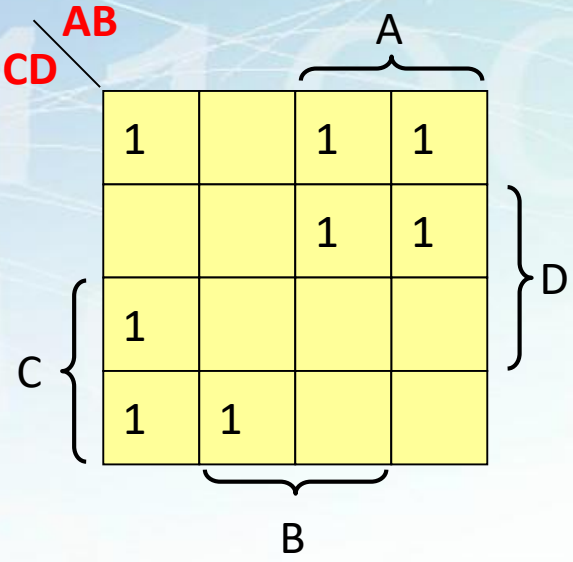
A product term is called a ***prime implicant*** of **F** if it cannot be combined with another term to eliminate a variable.

A product term is an ***essential prime implicant*** of **F** if there is a minterm that is only covered by that prime implicant.

The minimal sum-of-products form of **F** must include all the essential prime implicants of **F**.

Prime Implicants

Example:



If a function **F** is represented by this Karnaugh Map. Which of the following terms are implicants of **F**, and which ones are prime implicants of **F**?

(a) AC'D'	Implicant
(b) BD	Not an Implicant
(c) A'B'C'D'	Implicant
(d) AC'	Prime Implicant
(e) B'C'D'	Prime Implicant

Simplifying logic functions using Karnaugh maps … looping

- The logic function can be simplified by replacing canonical terms with a minimum cover of prime implicants, obtained by properly combining squares (**looping**) of the Karnaugh maps which contain **1**s.
- Looping a pair of adjacent **1**s eliminates the variable that appears in both direct and complemented form.

