Boolean Function Minimization and Simple Logic Circuit Design

4.1 Introduction of minimization

4.2 Minimization using Boolean algebra

4.3 Karnaugh map

4.4 Minimization using Karnaugh map

4.5 Logic functions with don’t care conditions

4.6 Logic circuit design

4.1 Introduction of minimization

- Boolean algebra is a tool for simplifying/minimizing logic circuits

Example: $f = x'yz + x'yz' + xz$

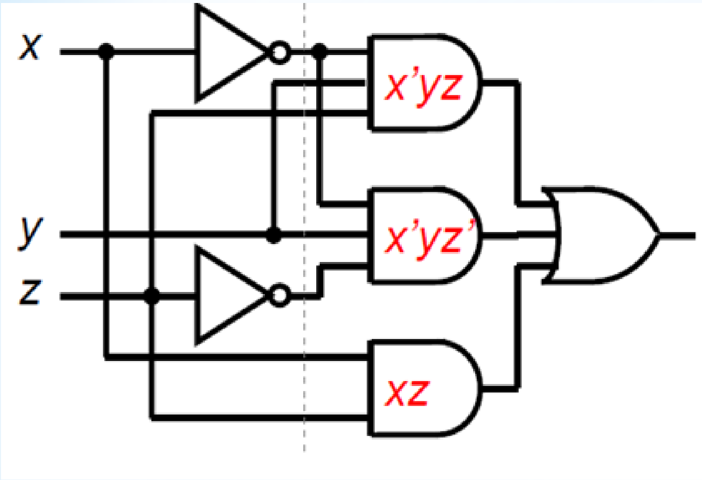
can be simplified as

$$f = x'y + xz$$

Result:

Reduced f from sum of 3 products to sum of 2 product terms.

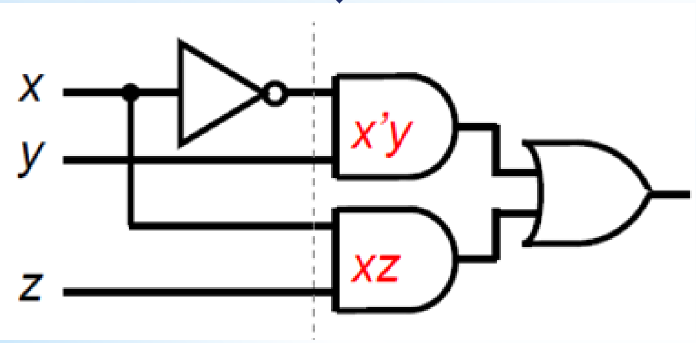
Logic circuit with minimization



$f = x'yz + x'yz' + xz$

6 gates

Equivalent to



$f = x'y + xz$

4 gates

Advantages of minimization

- **Obtain a simple (or simplest) logic circuit**
- **Reduce the cost of production**
  - Less logic gate ICs required
  - PCB (Printed Circuit Board)
    - ➔ **size reduced**
- **Reduce the power consumption**
  - ➔ **longer battery usage time**

4.2 Minimization using Boolean Algebra

Example:

Given 4 equivalent Boolean functions f_1 to f_4 expressed in SOP form already (to be proved in later session).

$f_1(a,b,c) = a'bc' + a'bc + ab'c' + ab'c + abc$ (5 product terms, 15 literals)

$f_2(a,b,c) = a'b + ab' + abc$ (3 product terms, 7 literals)
$f_3(a,b,c) = a'b + ab' + ac$ (3 product terms, 6 literals)
$f_4(a,b,c) = a'b + ab' + bc$ (3 product terms, 6 literals)
Both f_3 & f_4 are the minima

How can you simplify f_1 to f_3?

Simplification

$$f_1(a,b,c) = a'bc' + a'bc + ab'c' + ab'c + abc$$
$$= (a'bc' + a'bc) + (ab'c' + ab'c) + abc$$
$$= a'b + ab' + abc = f_2$$
$$= a'b + a(b' + bc)$$
$$= a'b + a(b' + c)$$
$$= a'b + ab' + ac$$
$$= f_3$$

How to obtain f_4 ?

$$f_4(a,b,c) = a'b + ab' + bc$$

Example

$$f(A,B,C,D) = \boxed{\overline{AB + AC}} + \overline{A}\,\overline{B}C$$

$$= \boxed{(\overline{AB})(\overline{AC})} + \overline{A}\,\overline{B}C$$

DeMorgan

$$= \boxed{(\overline{A} + \overline{B})(\overline{A} + \overline{C})} + \overline{A}\,\overline{B}C$$

$$= \overline{A}\,\overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}C$$

$$= \overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

$$= \overline{A} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

$$= \overline{A} + \overline{B}\,\overline{C}$$

Example

$$f(A,B,C,D) = AB + B\overline{C} + CD + B\overline{D}$$
$$= AB + CD + B(\overline{C} + \overline{D})$$
$$= AB + CD + B\overline{CD}$$
$$= AB + (CD + B)(CD + \overline{CD})$$
$$= AB + CD + B$$
$$= B(A+1) + CD$$
$$= B + CD$$

A very useful rule for simplifying Boolean function

Basic: $$a + \overline{a}b = (a + \overline{a})(a + b) = a + b$$

We can have the following general form:

$$a + \overline{a}[\cdots] = a + [\cdots]$$

Examples: $$x + \overline{x}(w + yz) = x + (w + yz)$$

$$a + \overline{a}(bd + b\overline{cd} + \overline{c}d) = a + bd + b\overline{cd} + \overline{c}d$$

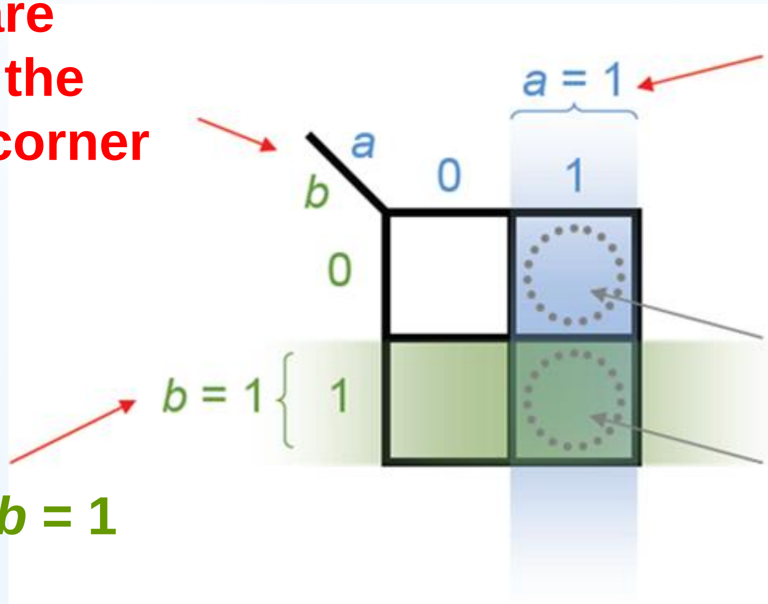
4.3 Karnaugh Map

- In 1953, Maurice Karnaugh introduced a map method known as **Karnaugh map (K-map)**
  - A straightforward procedure for minimizing Boolean functions in a table form
  - Graphical representation of a truth table
  - Minterm is used in the cell of the K-map
  - It is *n*-variable function (defined by 2^n):
    - Two-variable K-map has 4 cells
    - Three-variable K-map has 8 cells
    - Four-variable K-map has 16 cells

Two-variable K-map

Variables are labeled on the upper left corner of the map

This column represents *a* = 1



This cell means (*a* = 1) AND (*b* = 0)

This row represents *b* = 1

This cell means (*a* = 1) AND (*b* = 1)

b \ a	0	1
0	*a'b'*	*ab'*
1	*a'b*	*ab*

b \ a	0	1
0	m_0	m_2
1	m_1	m_3

Minterm representations

Plotting functions in K-map

$f(a,b) = \sum m(0,3)$ Canonical form (contain Minterm)

b \ a	0	1
0		
1		

or

b \ a	0	1
0	1	
1		1

Put a 0 or leave blank for those minterms not included in the function

Put a 1 in the corresponding cells

$f(a,b) = a'b + ab'$ Function must be formed by Minterm

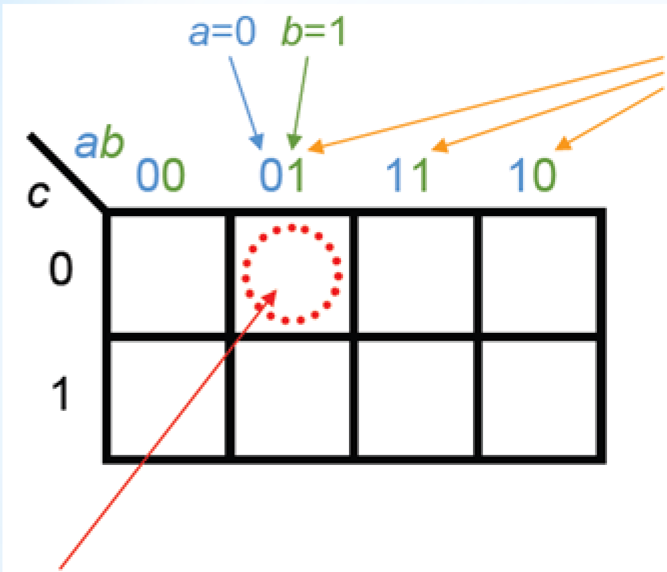
b \ a	0	1
0	1	1
1	0	0

or

b \ a	0	1
0	1	1
1		

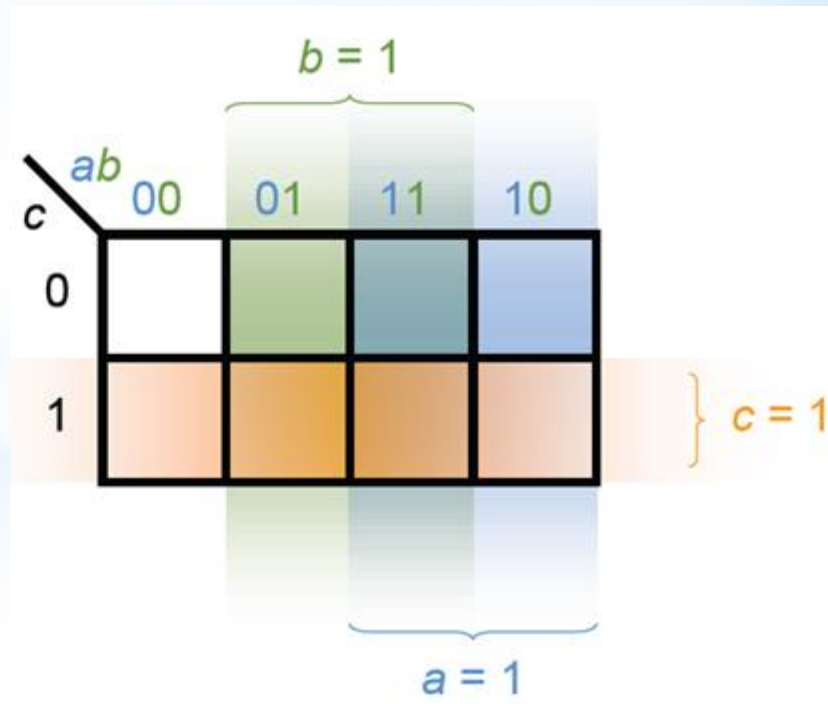
Functions represented graphically with corresponding minterm cells labeled to value 1

Three-variable K-map



Note: the columns are not in numerical order, but <u>Gray code</u> order (why)?

