Unit 3
Simplification of Boolean Functions

Karnaugh-Map (K-Map)

Introduction to K-Maps

- Simplification of Boolean functions leads to simpler (and usually faster) digital circuits.
- Simplifying Boolean functions using identities is time-consuming and error-prone.

- This special section presents an easy, systematic method for reducing Boolean expressions.

Karnaugh Maps (K-Maps)

- A K-Map is a matrix consisting of rows and columns that represent the output values of a Boolean function.

- The output values placed in each cell are derived from the minterms of a Boolean function.

- A *minterm* is a product term that contains all of the function's variables exactly once, either complemented or not complemented.

2 – Variable K-Map

The two variables A and B have four possible combinations that can be represented by the map as follows

A	B	Minterm
0	0	$m_0 = A'B'$
0	1	$m_1 = A'B$
1	0	$m_2 = AB'$
1	1	$m_3 = AB$

B

A 0 1

0 1

2

0

3 – Variable K-Map

The three variables A, B and C have eight possible combinations that can be represented by the map as follows

A	**B**	**C**	**Minterm**
0	0	0	m_0 = A'B'C
0	0	1	m_1 = A'B'C
0	1	0	m_2 = A'BC
0	1	1	m_3 = A'BC
1	0	0	m_4 = AB'C
1	0	1	m_5 = AB'C
1	1	0	m_6 = ABC'
1	1	1	m_7 = ABC

BC

A 00 10 01 11

0	1	3	2
4	5	7	6

0

1

Minterm Number

4 – Variable K-Map

The four variables A, B, C and D have sixteen possible combinations that can be represented by the map as follows

A	B	C	D	Minterm
0	0	0	0	$m_0 = A'B'C'D'$
0	0	0	1	$m_1 = A'B'C'D$
0	0	1	0	$m_2 = A'B'CD'$
0	0	1	1	$m_3 = A'B'CD$
0	1	0	0	$m_4 = A'BC'D'$
0	1	0	1	$m_5 = A'BC'D$
0	1	1	0	$m_6 = A'BCD'$
0	1	1	1	$m_7 = A'BCD$

A	B	C	D	Minterm
1	0	0	0	$m_8 = AB'C'D'$
1	0	0	1	$m_9 = AB'C'D$

1	0	1	0	m_{10} = AB’CD’
1	0	1	1	m_{11} = AB’CD
1	1	0	0	m_{12} = ABC’D’
1	1	0	1	m_{13} = ABC’D

