Lecture 4

Karnaugh Maps

Learning outcomes

- Construct a Karnaugh map (K-map) for 3 to 6 variables
- Identify adjacent cells in a K-map
- Use a K-map to obtain a minimum SOP or POS expression.
- Exploit “don’t cares” conditions in the K-map to simplify Boolean expressions.

Algebraic Manipulation

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

$f = x'y'z + x'yz' + x'yz + xy'z + xyz$ (Sum-of-Minterms)

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

$f = x'z(y' + y) + x'yz' + xz(y' + y)$ Distributive $\cdot$ over +

$f = x'z + x'yz' + xz$ Uniting (7 literals)

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

$f = (x' + x)z + x'yz'$ Distributive $\cdot$ over +

$f = z + x'yz'$ Uniting (4 literals)

$f = (z + x'y)(z + z')$ Distributive + over $\cdot$

$f = z + x'y$ Uniting (3 literals)

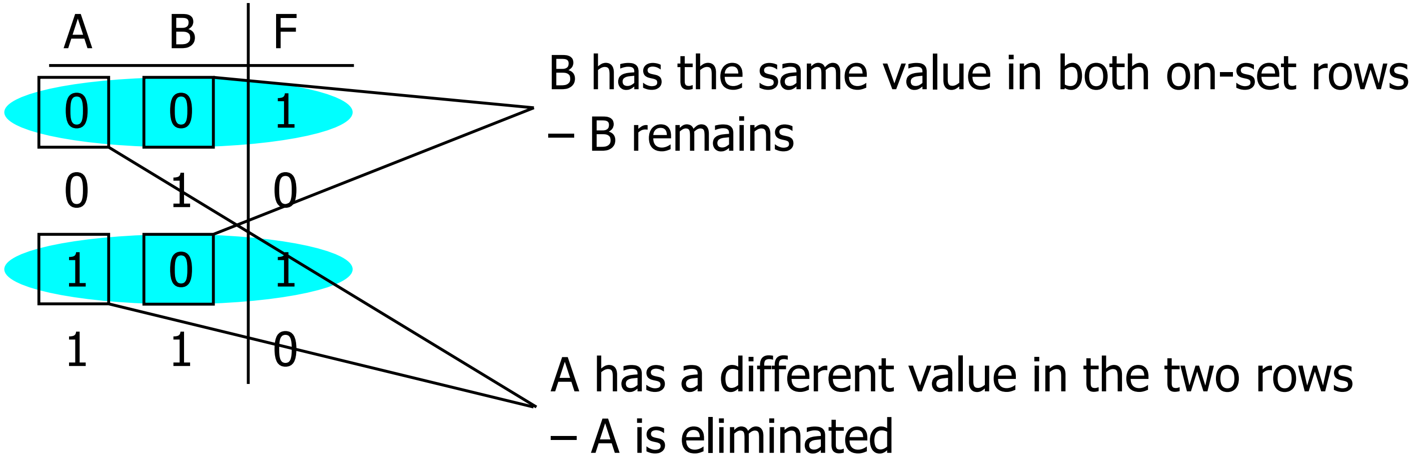
Algebraic Manipulation – issues

- 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

The uniting theorem

- Key tool to simplification: A (B’ + B) = A
- Essence of simplification of two-level logic
  - find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

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



A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

B has the same value in both on-set rows
– B remains

A has a different value in the two rows
– A is eliminated

- The On-set contains all input combinations for which the function is 1.
- The Off-set contains all input combinations for which the function is 0.
- The dc-set or “don’t care” set contains all input combinations for which the function is X.

Karnaugh maps (K-maps)

- Introduced by Maurice Karnaugh, an American physicist working at Bell Labs.
- When properly used, the Karnaugh map can provide a minimal 2-level SOP or POS implementation.
- Commonly used to minimize Boolean equations with two to 4 variables
- The K-map is a reorganized graphical representation of the truth table.
- Its key idea is to place the minterms that differ in only one variable adjacent to one another, so that one can "visualize" opportunities to apply the **uniting theorem** **A(B' + B) = A** and **eliminate** one variable.

2-variable K-map - organization

- In a 2-variable K-map, one square represents a minterm with 2 variables.
- Minterms m_0 and m_1 are adjacent since they differ in the value of a single variable: y.
- Minterms m_2 and m_3 are adjacent since they differ in the value of a single variable: y.
- Minterms m_0 and m_2 are adjacent since they differ in the value of a single variable: x.
- Minterms m_1 and m_3 are adjacent since they differ in the value of a single variable: x.

x y	f
0 0	m_0
0 1	m_1
1 0	m_2
1 1	m_3

Truth Table

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

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

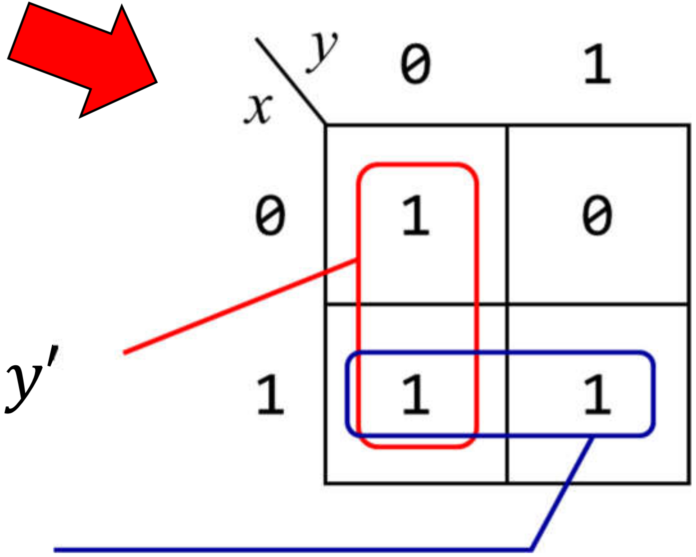
Two-variable K-map

2-variable K-map - Example

- $f = m_0 + m_2 + m_3 = x'y' + xy' + xy$ (6 literals)
- Two adjacent squares containing 1's can be combined to eliminate one variable.