This cell means:
(*a* = 0) AND (*b* = 1) AND (*c* = 0)



Minterm representations

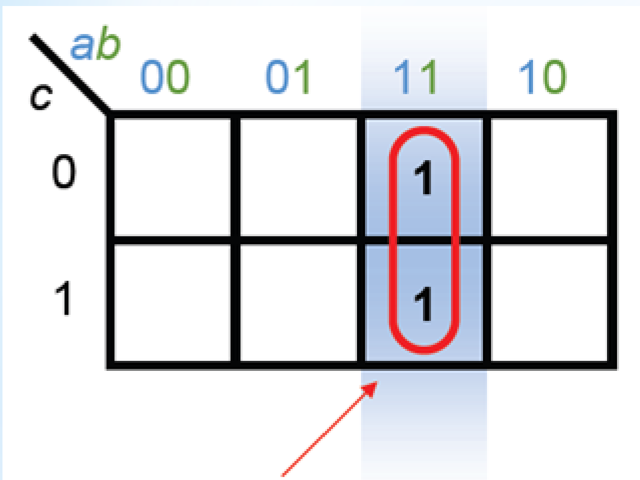
c \ ab	00	01	11	10
0	a'b'c'	a'bc'	abc'	ab'c'
1	a'b'c	a'bc	abc	ab'c

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

Simplification of Production Terms

Example: Simplify $f(a,b,c) = \sum m(6,7)$

Only one-variable difference



This group contains both 0 and 1 for *c* (i.e. no longer depends on *c*, depends on *a* and *b* only

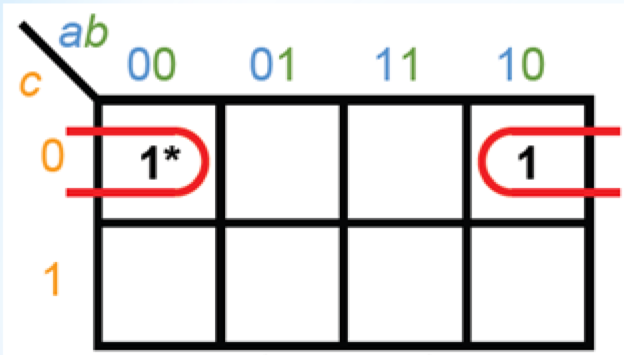
Using Boolean Algebra:

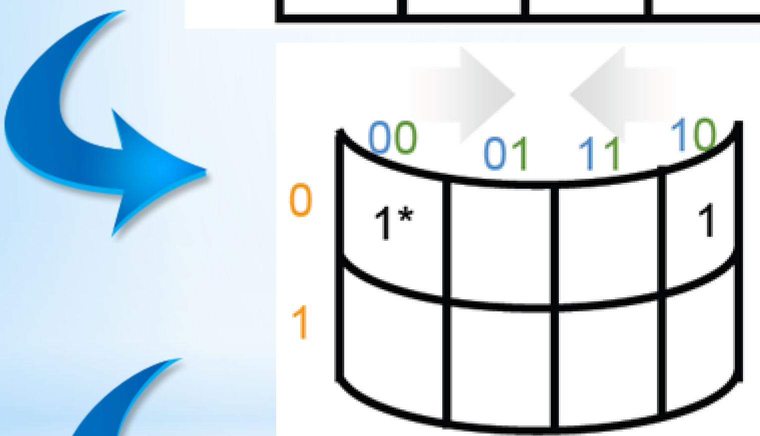
$f(a,b,c) = abc' + abc$

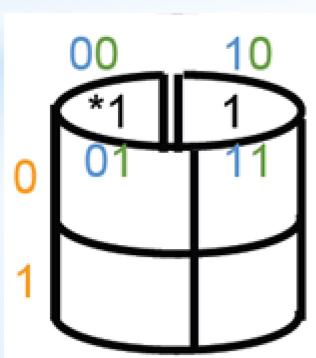
$= ab$ (*adjacency*)

Rule: whenever we group two adjacent cells, they can form a product term with one variable less

Wrap-around Adjacency

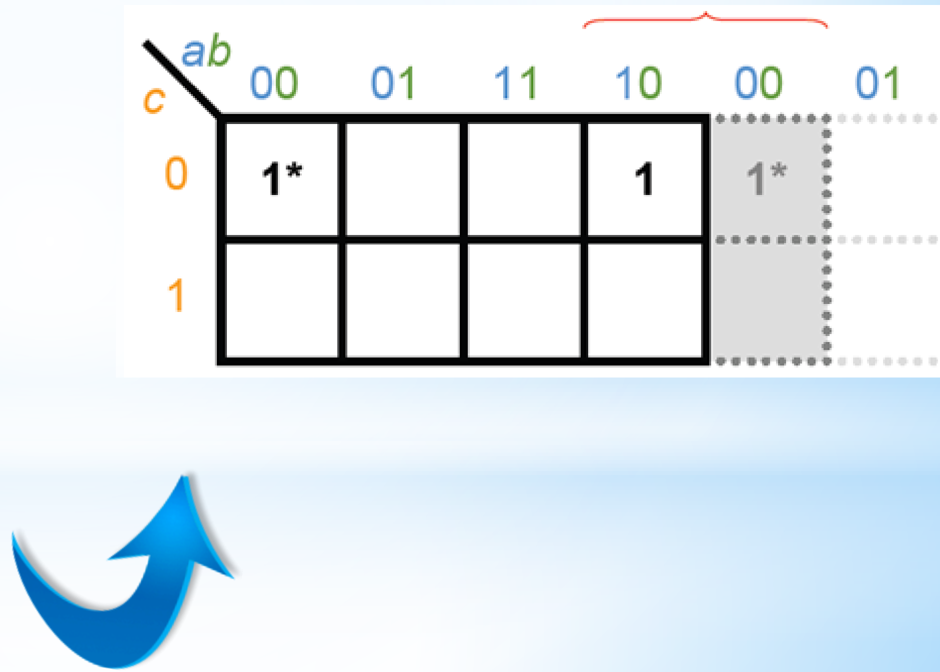






Form a cylinder!

Adjacent cells (1-bit change only)



More examples

c \ ab	00	01	11	10
0			1	1
1				

f(*a*,*b*,*c*) = *abc*' + *ab*'*c*'
= *ac*' (*adjacency*)

We can even group adjacent 1's across the edges:

c \ ab	00	01	11	10
0	1			1
1				

Also one-variable difference!

f(*a*,*b*,*c*) = *a*'*b*'*c*' + *ab*'*c*'
= *b*'*c*' (*adjacency*)

Format of three-variable

Label rows with first variable, Columns with the others

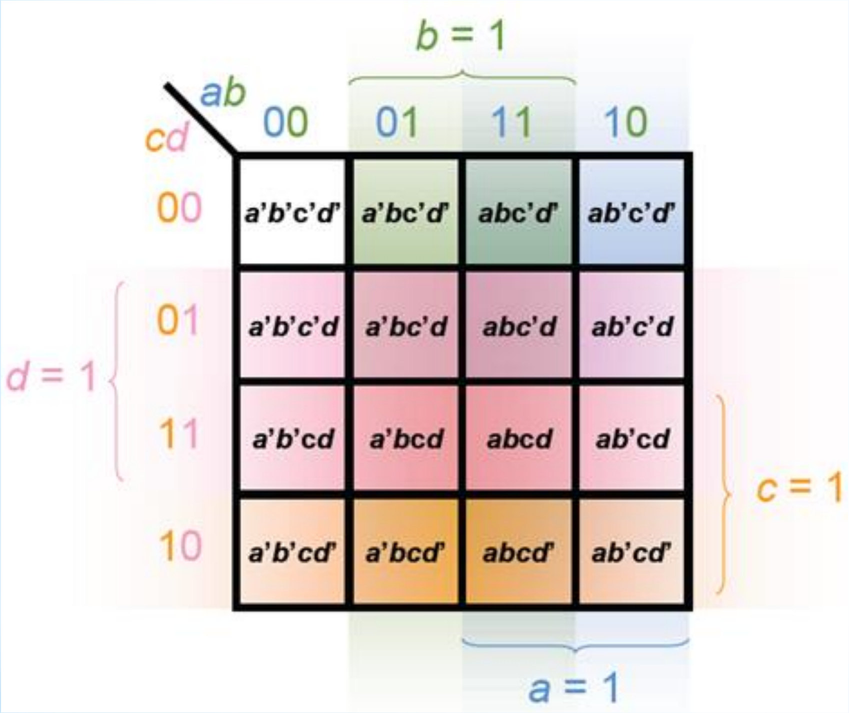
$a \backslash bc$	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

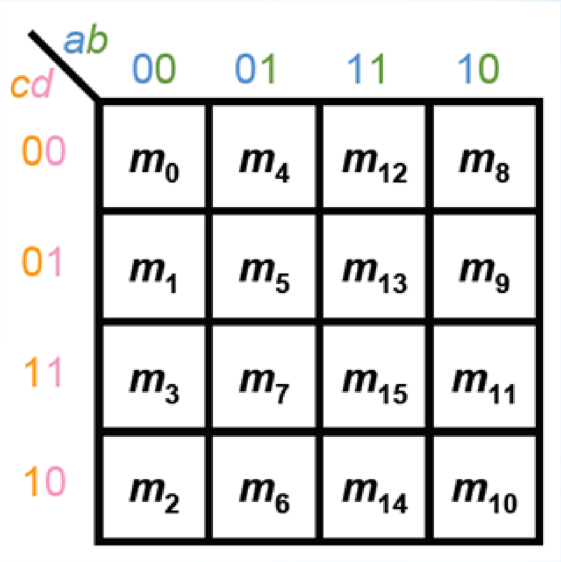
Vertical orientation of three-variable K-map

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

Although there are different ways drawing the K-map, we use the same method to group the adjacent 1's!