1	1	1	0	m_{14} = ABCD’
1	1	1	1	m_{15} = ABCD

4 – Variable K-Map

CD

AB 00 10

01 11

00 01 11 10

4	5	7	6
12	13	15	14

0	1

8	9

Function plotting in K-Map

- Consider function

$$F = AB + A'B$$

B

A 0 1

0 0	1 1
2 0	3 1

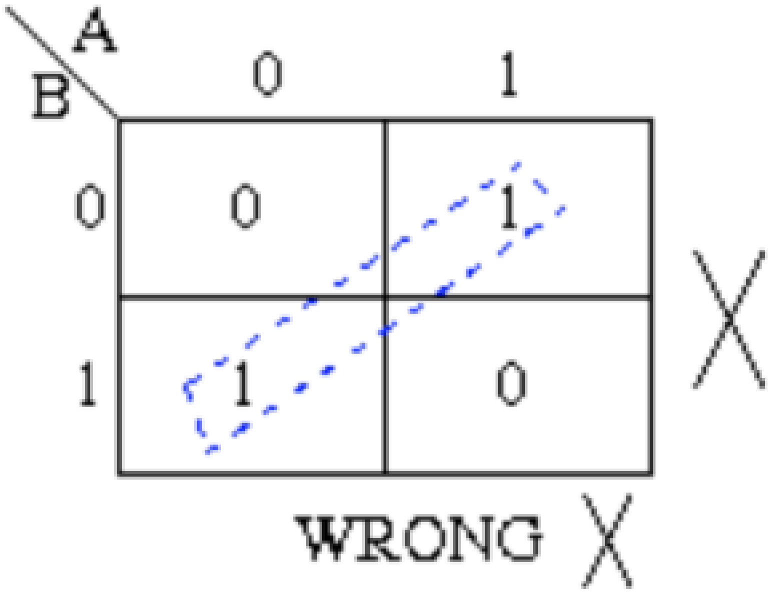
0

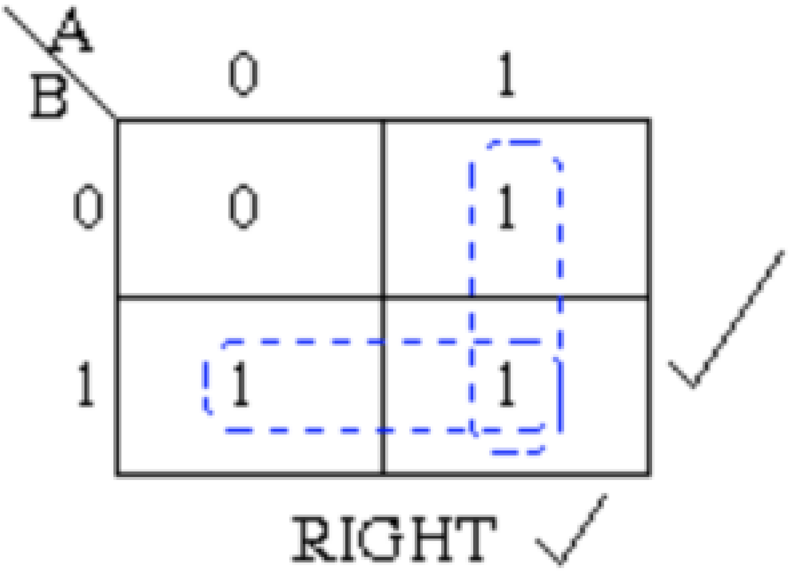
Karnaugh Maps - Rules of Simplification

- The Karnaugh map uses the following rules for the simplification of expressions by grouping together adjacent cells containing ones

Rules

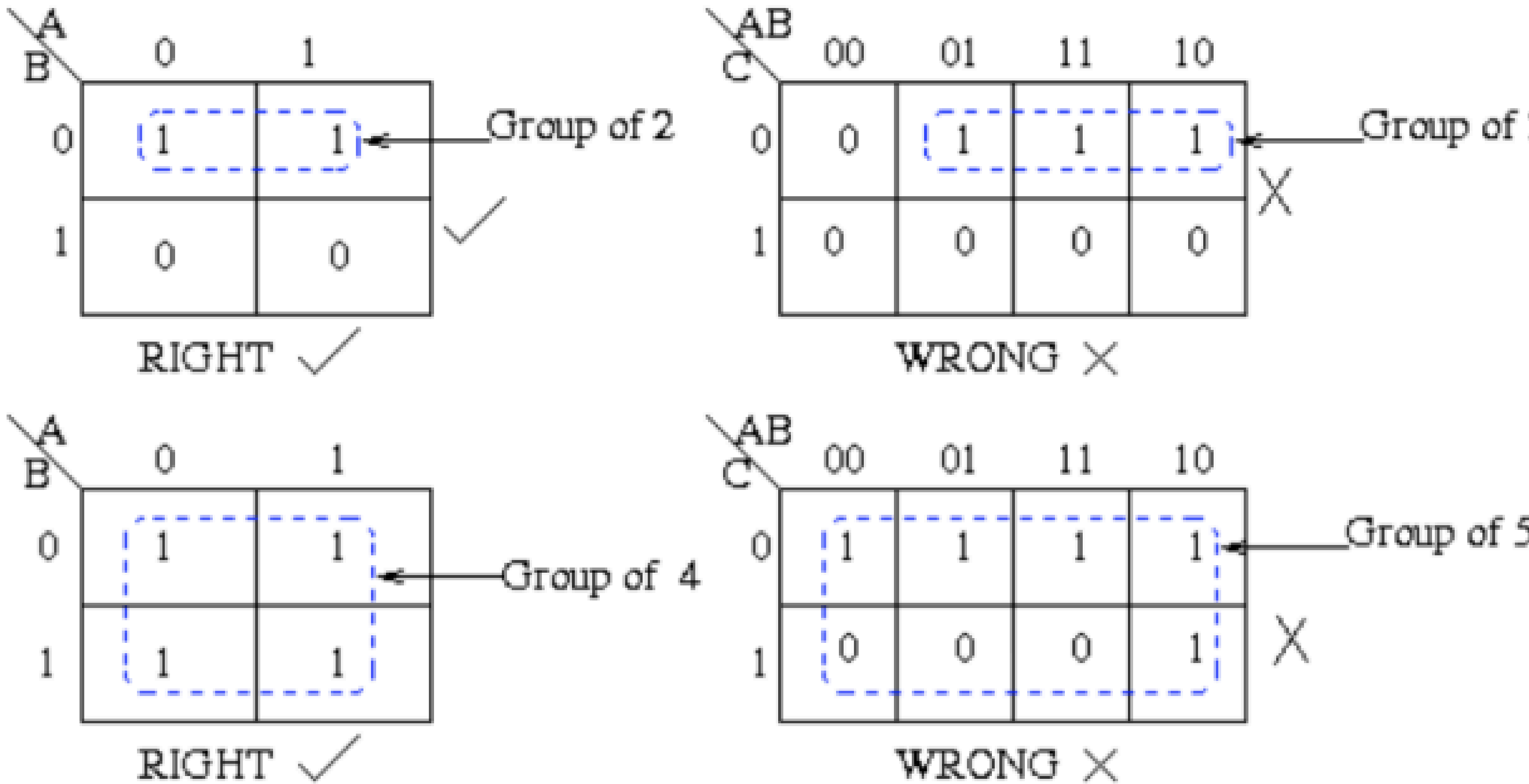
- **Groups may be horizontal or vertical, but not diagonal.**