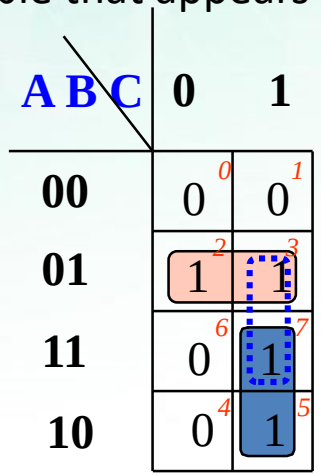
PI: AC, BC, A'B

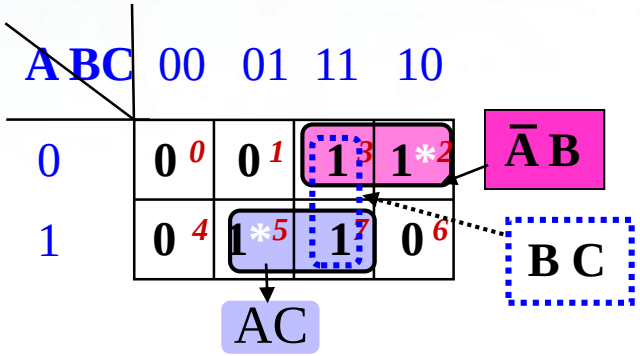
EPI: AC, A'B

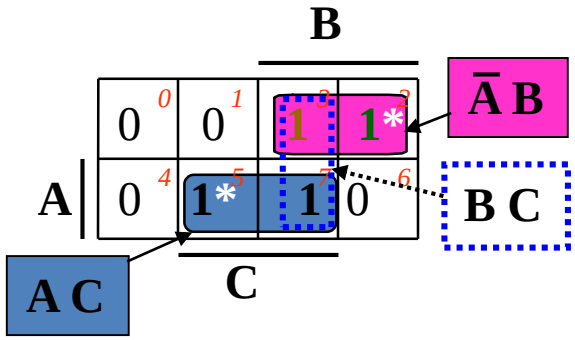
Redundant Implicant: **BC**

	A	B	C	F
(0)	0	0	0	0
(1)	0	0	1	0
(2)	0	1	0	1
(3)	0	1	1	1
(4)	1	0	0	0
(5)	1	0	1	1
(6)	1	1	0	0
(7)	1	1	1	1

$$F = \sum(2,3,5,7) = AC + \bar{A}B$$







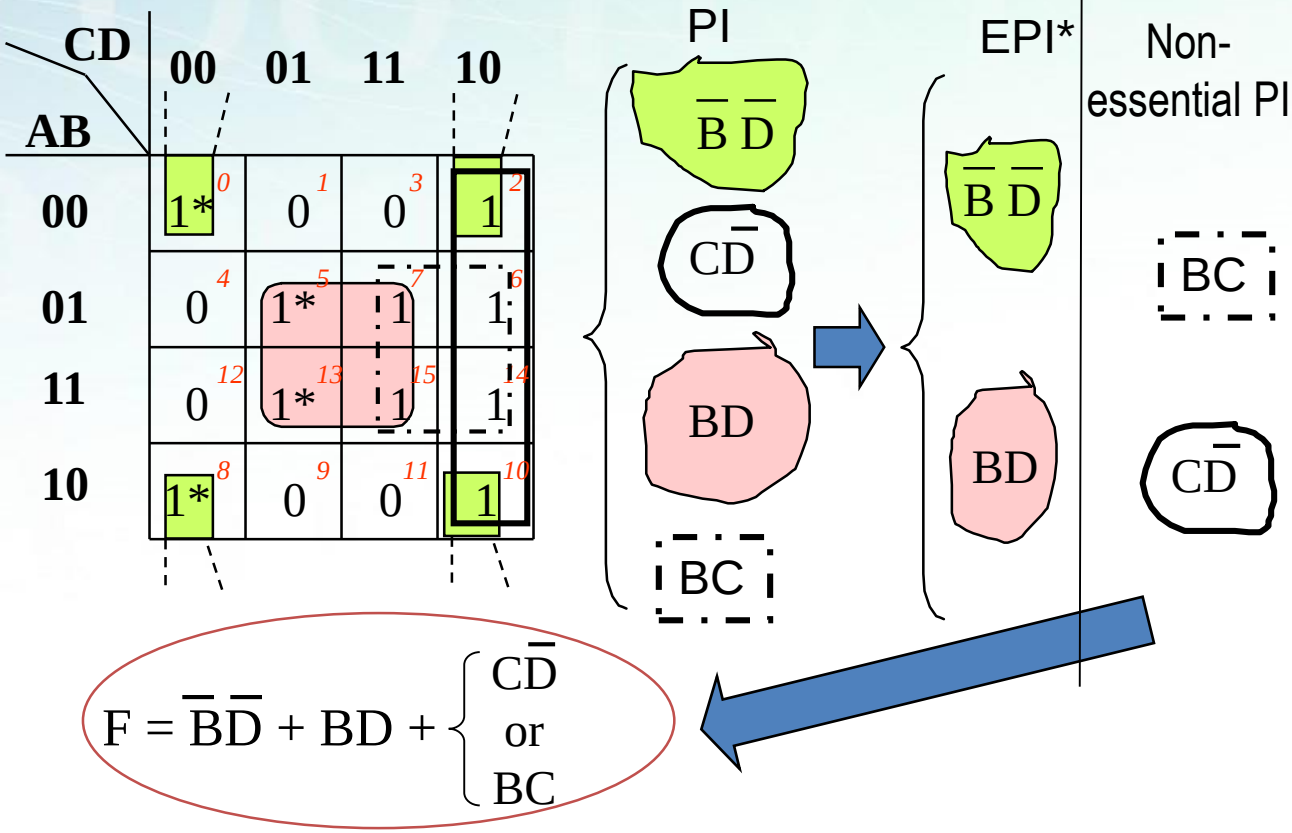
Simplifying logic functions using Karnaugh maps … more looping

$$F = \sum(0,2,5,6,7,8,10,13,14,15) =$$

$$= \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}D + ABC\bar{D} + ABCD$$

	A	B	C	D	F
(0)	0	0	0	0	1
(1)	0	0	0	1	0
(2)	0	0	1	0	1
(3)	0	0	1	1	0
(4)	0	1	0	0	0
(5)	0	1	0	1	1
(6)	0	1	1	0	1
(7)	0	1	1	1	1
(8)	1	0	0	0	1
(9)	1	0	0	1	0
(10)	1	0	1	0	1
(11)	1	0	1	1	0
(12)	1	1	0	0	0
(13)	1	1	0	1	1
(14)	1	1	1	0	1
(15)	1	1	1	1	1

Looping a *quad* of adjacent **1**s eliminates the two variables that appears in both direct and complemented form.