Four-variable K-map





Not: Only 1 bit difference between adjacent cells

Wrap-around Adjanceny for 4 Variables

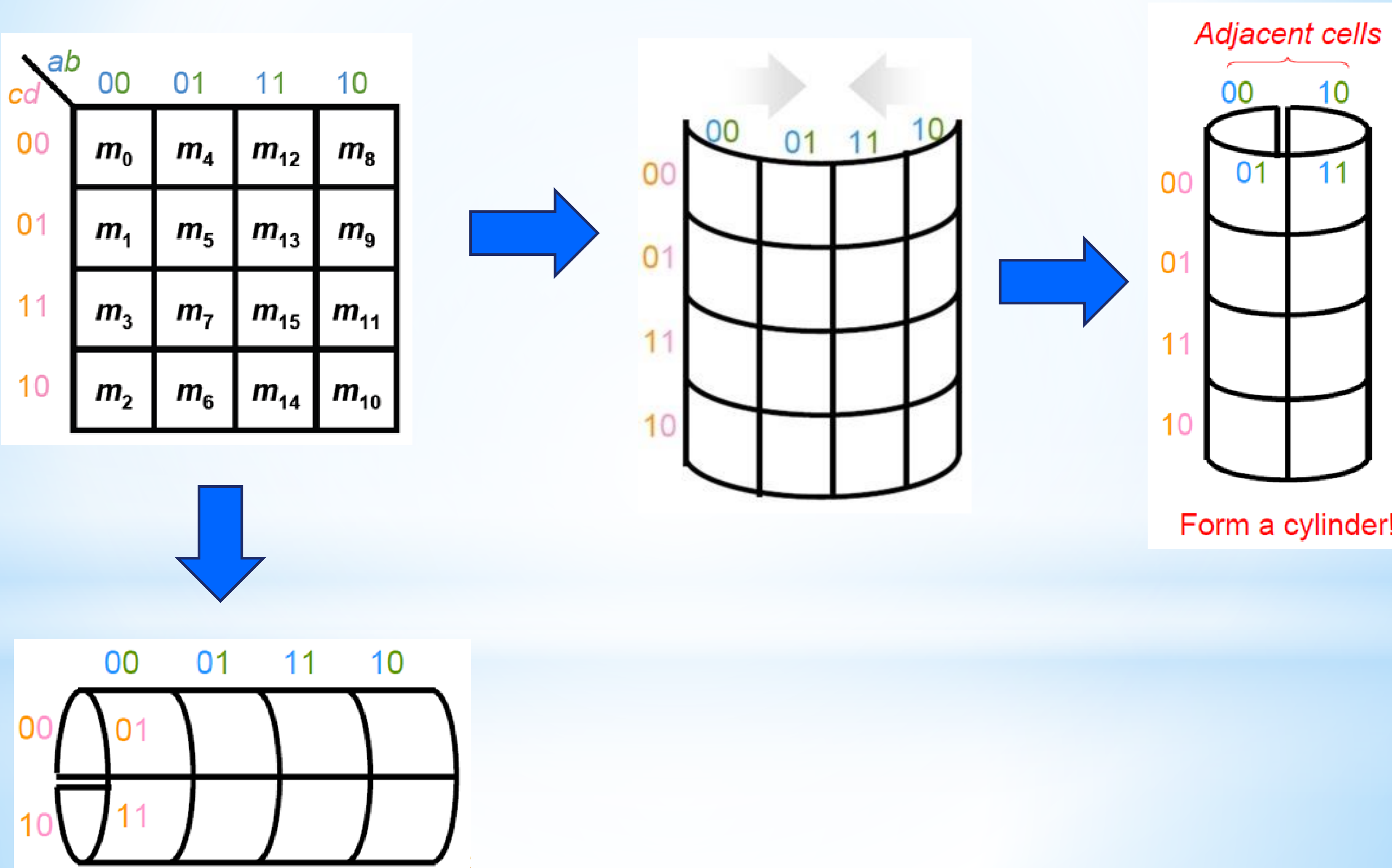
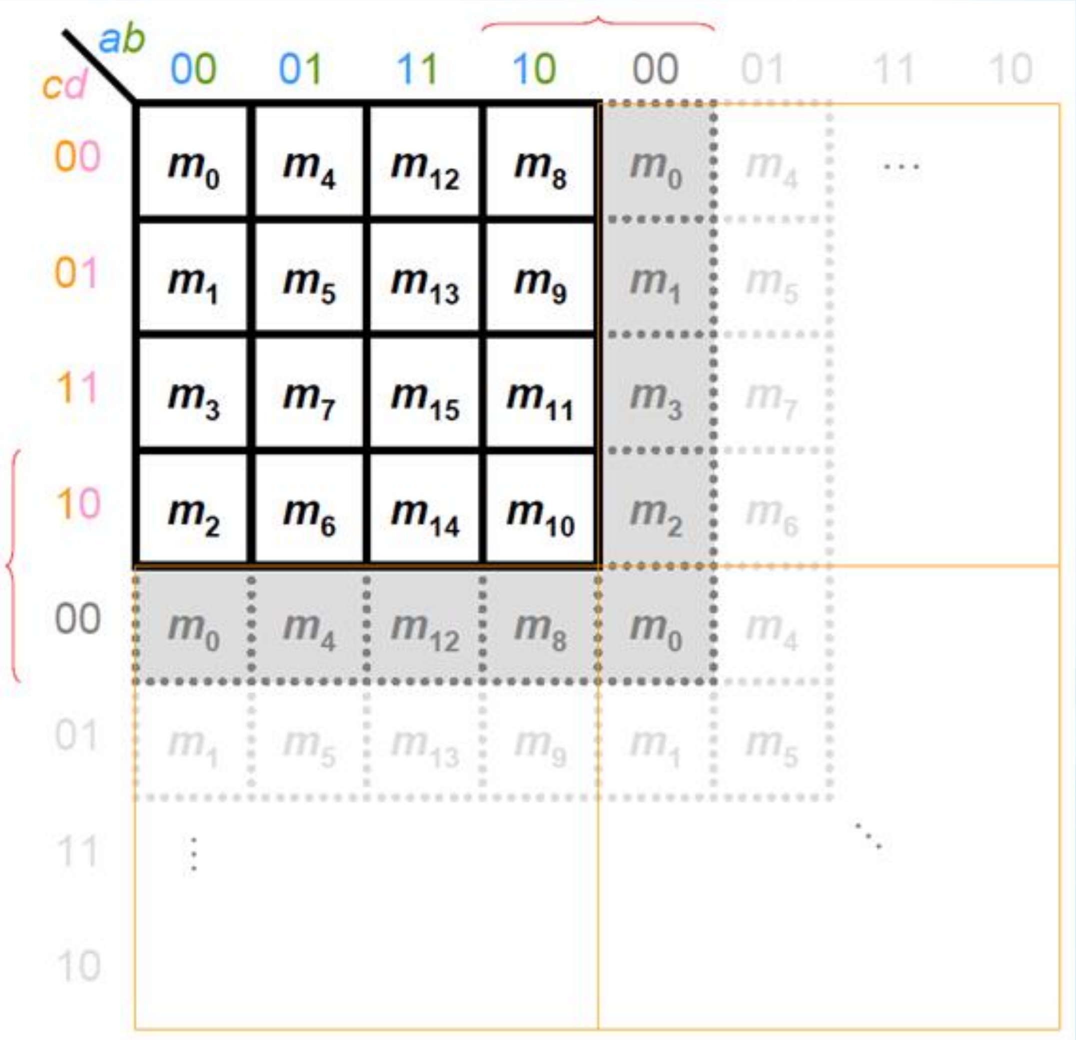


Image the map as ….

1-bit change only for every adjacent cells!



1-bit change only for every adjacent cells!

Examples of 4-variable K-map

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

f(*a*,*b*,*c*,*d*) = *a'b'd'* + *bc'd*

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

f(*a*,*b*,*c*,*d*) =

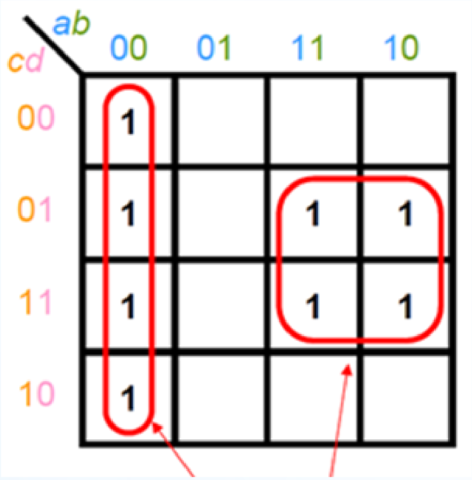
cd \ ab	00	01	11	10
00				
01				
11	1	1	1	1
10				

f(*a*,*b*,*c*,*d*) = *a'cd* + *acd* = *cd*

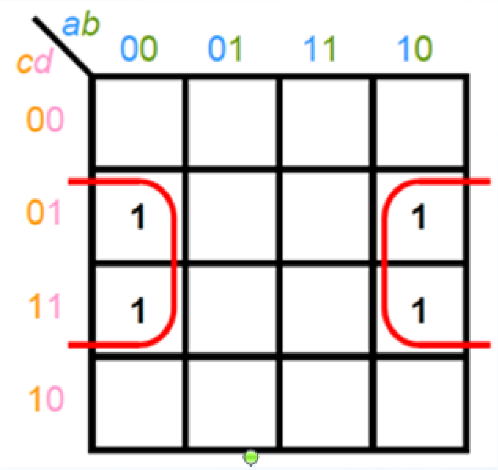
cd \ ab	00	01	11	10
00				
01				
11	1	1	1	1
10				

f(*a*,*b*,*c*,*d*) = *cd*

Examples of 4-variable K-map

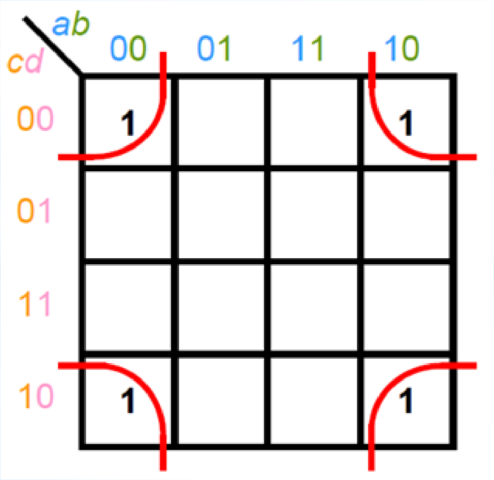


f(*a*,*b*,*c*,*d*) = *a*'*b*' + *ad*



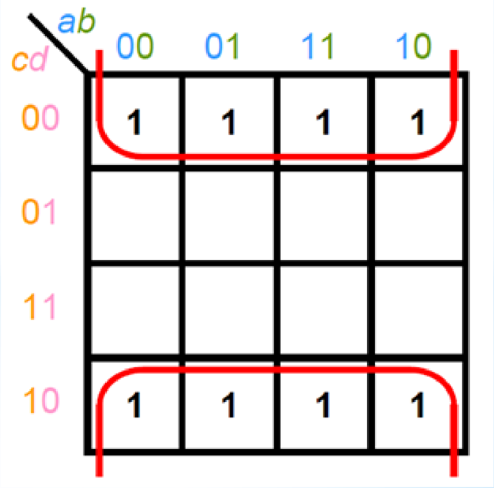
f(*a*,*b*,*c*,*d*) =

Across 4 corners:



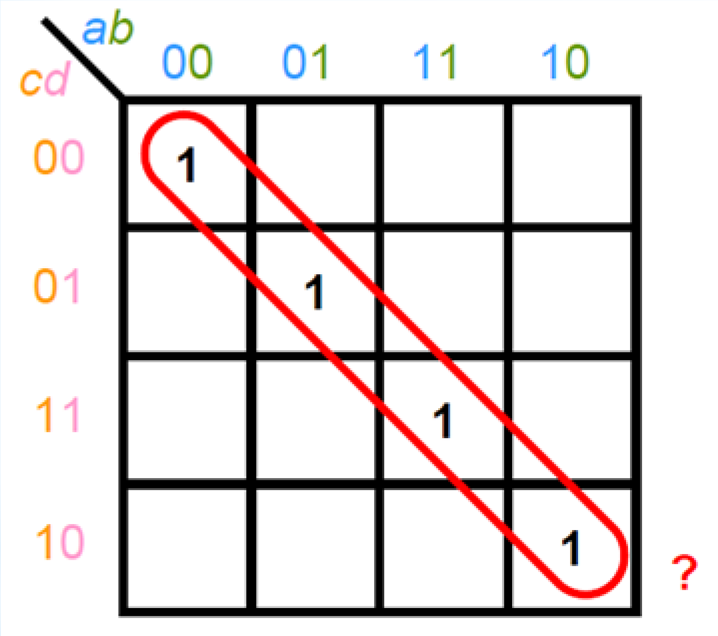
f(*a*,*b*,*c*,*d*) =

Group of 8 cells:

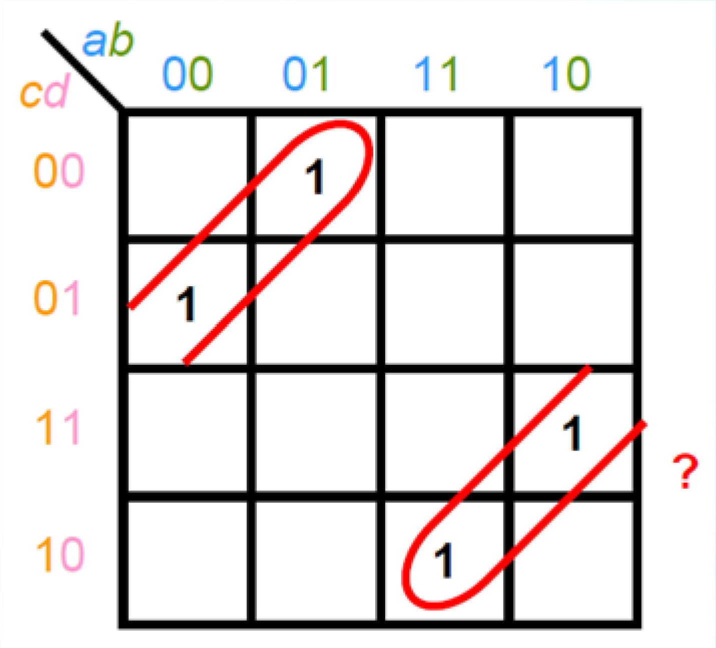


f(*a*,*b*,*c*,*d*) =

Are They Adjacent Cells?