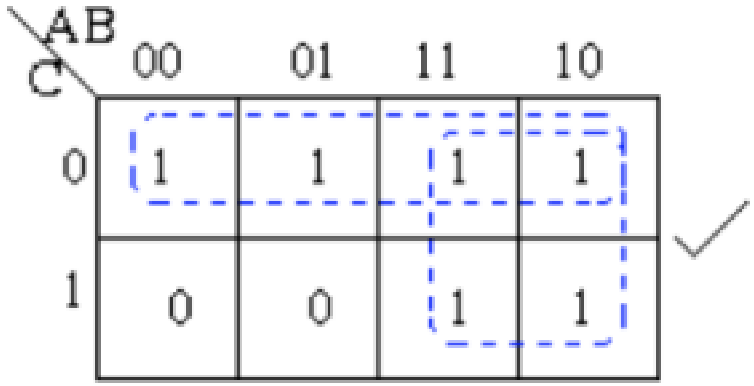
Rules

- **Groups must contain 1, 2, 4, 8, or in general 2^n cells.**
  That is if n = 1, a group will contain two 1's since $2^1 = 2$.
  If n = 2, a group will contain four 1's since $2^2 = 4$.

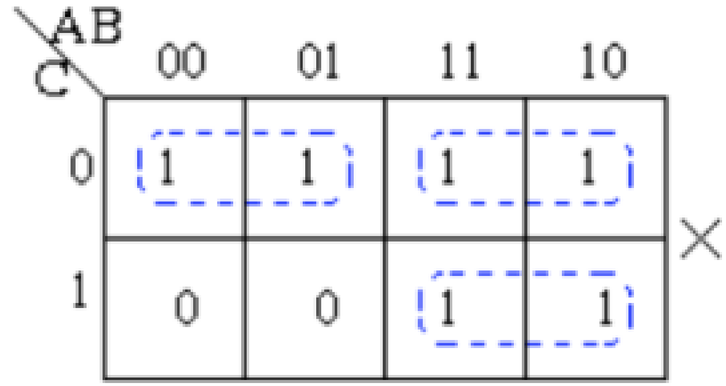


Rules

- **Each group should be as large as possible.**



RIGHT ✓

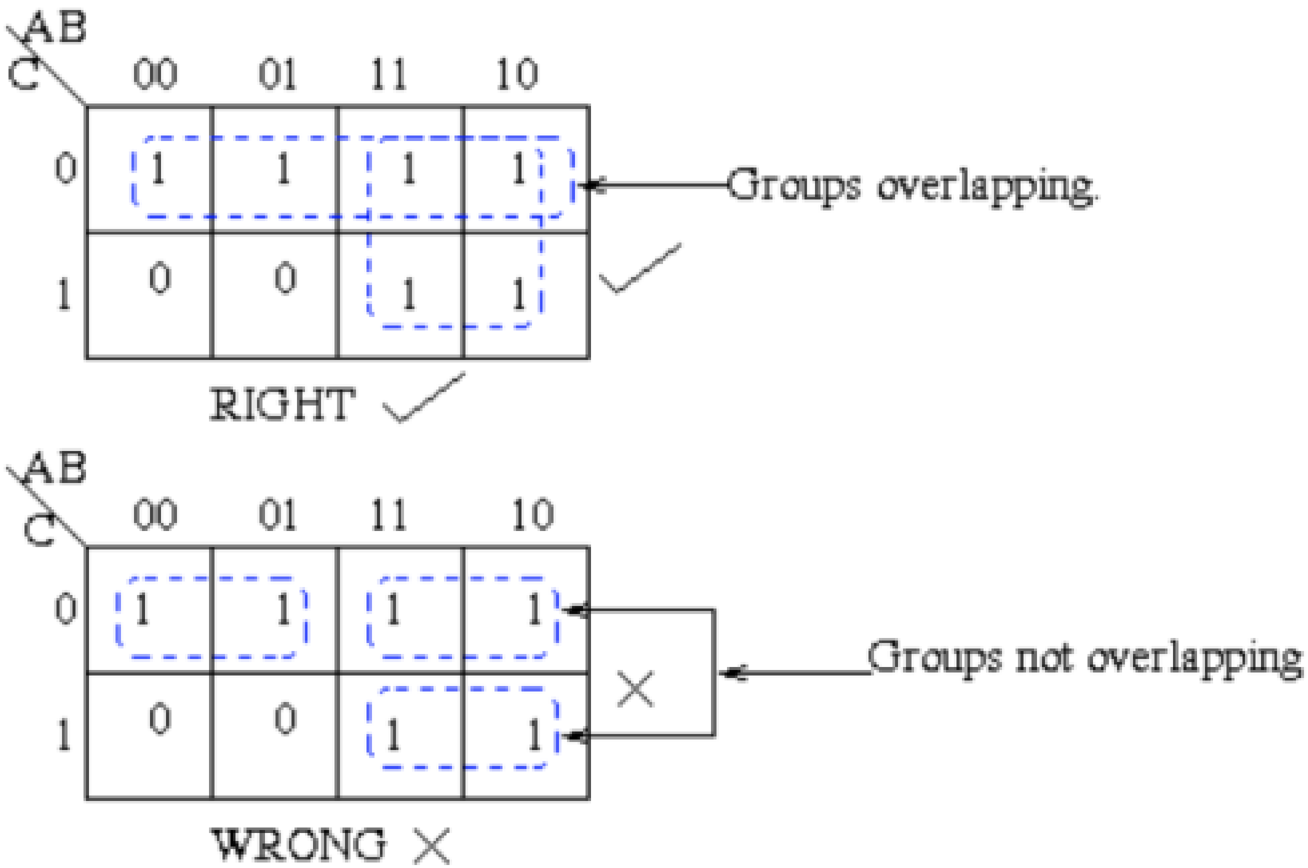


WRONG ×

(Note that no Boolean laws broken, but not sufficiently minimal)