x y	m_i	f
0 0	m_0	1
0 1	m_1	0
1 0	m_2	1
1 1	m_3	1

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



- $m_0 + m_2 = x'y' + xy' = (x' + x)y' = y'$
- $m_2 + m_3 = xy' + xy = x(y' + y) = x$
- Therefore, f can be simplified as: $f = x + y'$ (2 literals)

K-map minimization

- K-map minimization using minterms results in a minimum Sum-of-Products.

- **SOP expressions** are simplified using K-maps by grouping together adjacent squares which both contain a 1 (remember, K-maps are a graphical way of performing Boolean algebra).

- Larger groups can be formed but they must be square or rectangular with a width and length that must be a power of 2: 1, 2, 4, 8, etc…

- Each grouping of 1's formed is equivalent to replacing all the minterms represented by the squares in the group by a single expression formed by the **AND of the variables which remain constant every square of the grouping**. Thus, all those variables changing in the grouping are eliminated.

- The larger the grouping, the greater the degree of simplification:
  - Grouping 2 adjacent squares (minterms) eliminates 1 variable.
  - Grouping 4 adjacent squares (minterms) eliminates 2 variables.
  - Grouping 8 adjacent squares (minterms) eliminates 3 variables.
  - Grouping 2^N adjacent squares (minterms) eliminates N variables.

3-variable K-map - Organization

- For a 3-input logic circuit, the truth table will have 2^3=8 rows represented by 8 squares in the K-map.
- Numbering scheme based on Gray–code

  e.g., 00, 01, **11**, 10 (**be careful!**)

  only a single bit changes in code for adjacent squares
- Each K-map square is adjacent to 3 other squares.
- In a 3-variable K-map, one square represents a minterm with 3 variables.
- Minterms in adjacent squares can always be manipulated using the uniting theorem.

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

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

3-variable K-map – Example 1

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

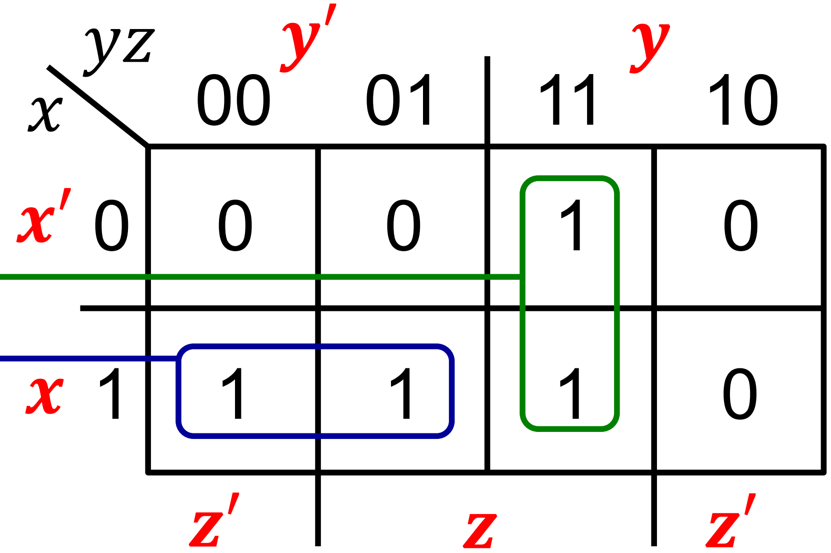
  $f = x'yz + xy'z' + xy'z + xyz$ (12 literals)

- Fill the K-map squares with the function values

- Find possible adjacent squares

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

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



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

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

3-variable K-map – Example 2

- Simplify the Boolean function: $f(x, y, z) = \sum m(3, 4, 6, 7)$

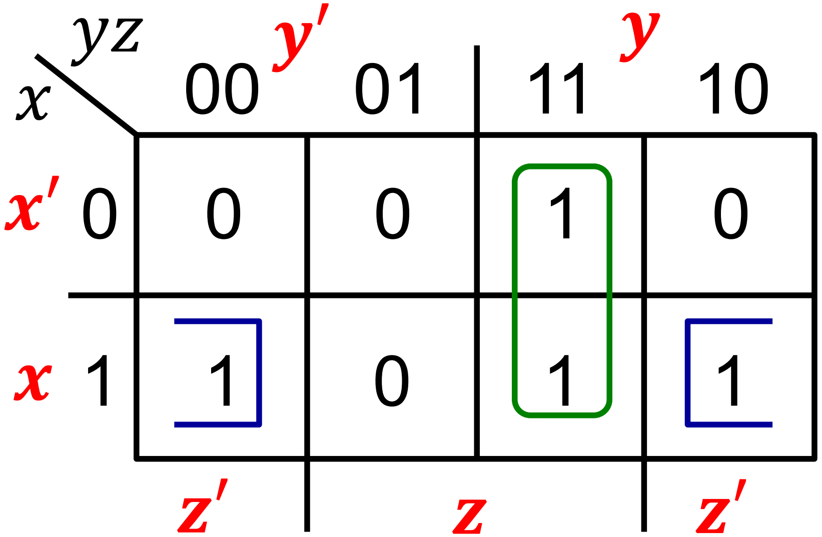
  $f = x'yz + xy'z' + xyz' + xyz$ (12 literals)