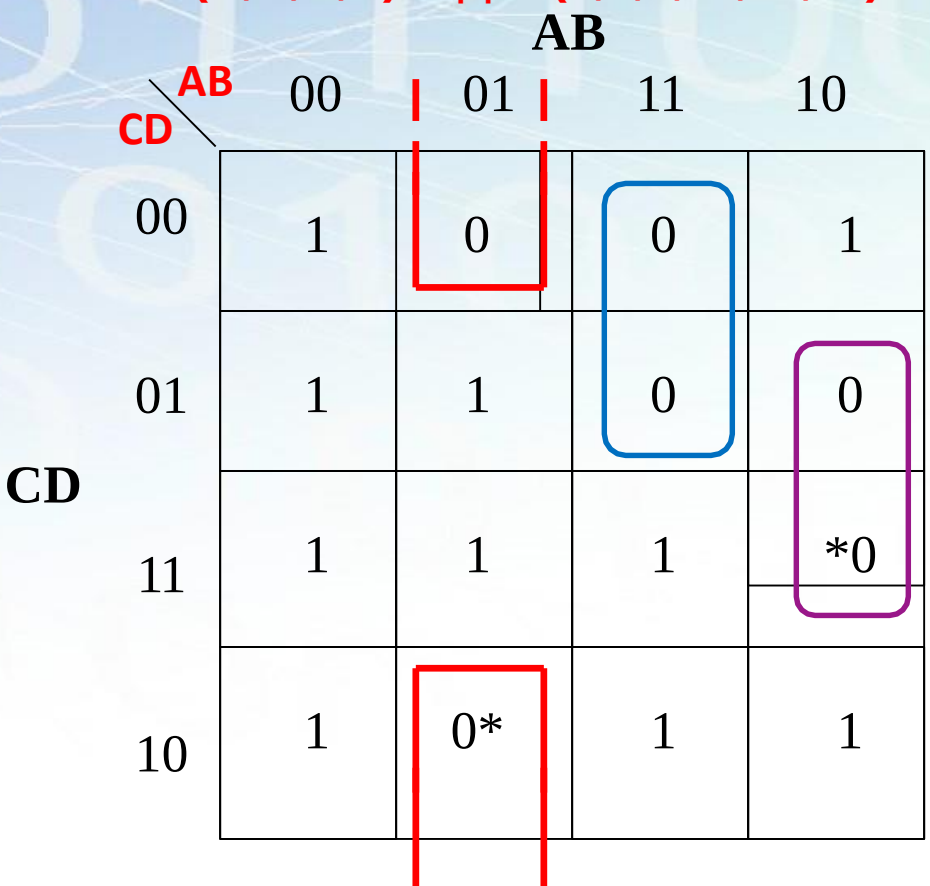


Product-of-Sums Simplification

- **The best way to show this is by the following example:**

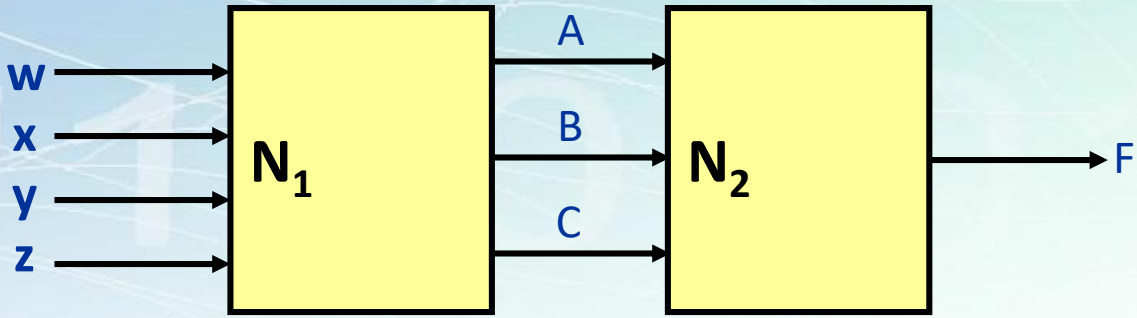
F(A,B,C,D) = Σ m(0,1,2,3,5,7,8,10,14,15)

F(A,B,C,D) = ∏ m(4,6,9,11,12,13)



F(A,B,C,D) = (A+B′+D) (A′+B+D′) (A′+B′+C)

Incompletely Specified Functions



Assumption: it never produces the combinations ABC = 001 and ABC = 110.