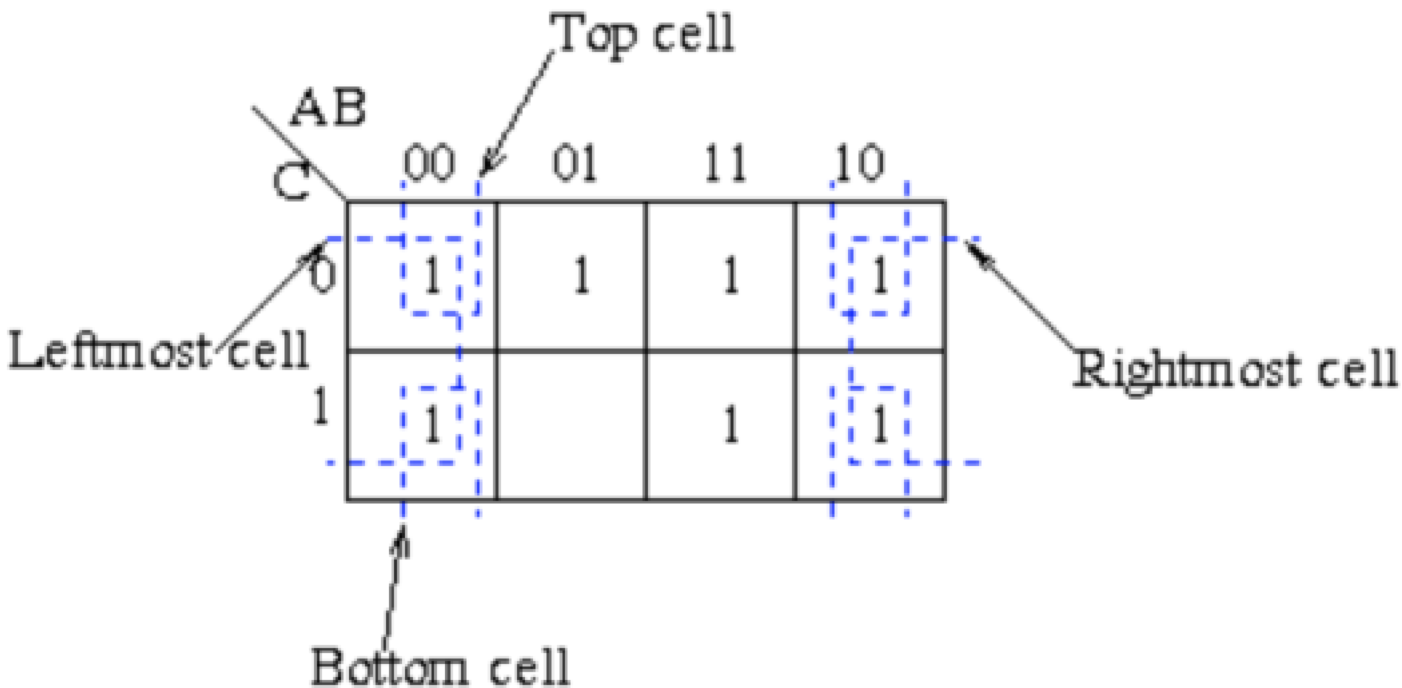
Rules

- **Groups may overlap.**



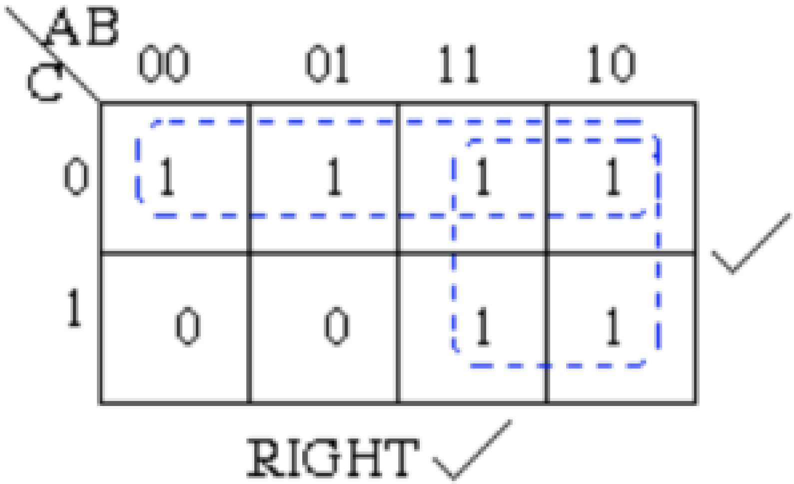
Rules

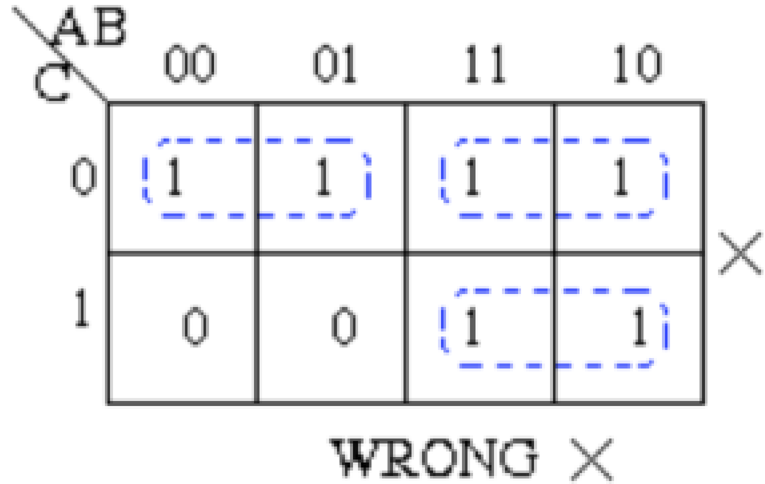
▪ Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



Rules

- There should be as few groups as possible, as long as this does not contradict any of the previous rules.





Summary of the Rules:

1. No zeros allowed.

2. No diagonals.

3. Only power of 2 number of cells in each group.

4. Groups should be as large as possible. 5. Every

one must be in at least one group. 6. Overlapping

allowed.

7. Wrap around allowed.

8. Fewest number of groups possible.

Reduce Boolean Expression

- Consider the function:

�� ��, ��, �� = ��′��′�� +

��′����� + ����′�� + ������ ▪ Its K-Map is

given below.

- What is the largest group of 1’s that is a power of 2?

AB

C 00 10

01 11

0	2	6	4
1 1	3 1	7 1	5 1

0

1

Reduce Boolean Expression