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

- Squares on the corners can also be combined:

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

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



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

3-variable K-map – Example 3

- Simplify the Boolean function: $f(x, y, z) = \sum m(2, 3, 5, 6, 7)$

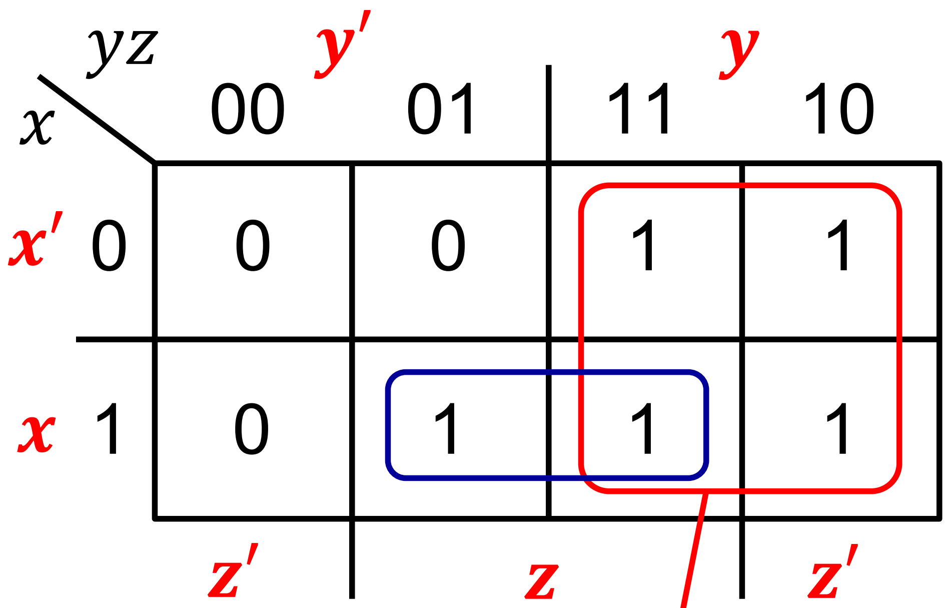
$$f = x'yz' + x'yz + xy'z + xyz' + xyz \quad \text{(15 literals)}$$

- The 4 minterms grouping reduces to term y

- The 2 minterms grouping reduces to term xz

- Thus, we have:

$$f = y + xz \quad \text{(3 literals)}$$



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

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

4-variable K-map - Organization

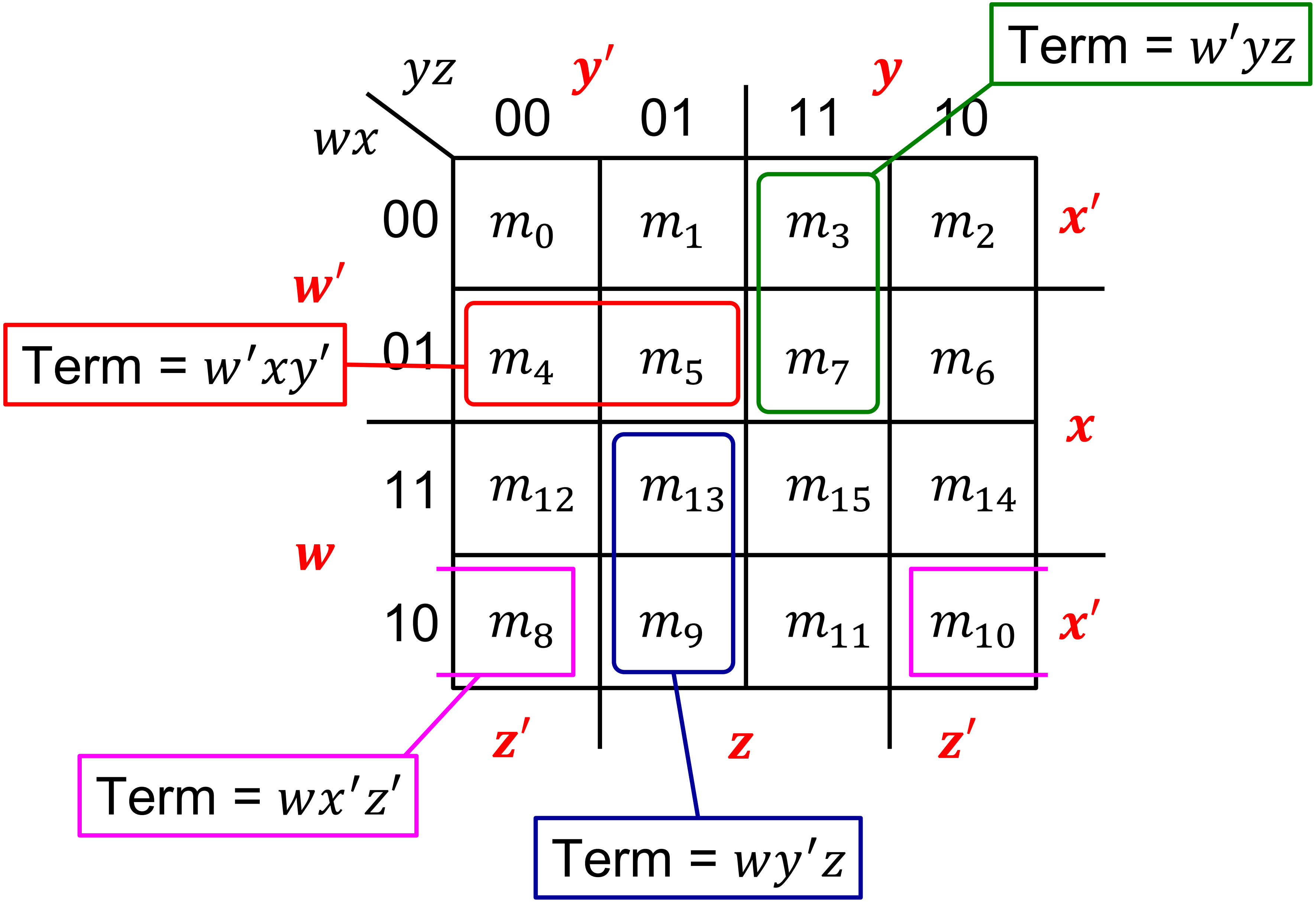
- For a 4-input logic circuit, the truth table will have 2^4=16 rows represented by 8 squares in the K-map.
- Numbering scheme based on Gray–code
  - e.g., 00, 01, **11**, 10 (**be careful!**)
  - only a single bit changes in code for adjacent squares
- Each K-map square is adjacent to 4 other squares.
- In a 4-variable K-map, one square represents a minterm with 4 variables.