Question: What value should F produce for the combinations ABC = 001 and ABC = 110?

We don't care!!!

Incompletely Specified Functions (Cont.)

How can we use the fact that we don't care about the value produced by F when ABC = 001 or ABC = 110 to simplify the circuit N_2 ?

Given that N_2 gives TRUE at m_0, m_2, m_3 and m_7

A	B	C	F
0	0	0	1
0	0	1	~~X~~ 0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	~~X~~ 0
1	1	1	1

If we assume that F(0,0,1) = 0 and F(1,1,0) = 0, we obtain the following equation

$$F(A,B,C) = A'B'C' + A'BC' + A'BC + ABC$$
$$= A'C'(B' + B) + (A' + A)BC$$
$$= A'C' \cdot 1 + 1 \cdot BC$$
$$= A'C' + BC$$

Incompletely Specified Functions (Cont.)

A	B	C	F
0	0	0	1
0	0	1	~~X~~ 1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	~~X~~ 1
1	1	1	1

However, if we assume that F(0,0,1) = 1 and F(1,1,0) = 1, we obtain instead the equation:

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

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

$$= A'B' \cdot 1 + A'B \cdot 1 + AB \cdot 1$$

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

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

$$= A'(B' + B) + (A' + A)B$$

$$= A' \cdot 1 + 1 \cdot B$$

$$= A' + B$$

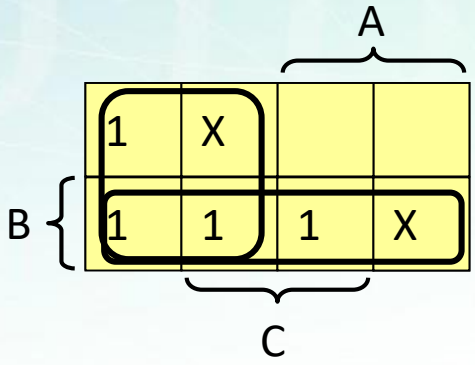
Compare this with the other solution: F(A,B,C) = A'C' + BC.

Which one is cheaper to implement?

Incompletely Specified Functions

Don’t care values are easily used to simplify incompletely specified functions.