- This grouping tells us that changes in the variables *A* and *B* have no influence upon the value of the function: They are irrelevant.
- This means that the function,

�� ��, ��, �� = ��′��′�� + ��′���� + ����′�� + ������

reduces to �� ��, ��, �� = ��

AB

C 00 10

01 11

0	2	6	4
1 1	3 1	7 1	5 1

0

1

Examples

�� = ������′�� + ����′��′��′ +

����′����′ + ��′��′����′ +
��′��′��′��′ AB

CD 00 10

01 11

2 1	6	14	10 1

00 01 11 10

0 1	4
1	5
3	7

�� =
�������′�
�+ ��′��′

Examples

�� = ����′�� +
��′��′��

AB

C 00 10 01 11

+ ��′�����′�� + ��′������

AB

CD 00 10 01 11

�� = ������′�� + ��������

1		

1

�� = ��′��

00 01 11 10

0

1	1
3	1
2	

�� = ����

Examples

�� = ��′����′�� + ��′��′��′�� + ��′��′��′��′

AB

CD 00 10

01 11

2	6	14	10

00 01 11 10

0 1	4
1 1	5 1
3	7

�� = ��′��′��′ + ��′��′��

Examples

f(A,B,C,D) = ς�� (0,1,4,5,10,11,14,15)

AB

CD 00 10

01 11

00 01 11 10

1 0	5 0	
3	7	

2	6	14 0	10 0

0 0	4 0	

�� = (�� + ��)(��′ + ��′)