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

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

4-variable K-map – Grouping 2 squares

- Grouping 2 adjacent squares (minterms) eliminates 1 variable.



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

Term = $w'yz$

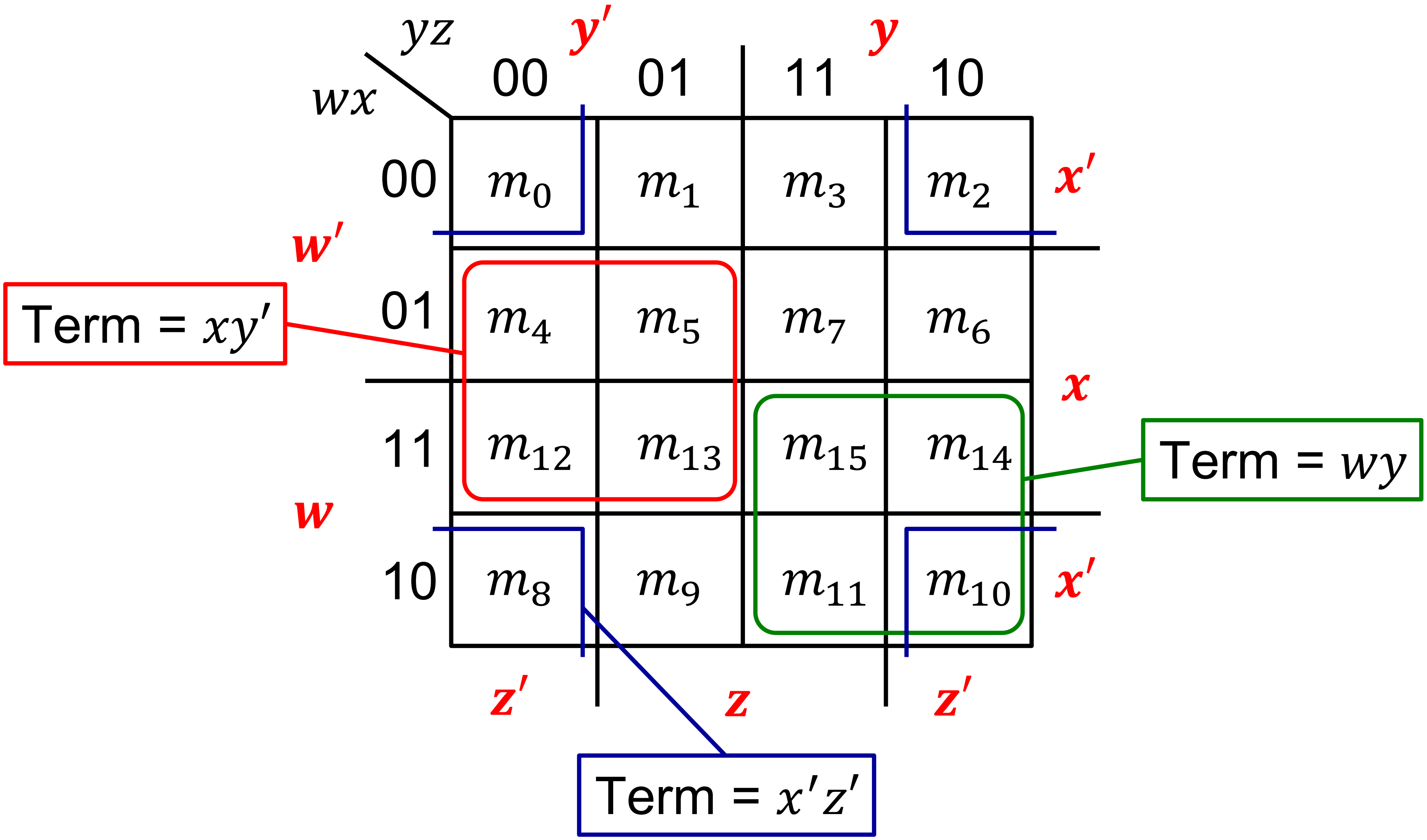
Term = $w'xy'$

Term = $wx'z'$

Term = $wy'z$

4-variable K-map – Grouping 4 squares

- Grouping 4 adjacent squares (minterms) eliminates 2 variables.