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



F = A’ + B

Incompletely Specified Logic Functions

$F = \sum(0,1,3,7,8,12) + dc(5,10,13,14)$ ✦ *dc* -> x = *don't care* terms

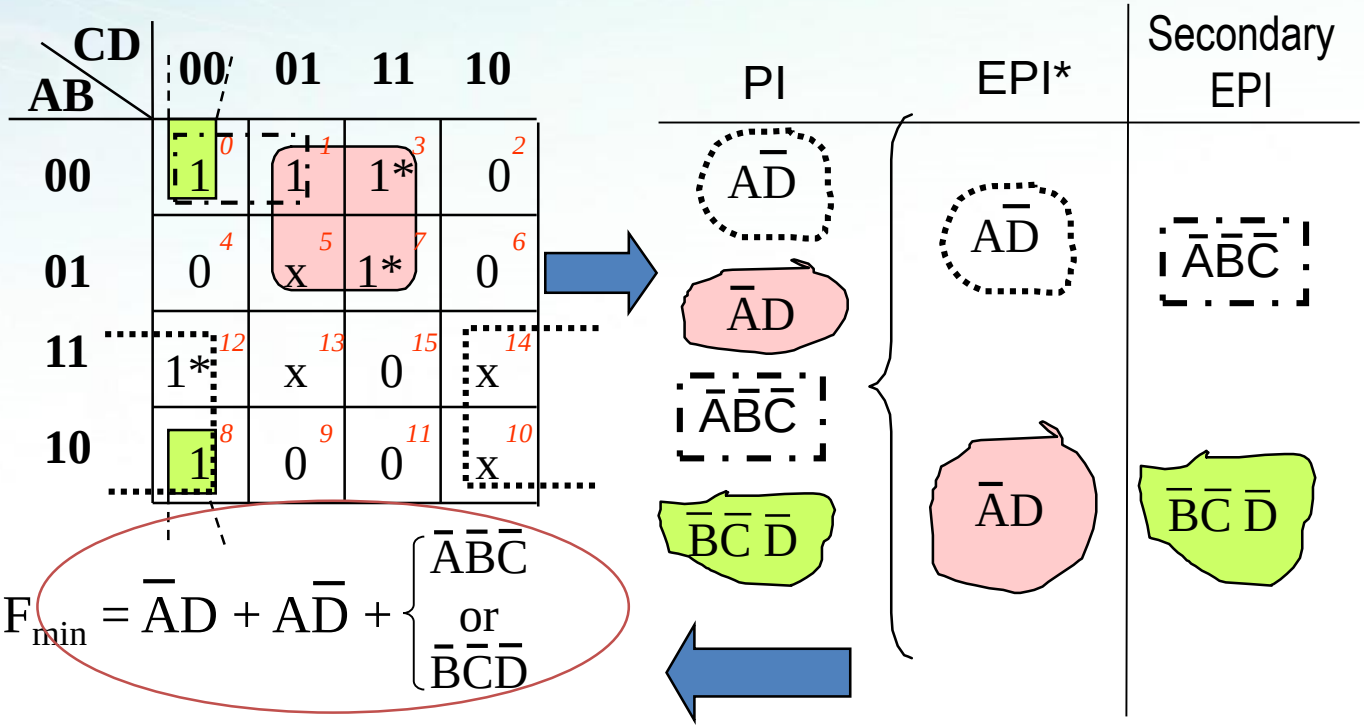
	A	B	C	D	F
(0)	0	0	0	0	1
(1)	0	0	0	1	1
(2)	0	0	1	0	0
(3)	0	0	1	1	1
(4)	0	1	0	0	0
(5)	0	1	0	1	x
(6)	0	1	1	0	0
(7)	0	1	1	1	1
(8)	1	0	0	0	1
(9)	1	0	0	1	0
(10)	1	0	1	0	x
(11)	1	0	1	1	0
(12)	1	1	0	0	1
(13)	1	1	0	1	x
(14)	1	1	1	0	x
(15)	1	1	1	1	0

Don't care terms can be treated as 1's or 0's, depending on which one is more advantageous.

So, if including a *don't care* in a group makes

1. the group bigger, or
2. makes it possible to reduce the number of groups,

the *don't care* should be included in the group, but not otherwise!

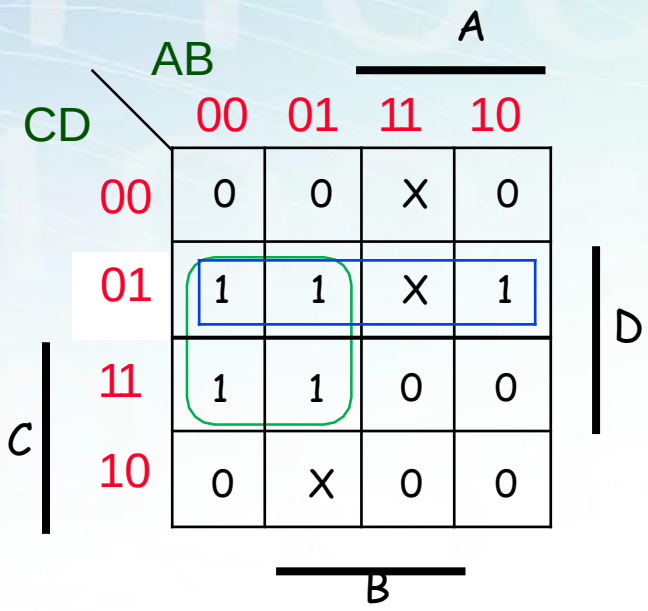


Don’t Cares Examples

F(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)

F(A,B,C,D)= A'D + B'C'D *without don’t cares*

F(A,B,C,D) =C'D+A'D *with don’t cares*



A	B	C	D	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	X
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	X
1	1	0	1	X
1	1	1	0	0
1	1	1	1	0

Example:
BCD to Braille Convertion

A	B	C	D	W Z	X Y
0	0	0	0	•	•
				•	•
0	0	0	1	•	
0	0	1	0	•	
				•	
0	0	1	1	•	•
0	1	0	0	•	•
					•
0	1	0	1	•	
					•
0	1	1	0	•	•
				•	
0	1	1	1	•	•
				•	•
1	0	0	0	•	
				•	•
1	0	0	1		•
				•	

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1
1	0	1	0	?	?	?	?
1	0	1	1	?	?	?	?
.	.	.	.	.	.	.	.
1	1	1	1	?	?	?	?