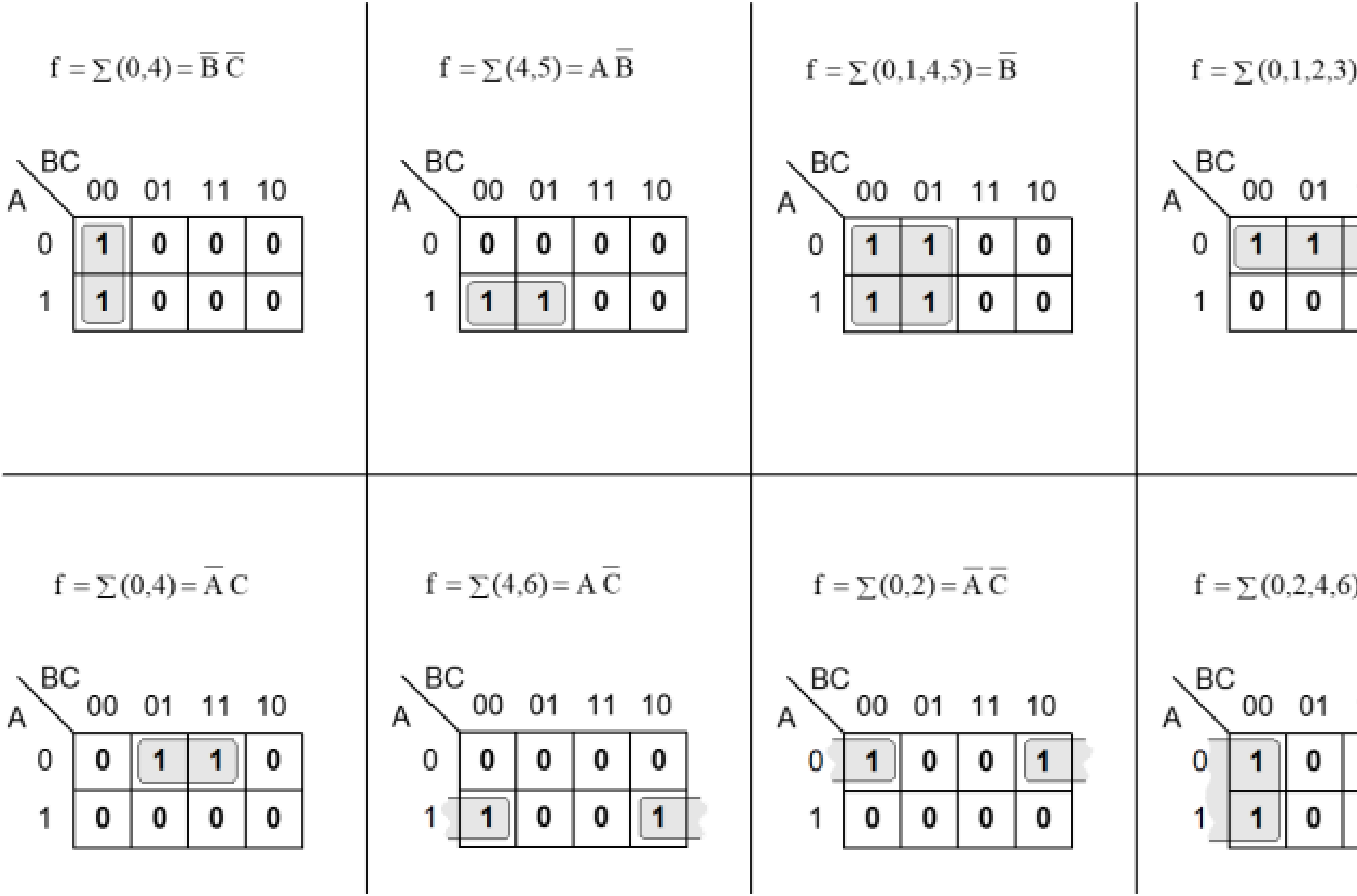
f(A,B,C,D) = ς�� (0,2,4,6,8,10, 12, 14)