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

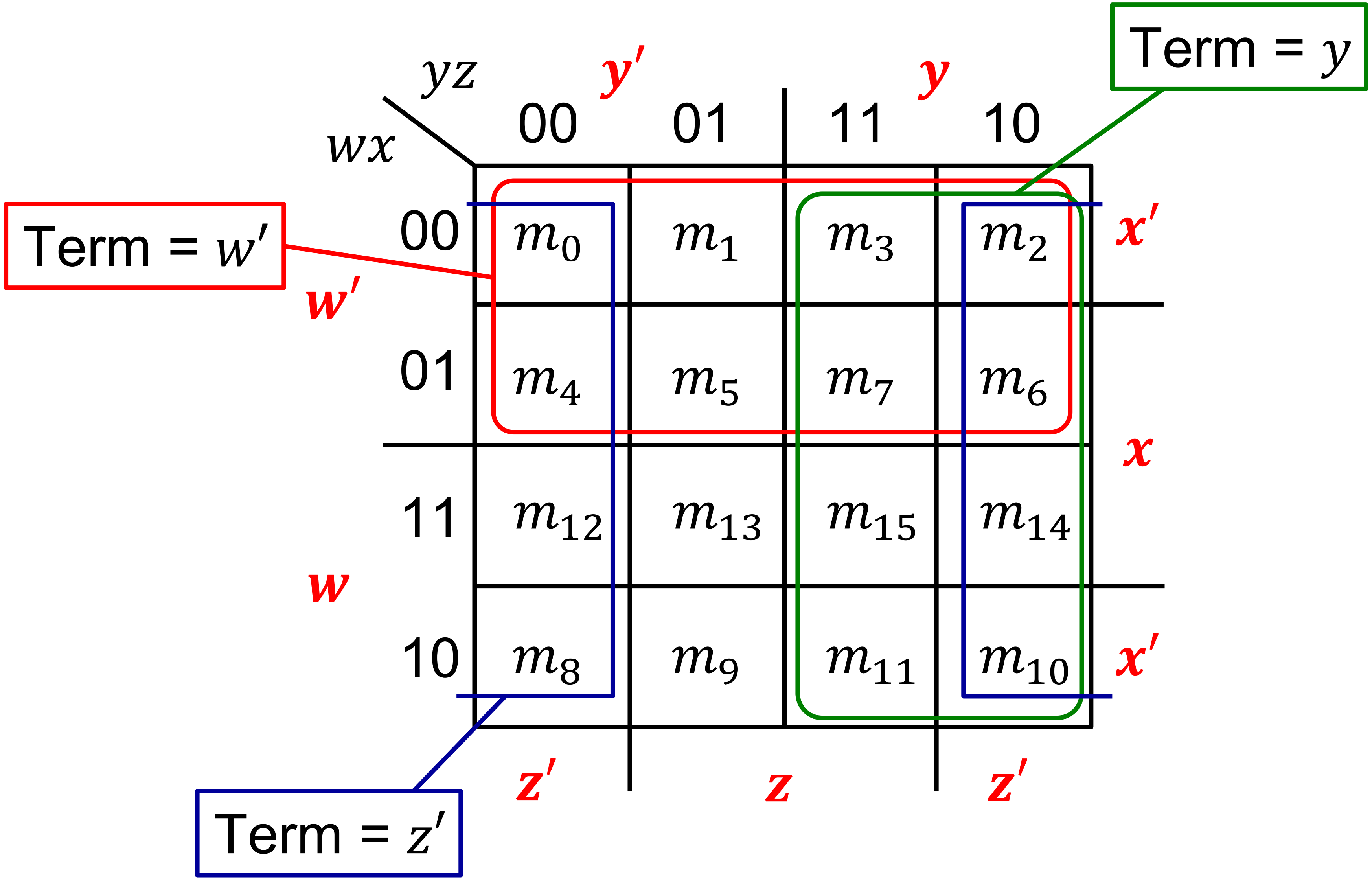
Term = xy'

Term = wy

Term = $x'z'$

4-variable K-map – Grouping 8 squares

- Grouping 8 adjacent squares (minterms) eliminates 3 variables.