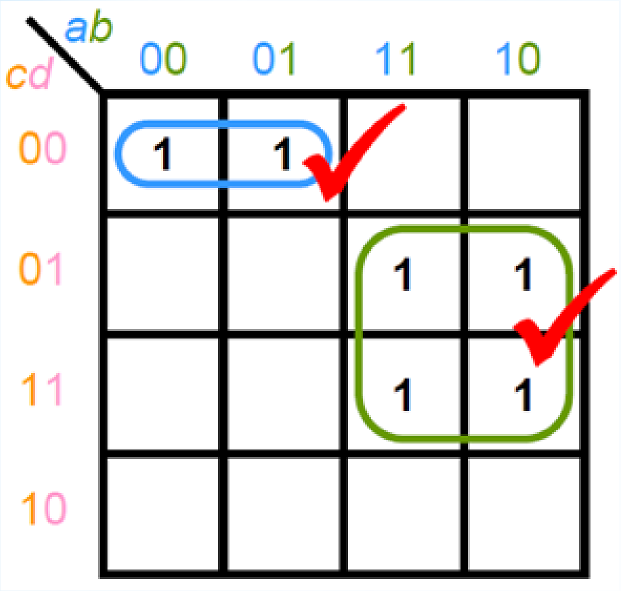


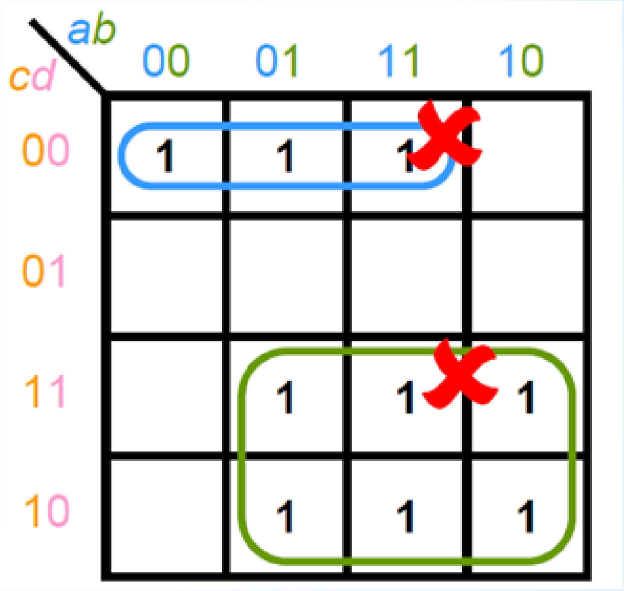
Diagonal X



Magic square X

Summary for K-map method





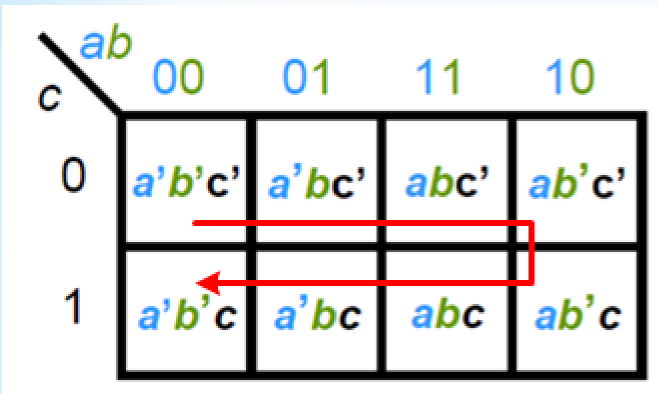
Group size is power of 2 (e.g. 2, 4, 8)

Other group size is illegal!

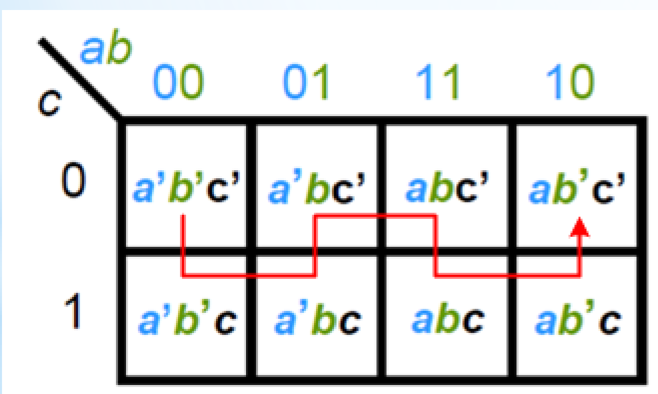
Limitations

- The Boolean functions minimized by K-map are always in SOP or POS form
- Can handle minimization for two-level circuits, but not three or more levels

Gray code generation by K-map



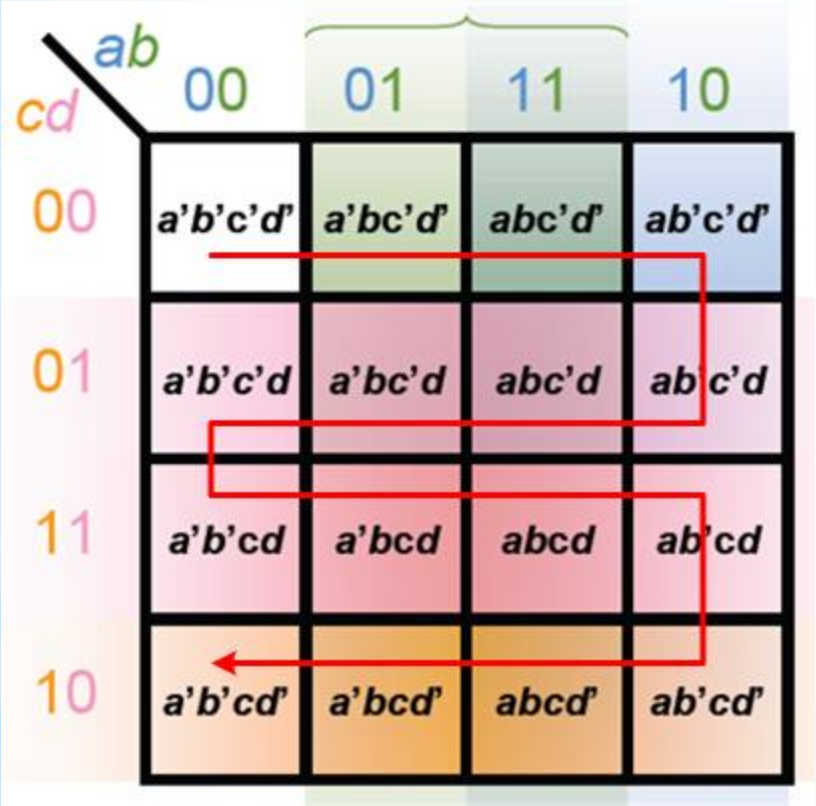
0	000	4	101
1	010	5	111
2	110	6	011
3	100	7	001

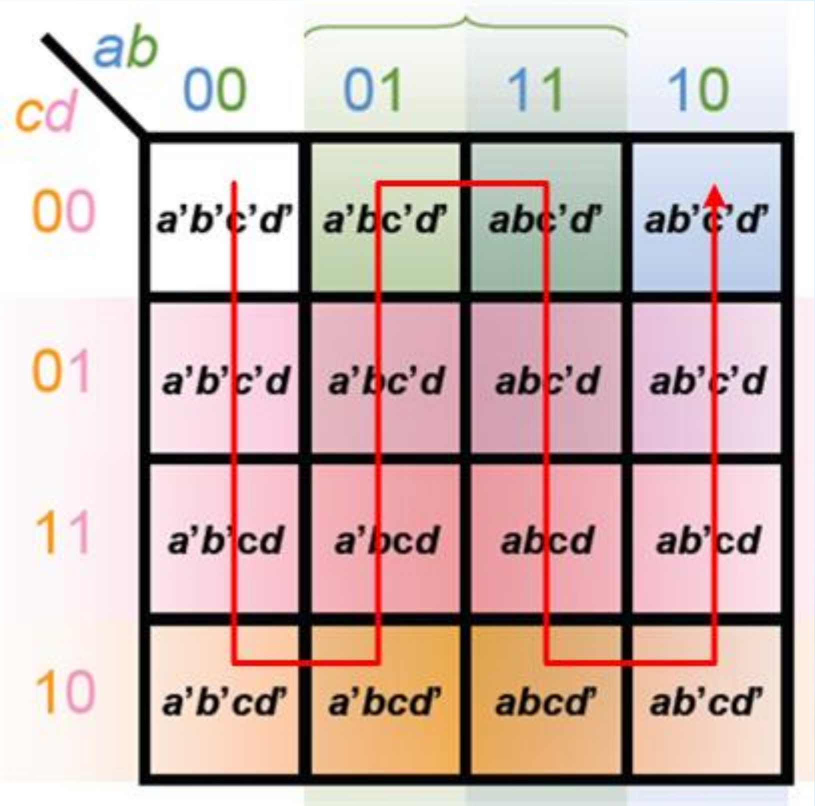


0	000	4	110
1	001	5	111
2	011	6	101
3	010	7	100

Gray code – unit distance code (one bit difference between adjacent numbers) ➔ many combinations.

4-bit Gray code from K-map





4.4 Minimization using Karnaugh map

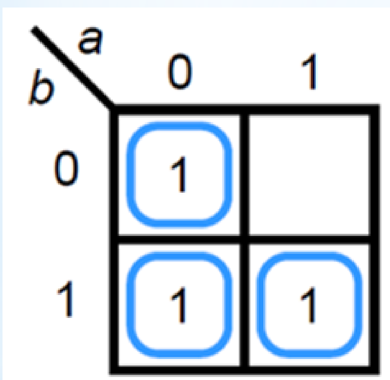
- Group the adjacent cells (the number of cells must be a power of 2)
- Rules
  - (1) To find the fewest group that covers all cells with marked of 1s
  - (2) The groups should be as large as possible
- Goal
  - Reduce the number of products (terms) to minimum
  - Save the cost

Example: Two-variable K-map

Simplify $f(a,b) = \sum m(0,1,3)$

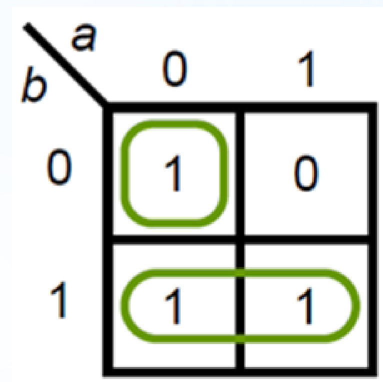
Many ways to group them. Which is best solution?