$f(A,B,C,D) = \sum(0,5,8,13)$

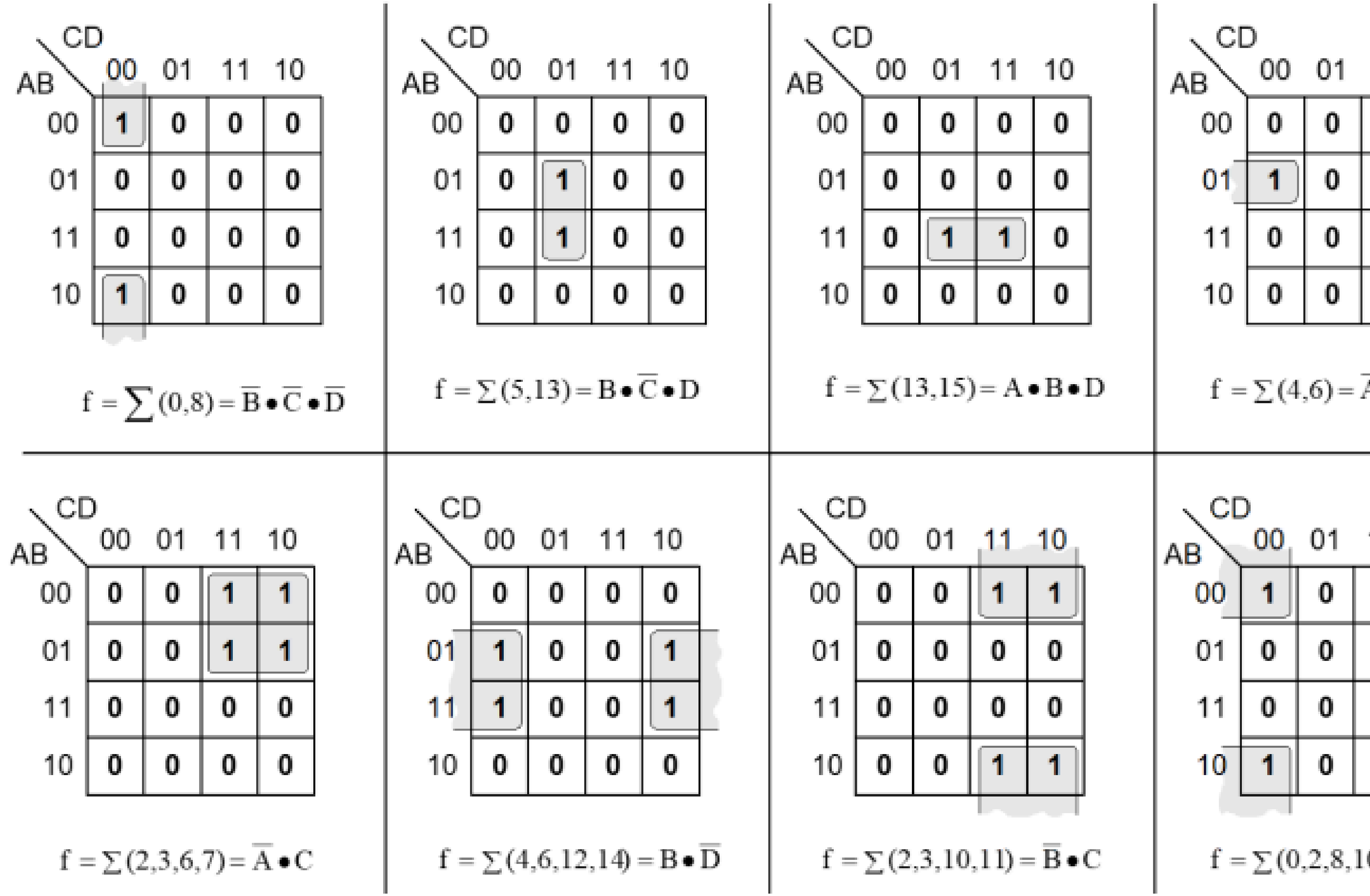
Don’t care conditions

- Suppose we are given a problem of implementing a circuit to generate a logical 1 when a 2, 7, or 15 appears on a four-variable input.
- A logical 0 should be generated when 0, 1, 4, 5, 6, 9, 10, 13 or 14 appears.
- The input conditions for the numbers 3, 8, 11 and 12 never occur in the system. This means we don’t care whether inputs generate logical 1 or logical 0.
- Don’t care combinations are denoted by ‘x’ in K-Map which can be used for the making groups.
- The above example can be represented as