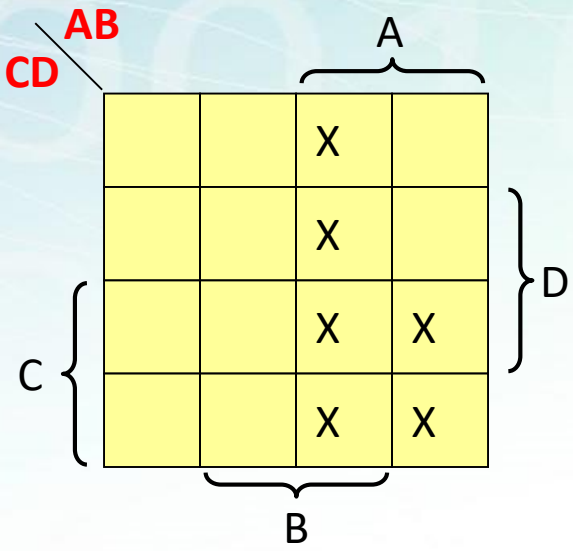
Example:
BCD to Braille Convertion

A	B	C	D	W Z	X Y
0	0	0	0	• •	• •
0	0	0	1	•	
0	0	1	0	• •	
0	0	1	1	•	•
0	1	0	0	•	• •
0	1	0	1	•	•
0	1	1	0	• •	•
0	1	1	1	• •	• •
1	0	0	0	• •	•
1	0	0	1	•	•

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
.	.	.	.	.	.	.	.
1	1	1	1	X	X	X	X

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

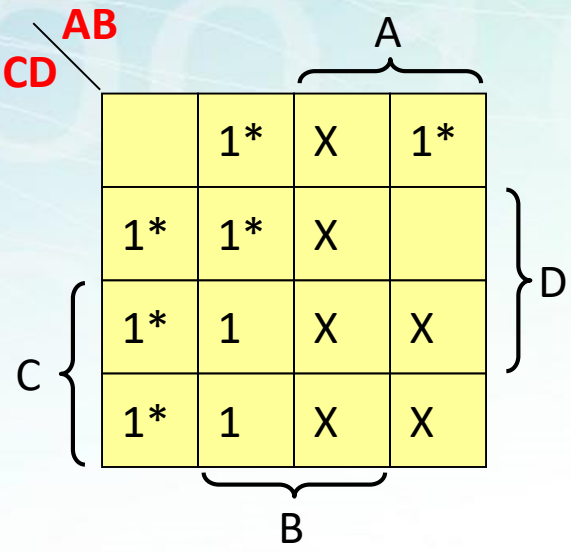


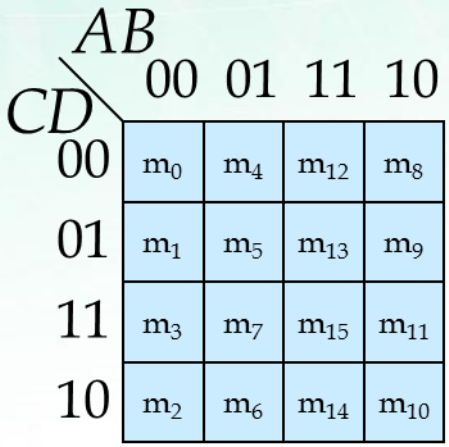
AB / CD	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

W = ?

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

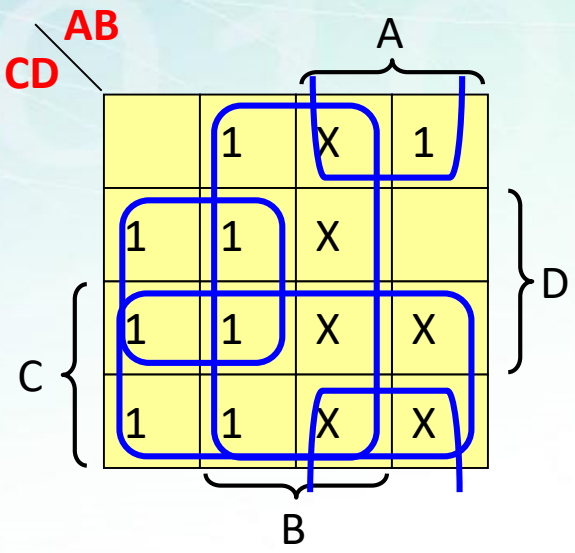




W = ?