(i)



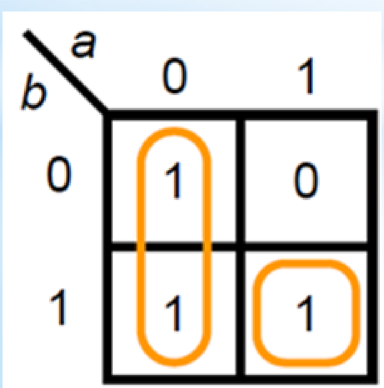
3 groups
$f = a'b' + a'b + ab$

(ii)



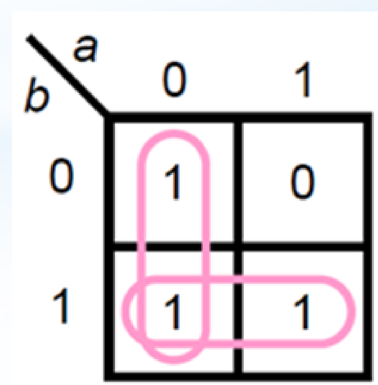
2 groups
$f = a'b' + b$

(iii)



2 groups
$f = a' + ab$

(iv)



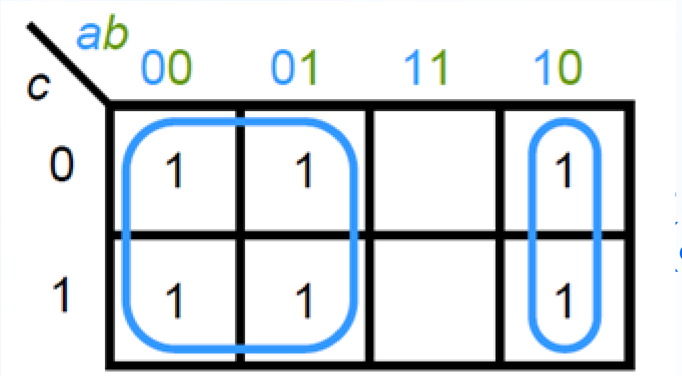
2 groups (the groups can be overlapped)
$f = a' + b$

Example: Three-variable K-map

Simplify $f(a,b,c) = \sum m(0,1,2,3,4,5)$

Which solution is better?

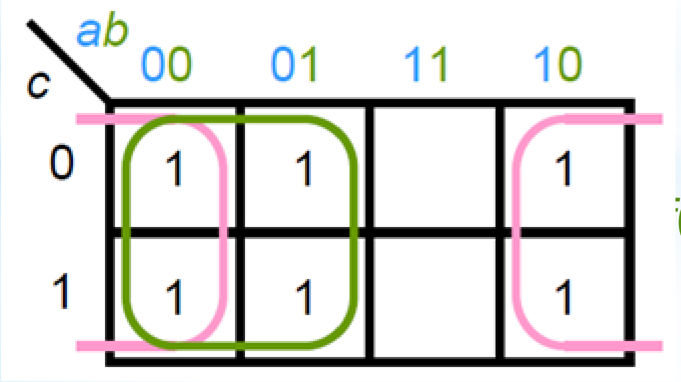
(i)



groups

$(a,b,c) = a' + ab'$

(ii)



groups

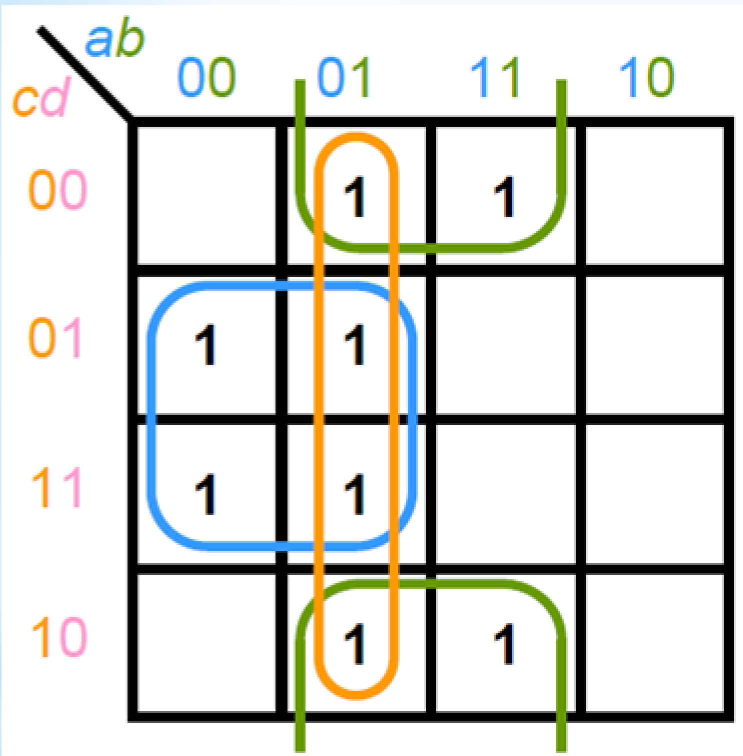
$(a,b,c) = a' + b'$

Example: Four-variable K-map

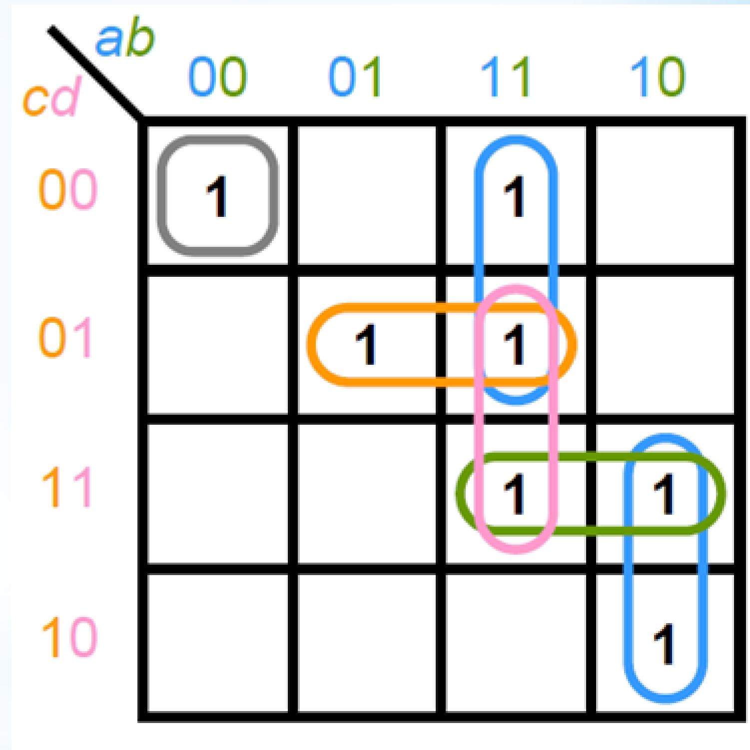
Simplify $f(a,b,c,d) = \sum m(0,1,2,3,4,5,7,8,10,11,15)$

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

Grouping of K-map

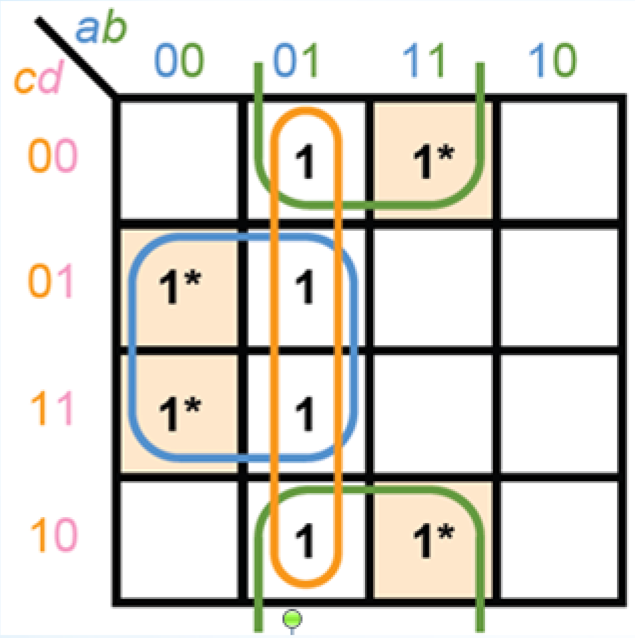


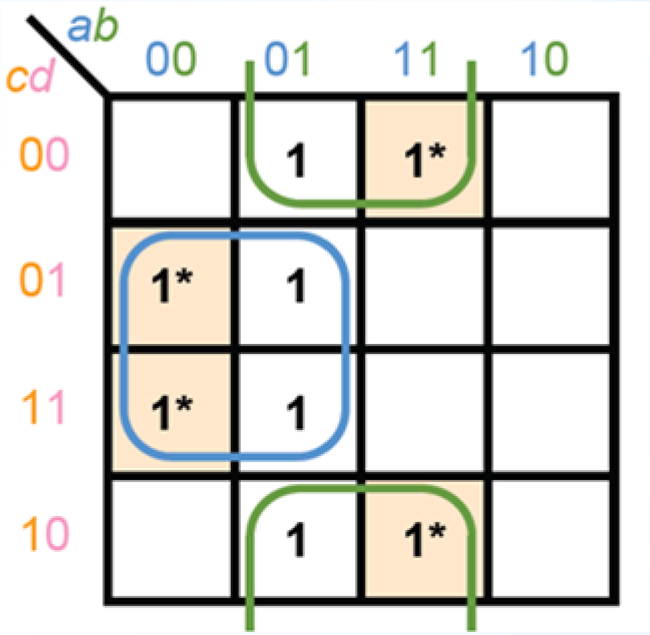
Three groups overlapped!



Too many overlaps!

How to minimize?

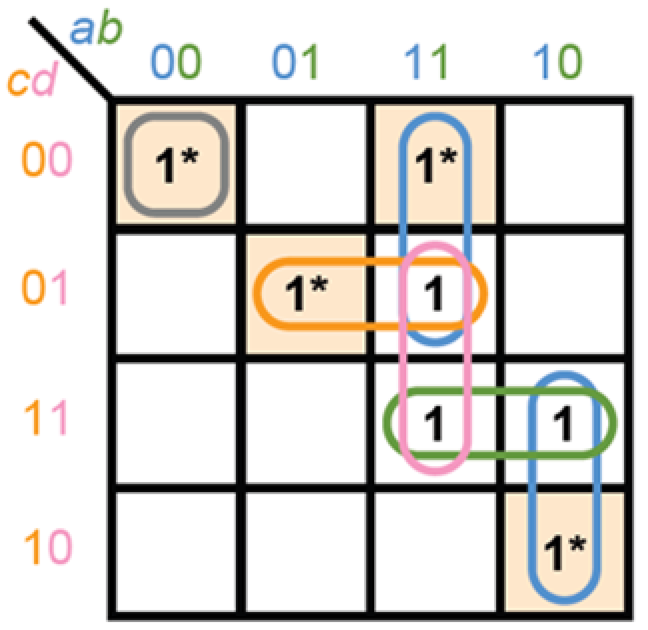


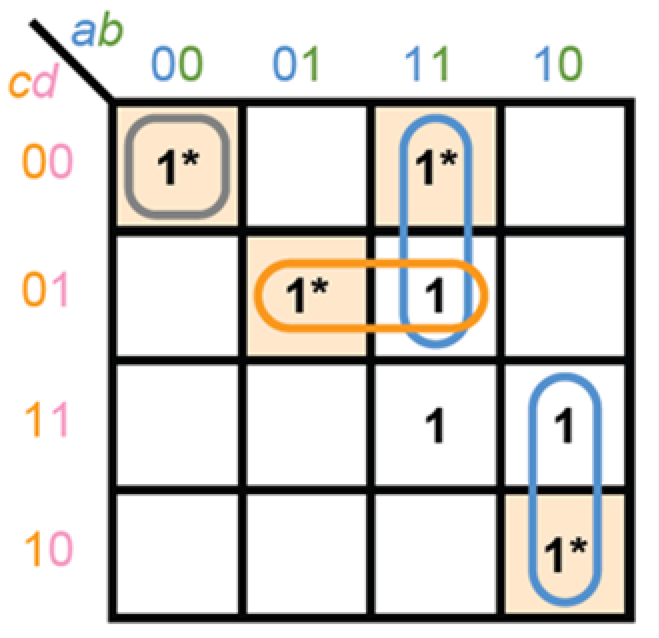


Select the least number of groups to cover all minterms:

$$f(a, b, c, d)' = a'd + bd'$$

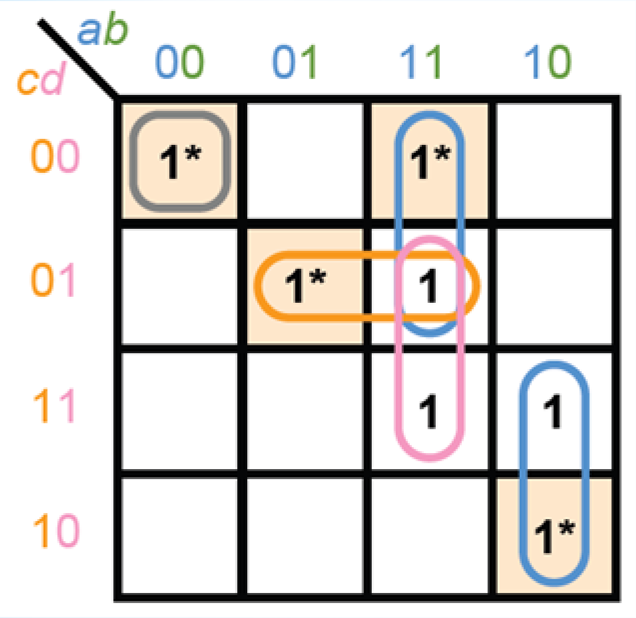
How to minimize?