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

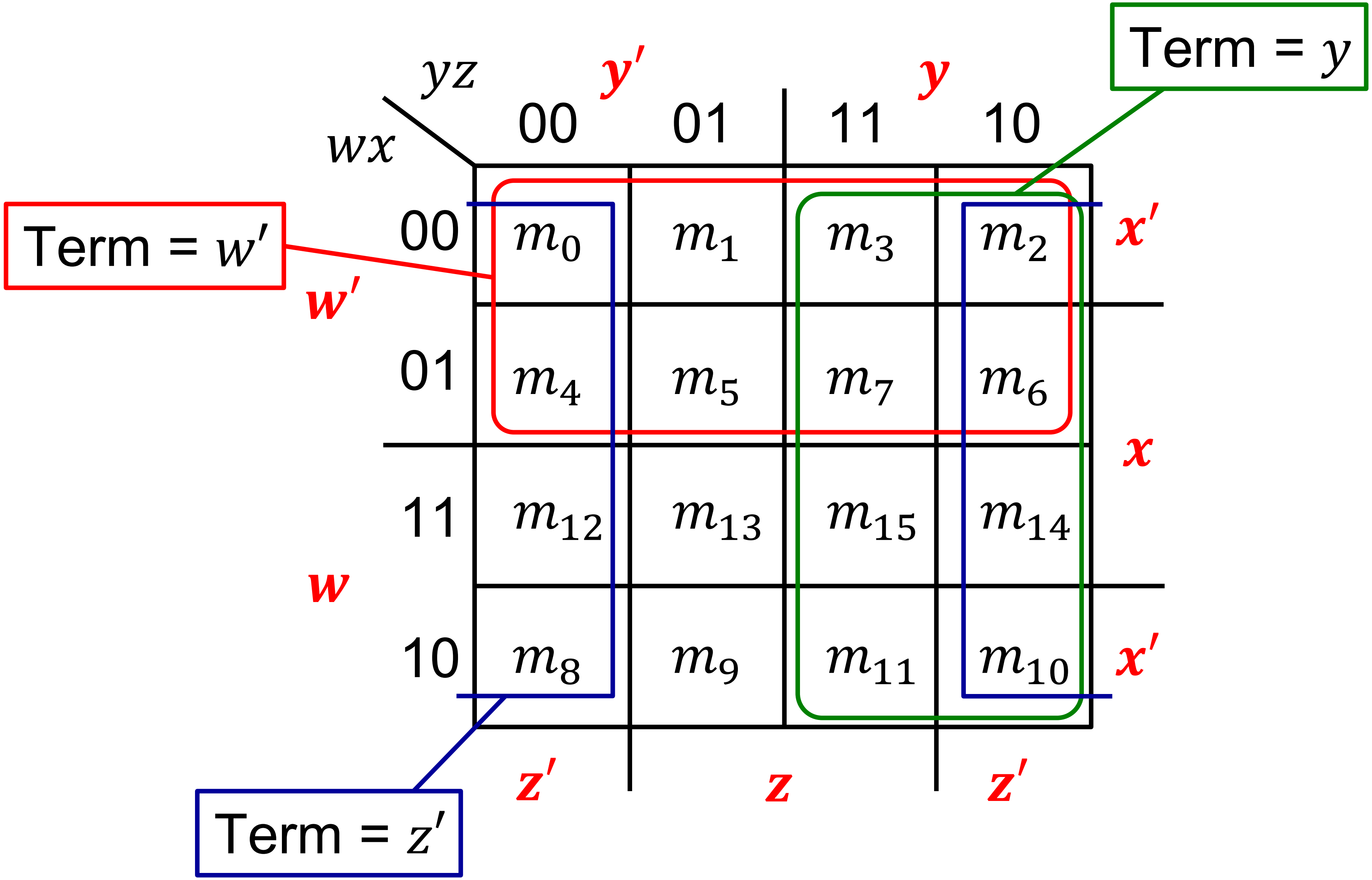
Term = y

Term = w'

Term = z'

4-variable K-map – Grouping 8 squares

- Grouping 8 adjacent squares (minterms) eliminates 3 variables.



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

Term = y

Term = w'

Term = z'

K-maps with don’t cares

- In some applications of logic circuits, it is impossible for certain input conditions to occur, and in these cases the circuit output will not be specified. These “don’t care” states can be used to reduce a circuit’s complexity.

- To use a Karnaugh map, it is necessary to make an entry in every square (note that we can leave the square blank when the entry is “0”)

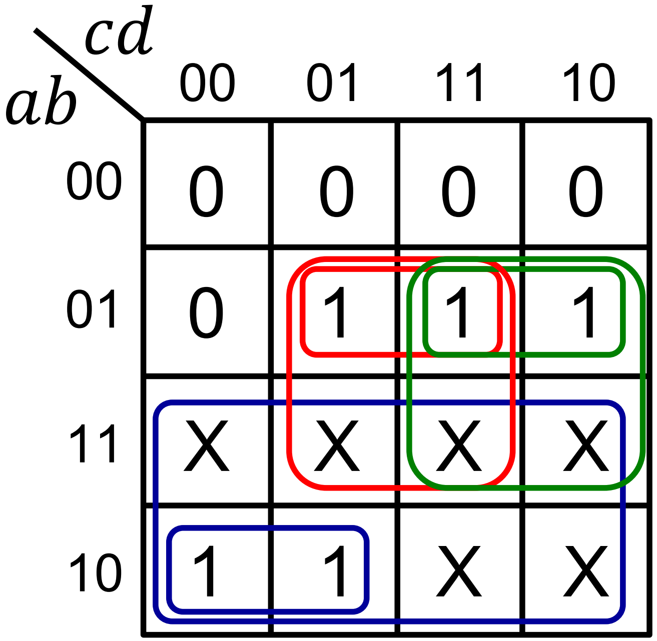
- We can choose the value of “don’t care” states such that they can be used to assist in minimization.

K-maps with don’t cares - Example

- Consider the Boolean function f defined by:

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

- When all X are taken as 0’s: $f = a'bd + a'bc + ab'c'$
  - 9 literals
- When all X are taken as 1’s: $f = a + bd + bc$
  - 5 literals
- One can choose the value of “don’t care” states to assist in the minimization of the Boolean expression.