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

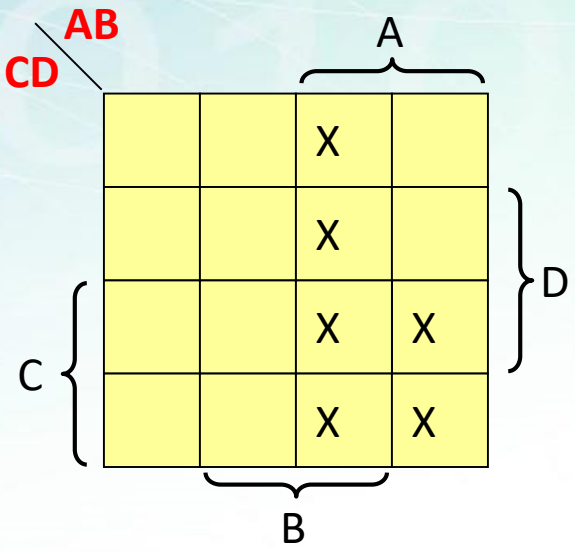


$W = B + C + A'D + AD'$

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'

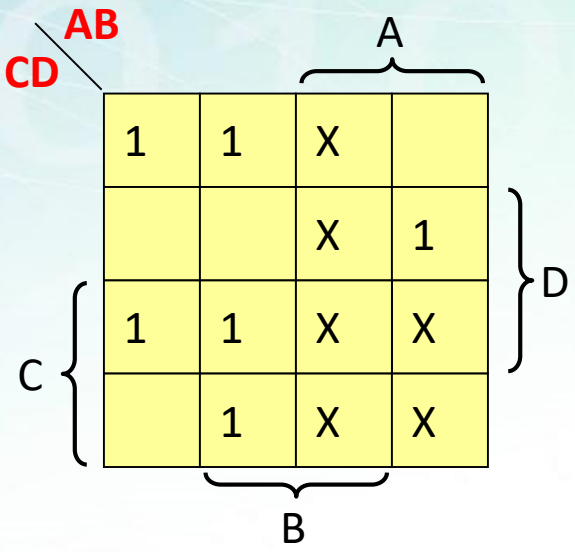


X = ?

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'

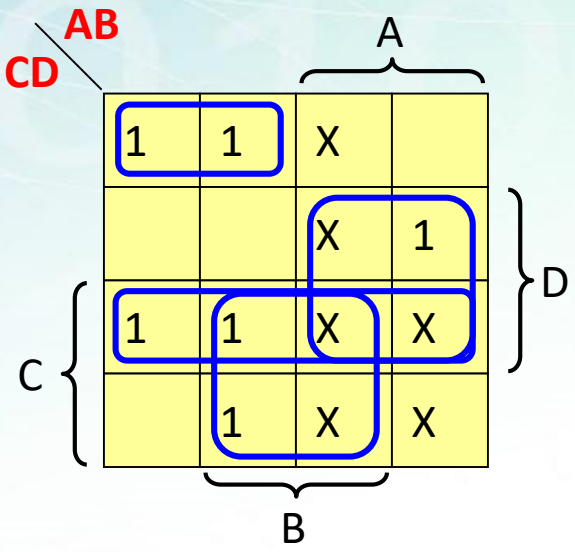


X = ?

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'



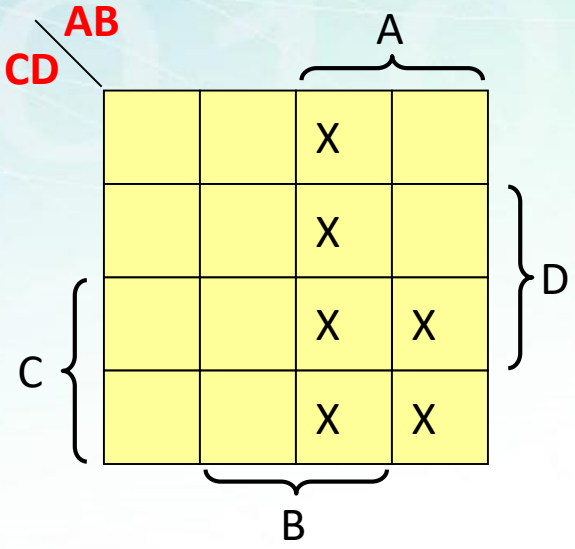
X = A'C'D' + CD + AD + BC

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'