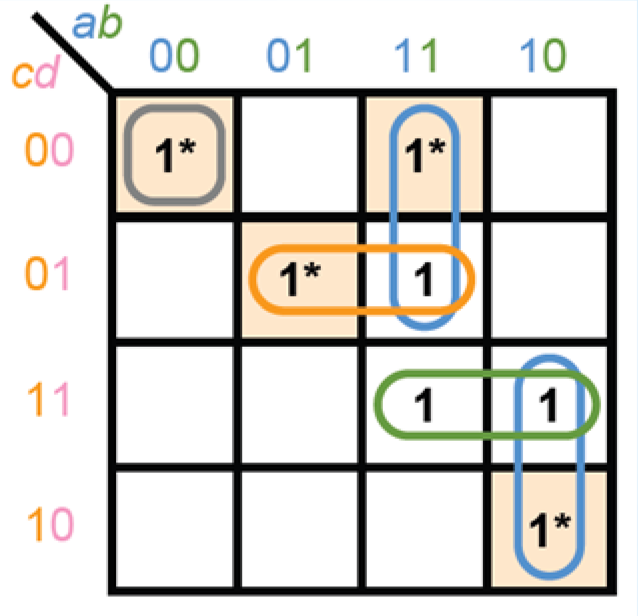




Select the least number of groups to cover all minterms.

Two Solutions





$f(a,b,c,d) = a'b'c'd' + abc' + ab'c + bc'd + abd$ or $f(a,b,c,d) = a'b'c'd' + abc' + ab'c + bc'd + acd$

We can choose either or

Re-visit this example with K-map

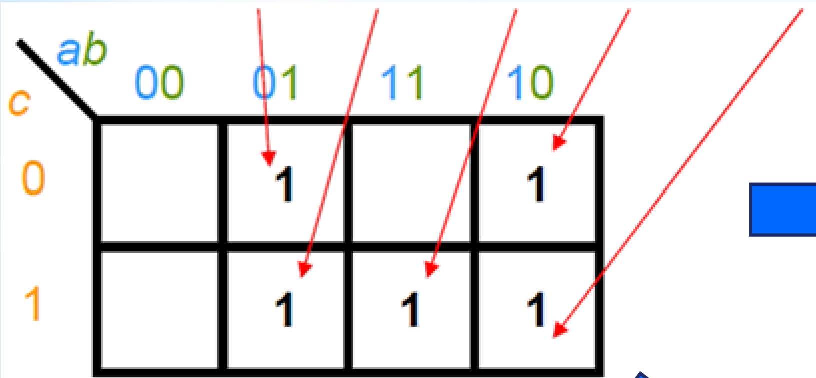
- e.g. Given 4 equivalent Boolean functions f_1 to f_4 expressed in SOP form already (example in page 6)

- $f_1(a,b,c) = a'bc' + a'bc + ab'c' + ab'c + abc$ (5 product terms, 15 literals)
- $f_2(a,b,c) = a'b + ab' + abc$ (3 product terms, 7 literals)
- $f_3(a,b,c) = a'b + ab' + ac$ (3 product terms, 6 literals)
- $f_4(a,b,c) = a'b + ab' + bc$ (3 product terms, 6 literals)

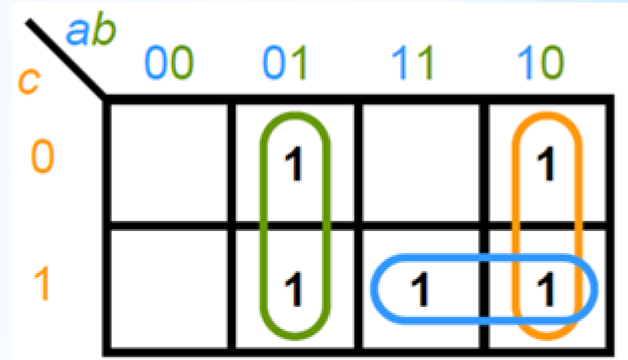
View Our Answer

First plot the 1's into K-map

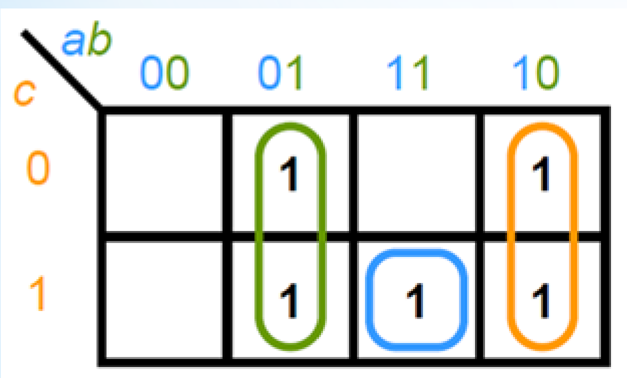
$f_1(a,b,c) = a'bc' + a'bc + abc + ab'c' + ab'c$



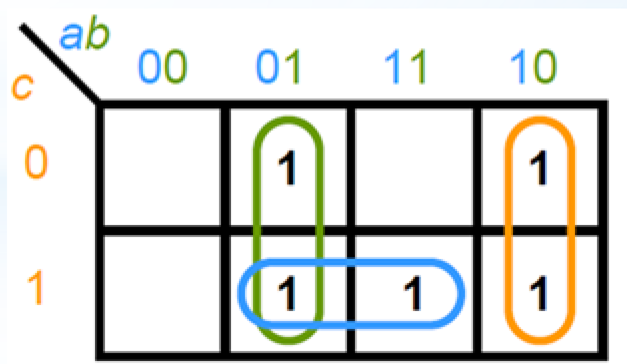
We can see that f_1 to f_4 are equivalent Boolean functions! And we can obtain the solutions f_3 & f_4 easily



$f_3(a,b,c) = a'b + ab' + ac$



$f_2(a,b,c) = a'b + ab' + abc$



$f_4(a,b,c) = a'b + ab' + bc$

Logic function in POS with K-map

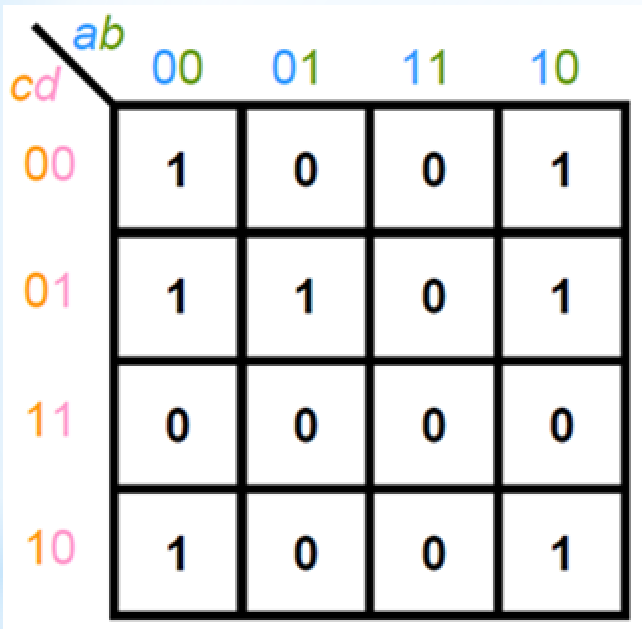
- Previous examples only show the simplified Boolean functions in SOP form
- How to obtain functions in POS form
  - Step 1:- Group the 0s to obtain *f'* in SOP form
  - Step2:- Apply DeMorgan's Theorem to find *f* in POS form

or

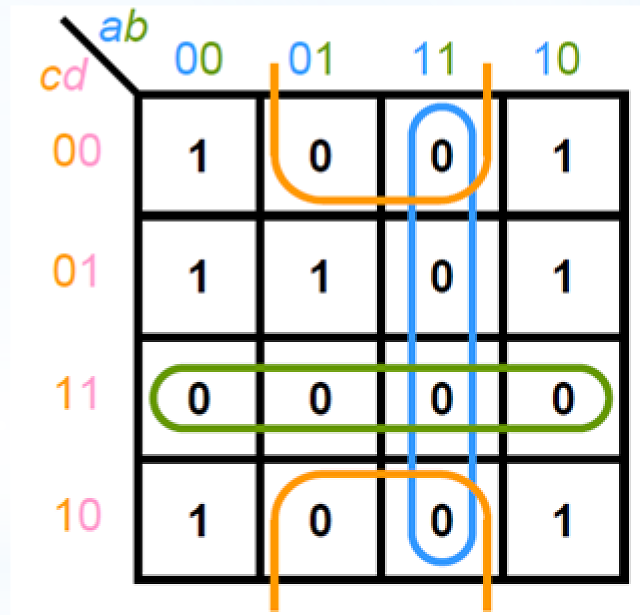
  - Group the 0s and express them directly in POS with Maxterms (require to complement the variables with +)

Example: Find POS

Simplify $f(a,b,c,d) = \sum m(0,1,2,5,8,9,10)$ in POS form



Fill the 1s and 0s into the map



Group the 0s using the same procedure as grouping the 1s

$f'(a,b,c,d) = ab + cd + bd'$

$f(a,b,c,d) = (a'+b')(c'+d')(b'+d)$

4.5 Logic functions with Don't'-care conditions

The output of Boolean functions are **incompletely specified functions**,

- For some input conditions, the outputs are unspecified
- Input condition has no effects to the function
  i.e., the output for those input are of no concern
- Output values are defined as **don't-care**
- Don't-care term can be minterm / maxterms
- Don't-care term indicates by an ×, *d*, ϕ or φ

Truth Table with Don't Care

a	*b*	*f*
0	0	0
0	1	1
1	0	1
1	1	X

What the table says is:

f is 0 if (*a* = 0 AND *b* = 0)
f is 1 if (*a* = 0 AND *b* = 1), or
(*a* = 1 AND *b* = 0)

f can be 0 or 1 if (*a* = 1 AND *b* = 1)

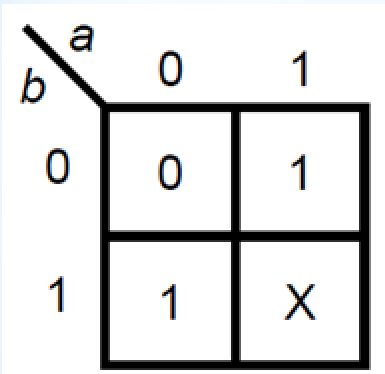
$f(a, b) = \sum m(1,2) + \sum d(3)$

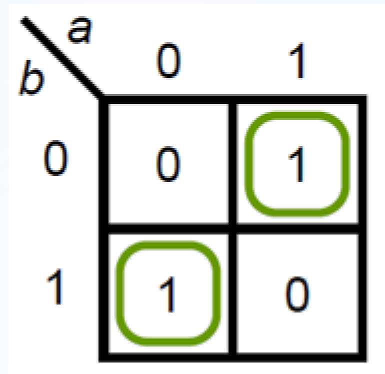
a	*b*	f_1	f_2
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

Both f_1 or f_2 of the table on the left are acceptable

K-map with Don't-care

Which solution is better?