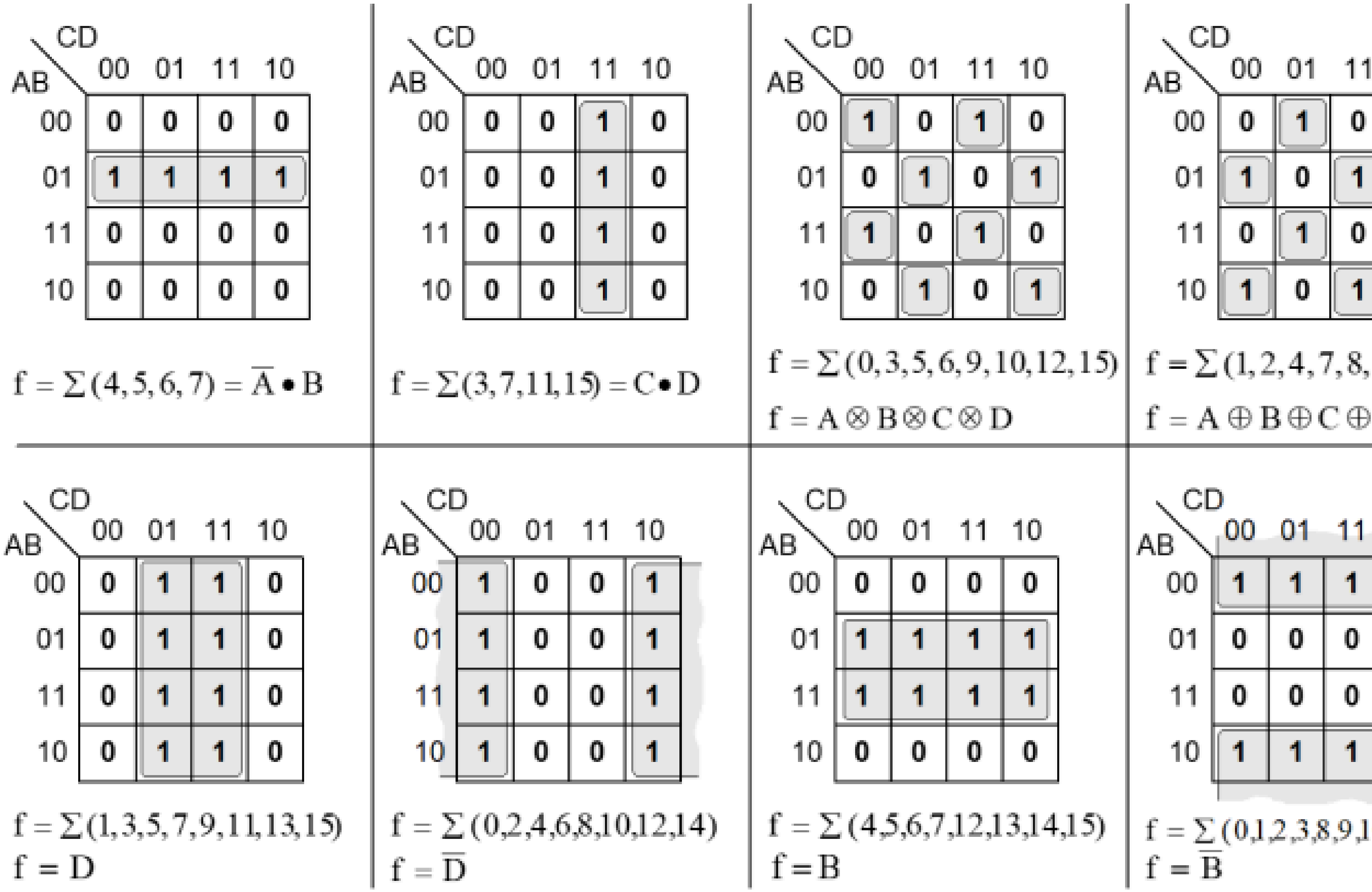
3-variable k-map



4-variable k-map



4-variable k-map



f = Σ(4,5,6,7) = A̅ • B
f = Σ(3,7,11,15) = C • D
f = Σ(0,3,5,6,9,10,12,15)
f = A ⊗ B ⊗ C ⊗ D
f = Σ(1,2,4,7,8,
f = A ⊕ B ⊕ C ⊕
f = Σ(1,3,5,7,9,11,13,15)
f = D
f = Σ(0,2,4,6,8,10,12,14)
f = D̅
f = Σ(4,5,6,7,12,13,14,15)
f = B
f = B̅

Examples

f(A,B,C,D) = σ�� (2,7,15) +

d(3,8,11,12)AB

CD 00 10

01 11

3 x	7 1	15 1	11 x
2 1	6	14	10

00 01 11 10

0	4
1	5

�� =

����+

��′��′��

Examples

F(W,X,Y,Z) = σ�� (1,3,7,11,15) +

d(0,2,5)WX

YZ 00 10

01 11

3 1	7 1	15 1	11 1
2 x	6	14	10

00 01 11 10

0 x	4
1 1	5 x

�� =

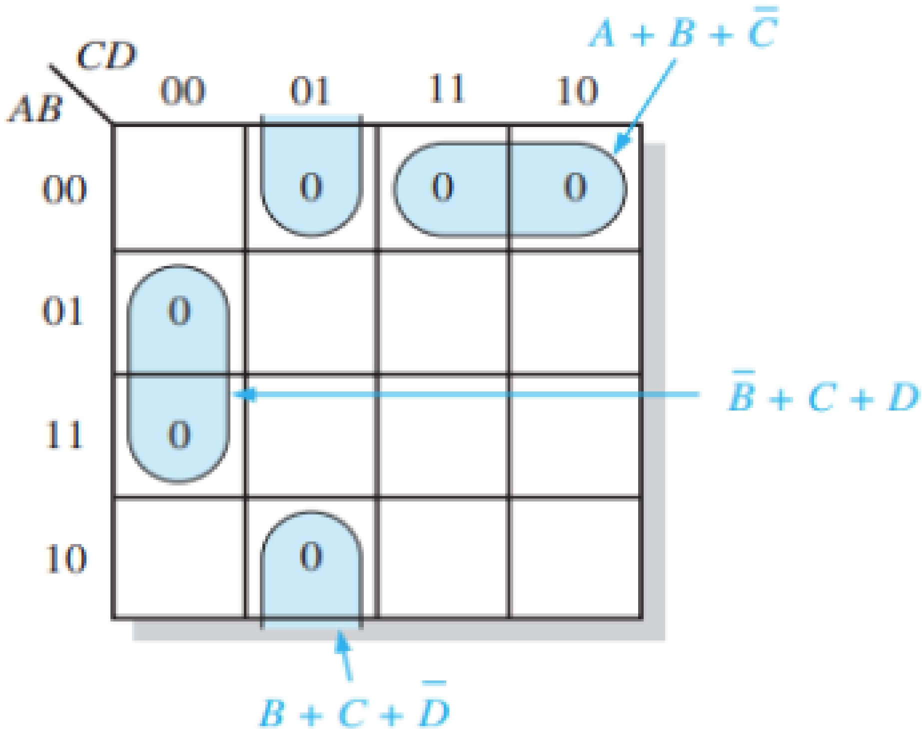
��′��′+�

���

Converting Between POS and SOP Using the K-Map

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a sta
expression, and a minimum SOP expression.

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



(a) Minimum POS: $(A + B + C)(\overline{B} + C + D)(B + C + \overline{D})$

NAND and NOR

Implementation

- Recall the De-Morgan's Law:

NAND and NOR as Universal Gates

NAND and NOR as Universal Gates

NAND Implementation

- Minimize the Function F = Σ(1,2,3,4,5,7) using k-map and implement with NAND.

NOR Implementation

- Converting to NOR Implementations

Self Check Exercise