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

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

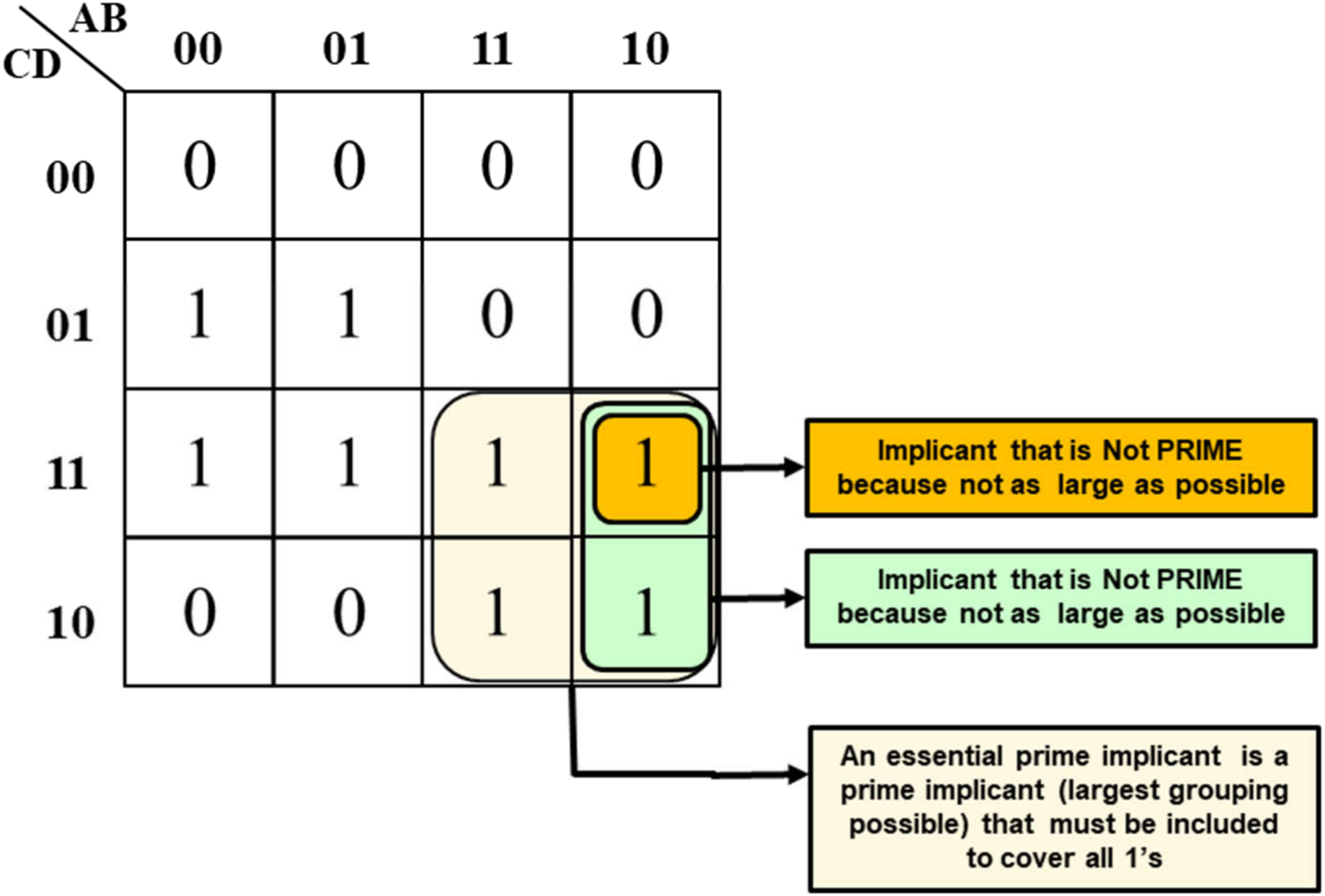
Definition of terms for two-level simplification

- Implicant
  - A product term that corresponds to a grouping of adjacent 1’s and/or don’t cares.
  - The grouping must be square or rectangular with a width and length that must be a power of 2: 1, 2, 4, 8, etc…
  - The larger the implicant, the smaller the product term.
- Prime implicant
  - A prime implicant is an implicant that cannot be completely contained in any other implicant.
  - A prime implicant cannot be combined with another implicant to form a smaller product term.
- Essential prime implicant
  - prime implicant is essential if it is the only prime implicant that covers (includes) one or more 1’s.
  - will participate in ALL possible covers of the 1’s of the function.
  - Don’t cares are used to form prime implicants but not to make implicant essential

Algorithm for systematic 2-level simplification

- Algebraic simplification: not an algorithmic/systematic procedure
  - How do you know when the minimum realization has been found?
  - How do you deal with al large number of inputs?
- Algorithm for a minimum SOP expression from a Karnaugh map:
  - Step 1: choose an element of the ON-set
  - Step 2: find "maximal" groupings of 1s and Xs adjacent to that element
    - consider top/bottom row, left/right column, and corner adjacencies
    - this forms prime implicants (number of elements always a power of 2)
  - Repeat Steps 1 and 2 to find all prime implicants
  - Step 3: revisit the 1s in the K-map
    - if covered by single prime implicant, it is essential, and participates in final cover
    - 1s covered by essential prime implicant do not need to be revisited
  - Step 4: if there remain 1s not covered by essential prime implicants
    - select the smallest number of prime implicants that cover the remaining 1s

Implicants - Examples



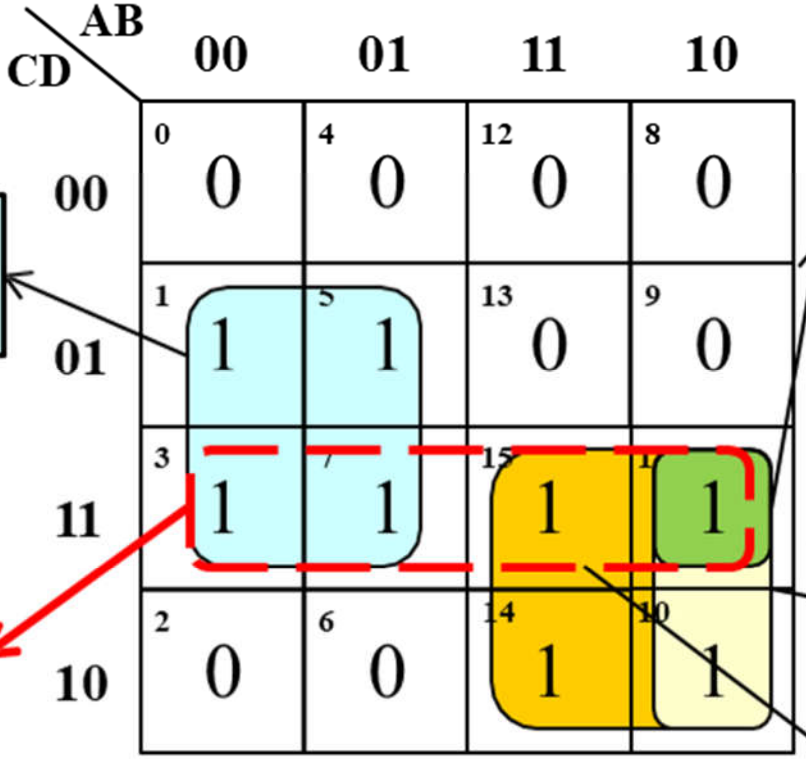
CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	1	1	1
10	0	0	1	1