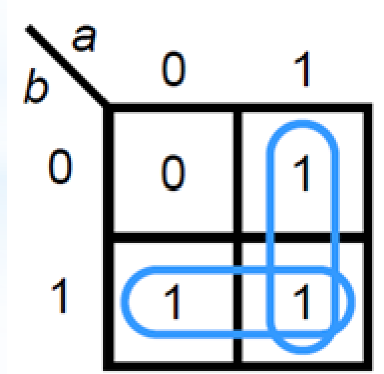




f_1 implementation

2 groups

$f = a'b + ab'$



f_2 implementation

2 groups

$f = a + b$

Procedure for K-map in don't-care cases

Simplify $f(a,b,c,d) = \sum m(1,3,7,11,15) + \sum d(0,2,5)$

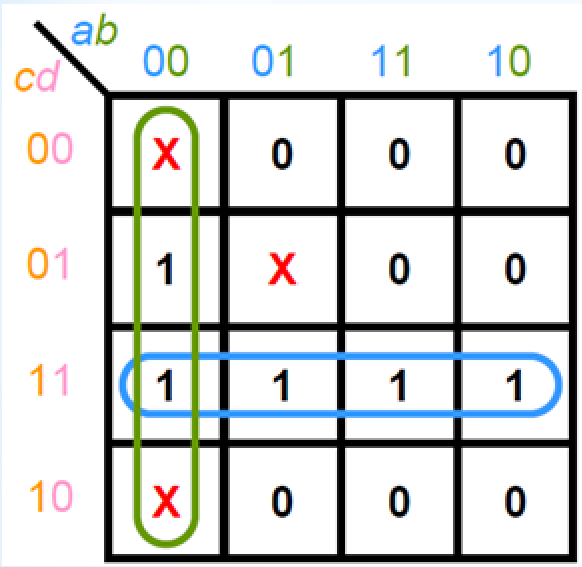
cd \ ab	00	01	11	10
00	X	0	0	0
01	1	X	0	0
11	1	1	1	1
10	X	0	0	0

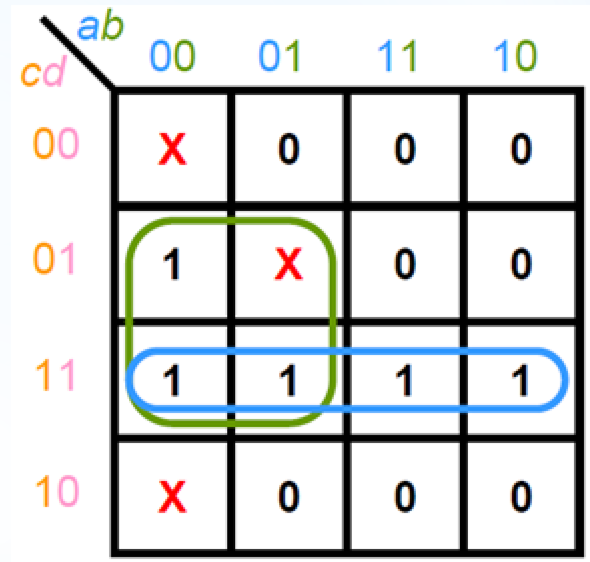
cd \ ab	00	01	11	10
00	X	0	0	0
01	1	X	0	0
11	1	1	1	1
10	X	0	0	0

$f(a,b,c,d) = a'b'd + cd$

Is it a good solution?

Other solutions





$f(a,b,c,d) = a'b' + cd$

$f(a,b,c,d) = a'd + cd$

Choose to include those Xs that eliminates more literals

4.6 Simple logic circuit design

- Design Procedure
- Examples
  - Adder
  - Code converter

4.6.1 Design Procedure

1. State the problem / specification for the design.
2. Determine the number of input variables and output variables.
3. Formulate truth tables / Boolean functions between inputs and outputs.
4. Simplification / minimization for the logic functions.
5. Design and draw the logic circuit diagram.

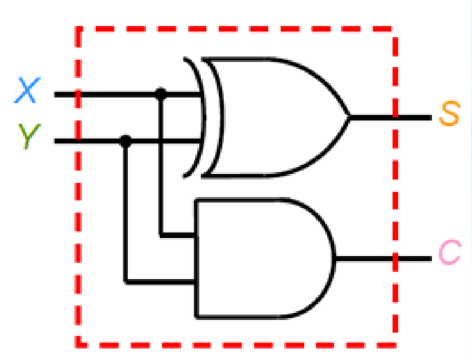
4.6.2 Adder

1-bit Half-Adder

- As known as half adder (HA)

x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S(X,Y) = \sum m(1,2)$$
$$= X \oplus Y$$
$$C(X,Y) = XY$$



1-bit Full-Adder (FA)

x	y	c_i	s	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum

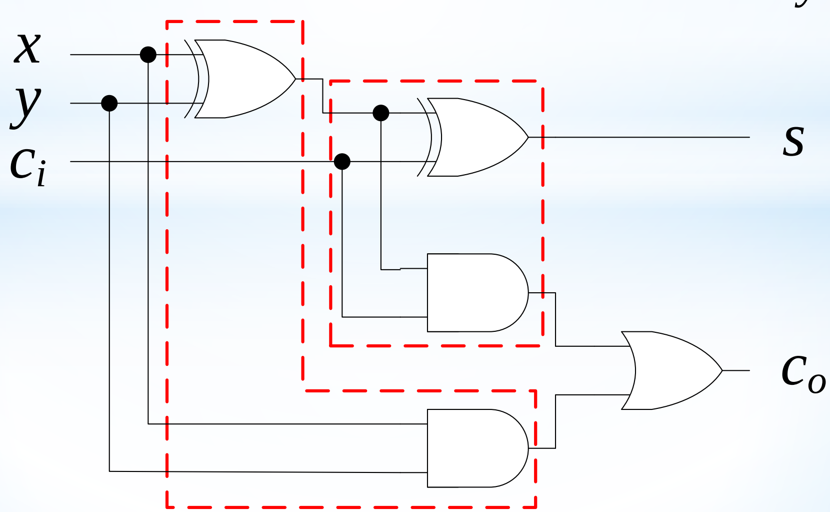
x \ yc_i				
		1		1
	1		1	

Carry out

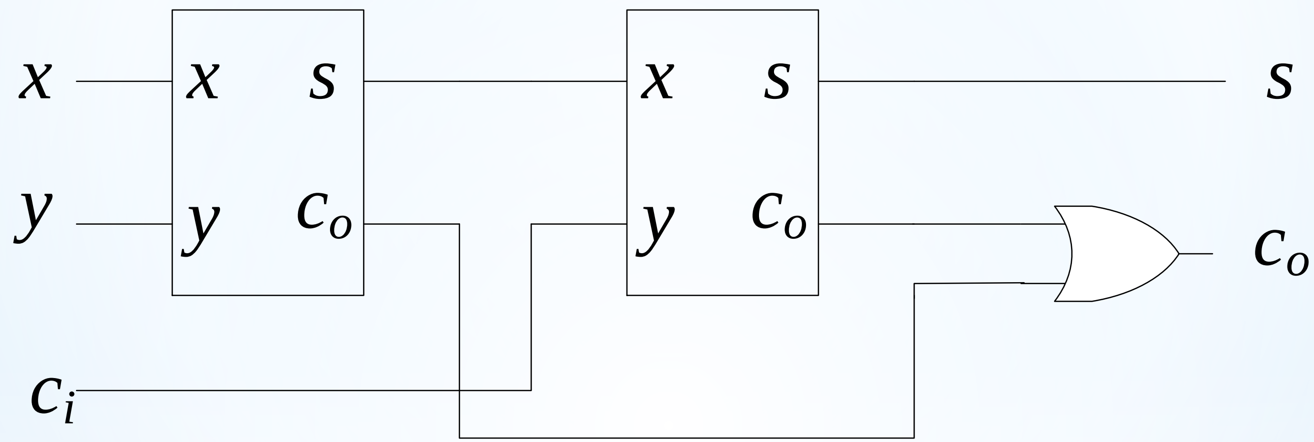
x \ yc_i				
			1	
		1	1	1

$$
\begin{aligned}
s &= \bar{x}\,\bar{y}c_i + \bar{x}y\bar{c}_i + x\bar{y}\,\bar{c}_i + xyc_i \\
&= \bar{x}(\bar{y}c_i + y\bar{c}_i) + x(yc_i + \bar{y}\,\bar{c}_i) \\
&= \bar{x}(y \oplus c_i) + x\overline{(y \oplus c_i)} \\
&= x \oplus y \oplus c_i
\end{aligned}
$$

$$
\begin{aligned}
c_o &= yc_i + xc_i + xy \\
&= xy + x(y + \bar{y})c_i + y(x + \bar{x})c_i \\
&= xy + xyc_i + x\bar{y}c_i + \bar{x}yc_i \\
&= xy(1 + c_i) + (x\bar{y} + \bar{x}y)c_i \\
&= xy + (x \oplus y)c_i
\end{aligned}
$$

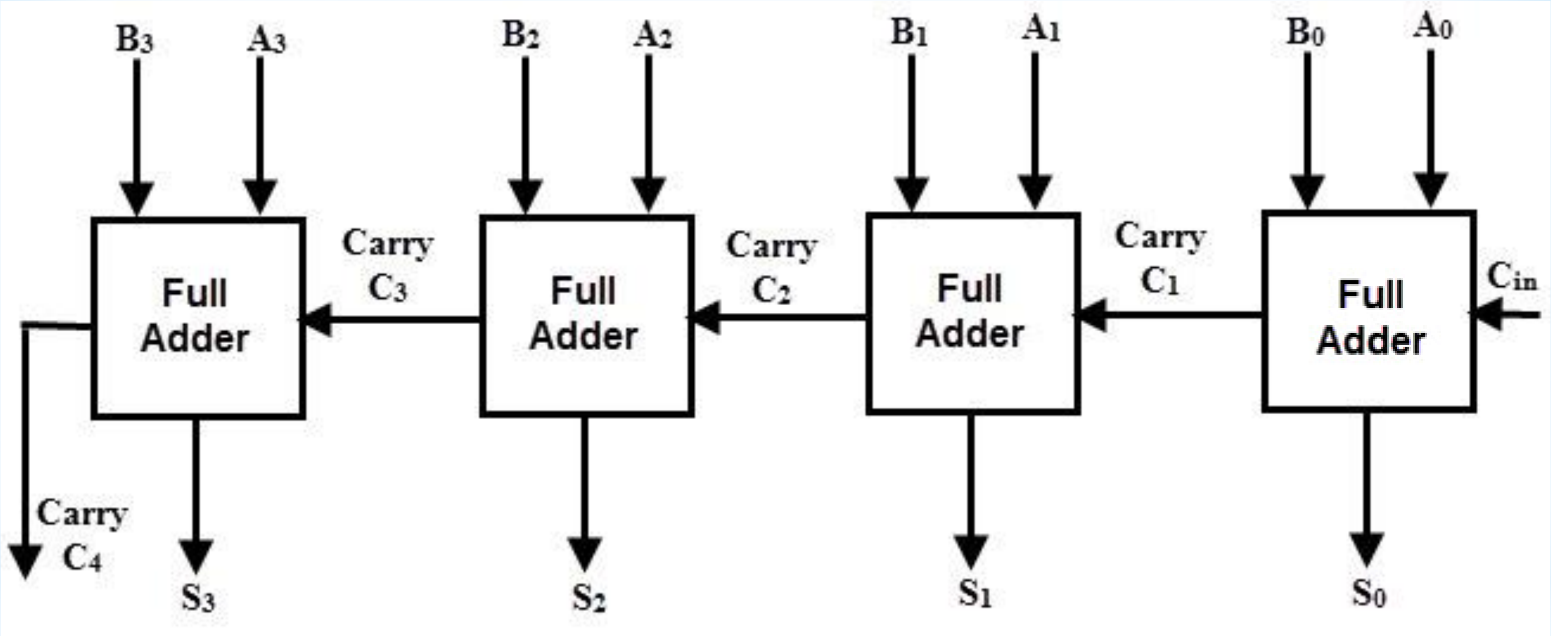


1-bit Full-Adder (FA)