X = A'C'D' + CD + AD + BC



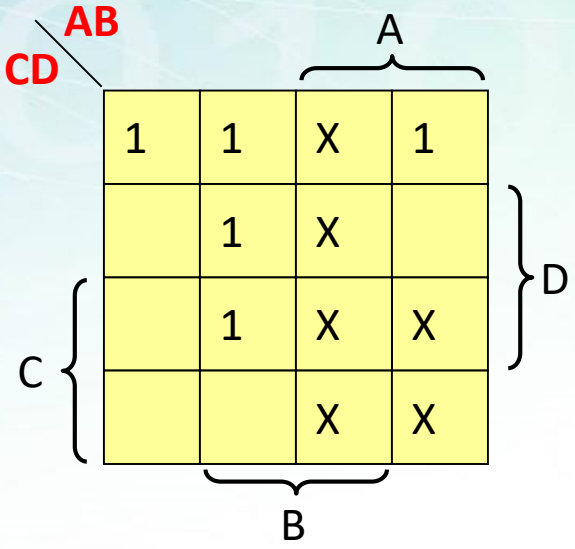
Y = ?

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'

X = A'C'D' + CD + AD + BC



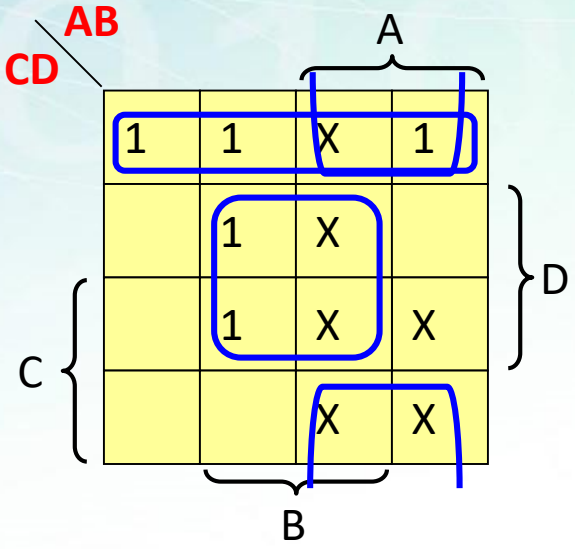
Y = ?

Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'

X = C'D' + CD + AD + BC



Y = A'C'D' + AD' + BD

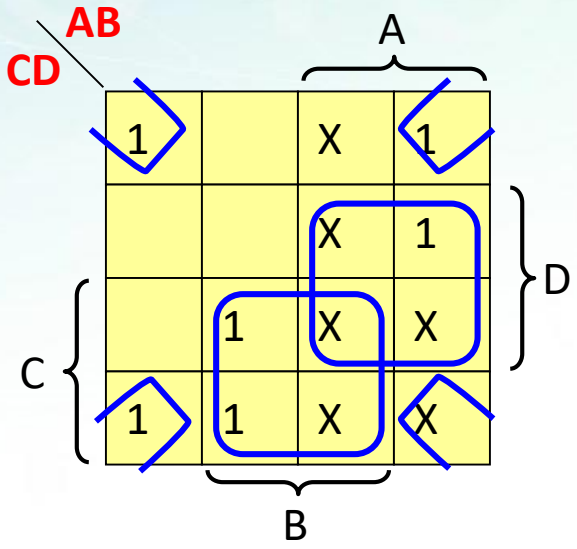
Example:
BCD to Braille Convertion

A	B	C	D	W	X	Y	Z
0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	0	1	0	1

W = B + C + A'D + AD'

X = A'C'D' + CD + AD + BC

Y = A'C'D' + AD' + BD



Z = AD + BC + B'D'

Example:
BCD to Braille Convertion

$W = B + C + A'D + AD'$

$X = A'C'D' + CD + AD + BC$

$Y = A'C'D' + AD' + BD$

$Z = AD + BC + B'D'$

How can we use these standard sum-of-product equations to implement a multiple-output NAND network?

We use the DeMorgan Law:

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

Hardware Implementation