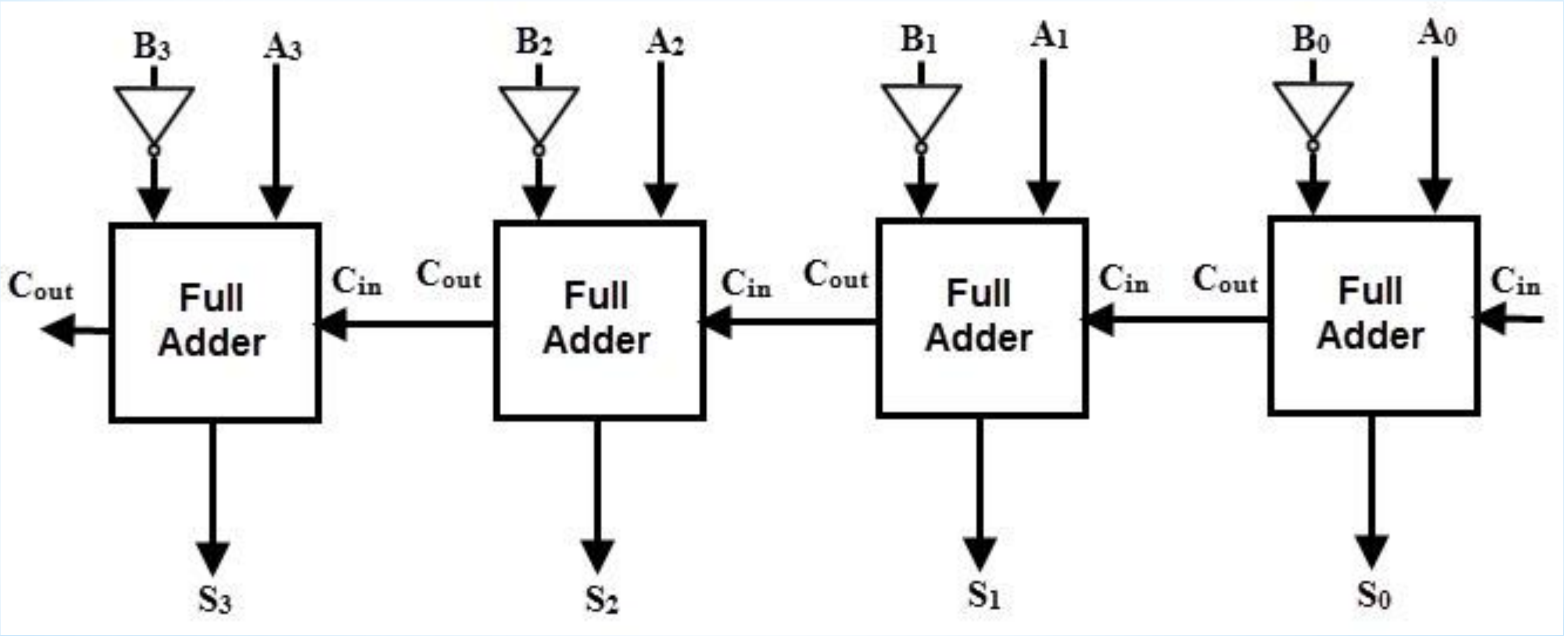


Arrangement of 2 half-adders to form a full-adder

4-bit parallel adder with 1-bit 4 FA modules

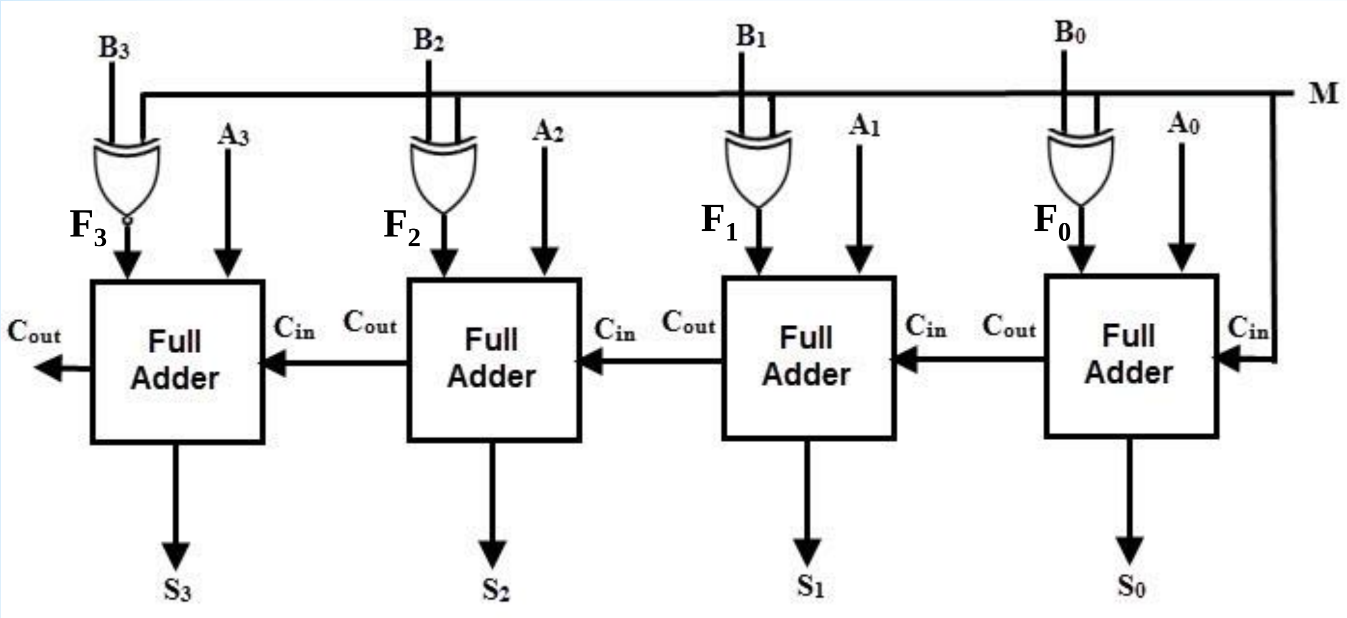


4-bit parallel subtractor with 1-bit 4 FA modules



$$A - B = A + (2\text{'s of } B) = A + B' + 1$$

4-bit Adder/Subtractor



M = 0 → Adder
M = 1 → Subtractor

M	B	F
0	0	0
0	1	1
1	0	1
1	1	0

$F = B \oplus M$

4.6.2 Code Converter

State the case

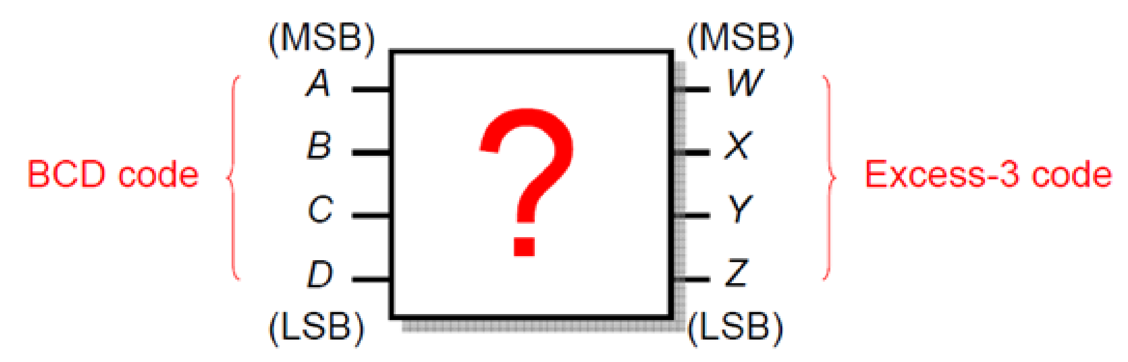
Design a circuit to convert the BCD to the Excess-3 code
- *A*, *B*, *C*, *D* are the input of BCD.
- *W*, *X*, *Y*, *Z* are the output of Excess-3 code.
- The output functions are:

  $W(A, B, C, D)$

  $X(A, B, C, D)$

  $Y(A, B, C, D)$

  $Z(A, B, C, D)$



Decimal digit	Input (8421 code)				Output (Excess-3 code)			
	A	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X

Unused outputs can consider as DON'T CARE.

- There are four variables (A,B,C,D) in the functions
- Each output variable depends on 4 variables
- So we need 4 four-variable K-maps

$$W(A,B,C,D) = \sum m(5,6,7,8,9) + \sum d(10,11,12,13,14,15)$$
$$X(A,B,C,D) = \sum m(1,2,3,4,9) + \sum d(10,11,12,13,14,15)$$
$$Y(A,B,C,D) = \sum m(0,3,4,7,8) + \sum d(10,11,12,13,14,15)$$
$$Z(A,B,C,D) = \sum m(0,2,4,6,8) + \sum d(10,11,12,13,14,15)$$

K-map for *W*:

CD \ AB	00	01	11	10
00			x	1
01		1	x	1
11		1	x	x
10		1	x	x

K-map for *X*:

CD \ AB	00	01	11	10
00		1	x	
01	1		x	1
11	1		x	x
10	1		x	x

K-map for *Y*:

CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11	1	1	x	x
10			x	x

K-map for *Z*:

CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11			x	x
10	1	1	x	x

$W = A + BC + BD$

CD \ AB	00	01	11	10
00			x	1
01		1	x	1
11		1	x	x
10		1	x	x

$X = B'C + B'D + BC'D'$

CD \ AB	00	01	11	10
00		1	x	
01	1		x	1
11	1		x	x
10	1		x	x

$Y = CD + C'D'$

CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11	1	1	x	x
10			x	x

$Z = D'$

CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11			x	x
10	1	1	x	x

Logic functions and circuit

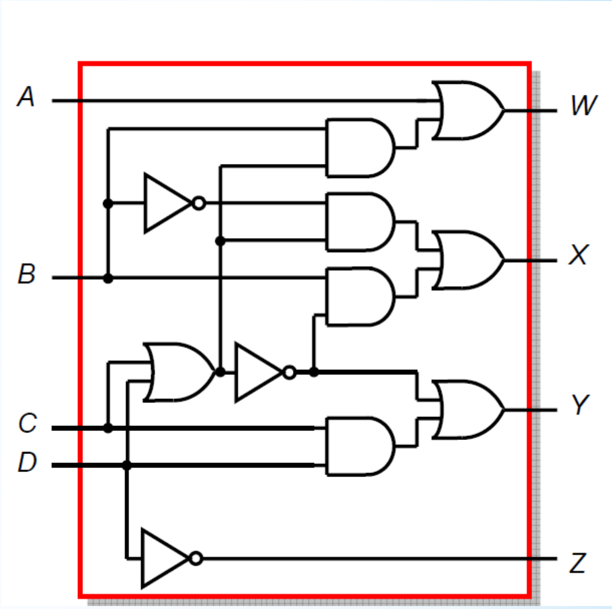
From previous K-maps,

$W = A + BC + BD$

$X = B'C + B'D + BC'D'$

$Y = CD + C'D'$

$Z = D'$



We further optimize them (optional)

$W = A + BC + BD = A + B(C + D)$

$X = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$

$Y = CD + C'D' = CD + (C + D)'$

DeMorgan's Law