Implicant that is Not PRIME because not as large as possible

Implicant that is Not PRIME because not as large as possible

An essential prime implicant is a prime implicant (largest grouping possible) that must be included to cover all 1's

Implicants - Examples



CD \ AB	00	01	11	10
00	0 0	4 0	12 0	8 0
01	1 1	5 1	13 0	9 0
11	3 1	7 1	15 1	11 1
10	2 0	6 0	14 1	10 1

An essential prime implicant

A prime implicant, but not an ESSENTIAL implicant because it is not needed to cover all 1's in the function

An implicant

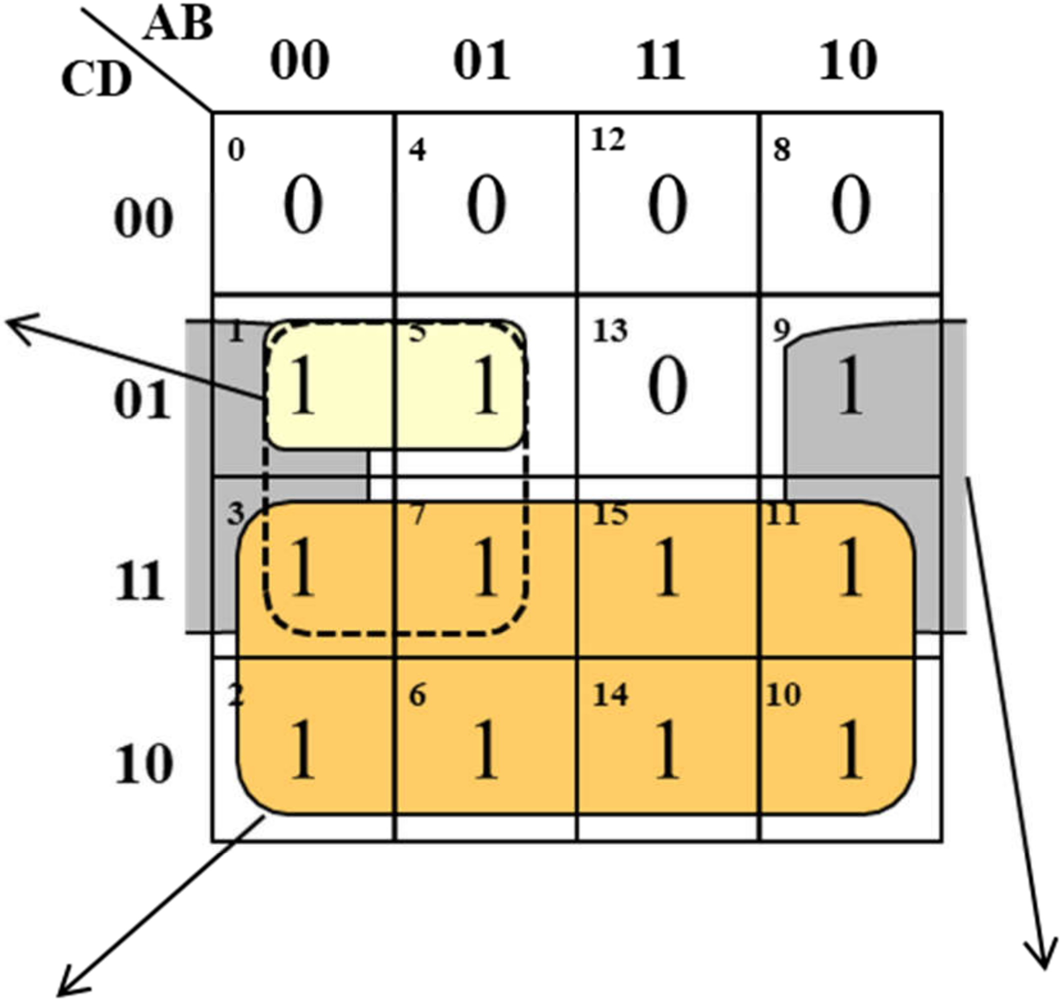
Not PRIME because not as large as possible

An implicant

An essential prime implicant

An essential prime implicant (largest grouping possible, that must be included to cover all 1's)

Implicants - Examples



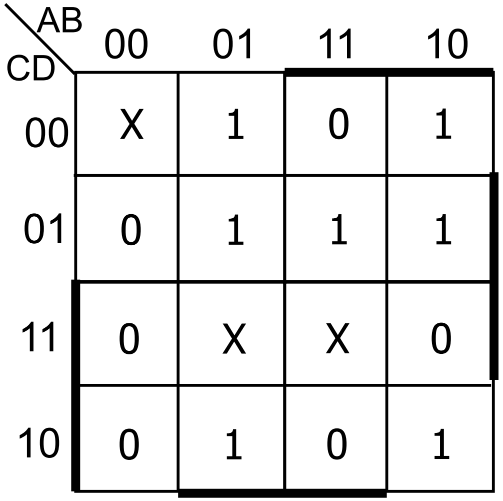
CD \ AB	00	01	11	10
00	0 0	4 0	12 0	8 0
01	1 1	5 1	13 0	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 1	14 1	10 1

An implicant, but not a PRIME implicant because it is not as large as possible (should expand to combo's 3 and 7)

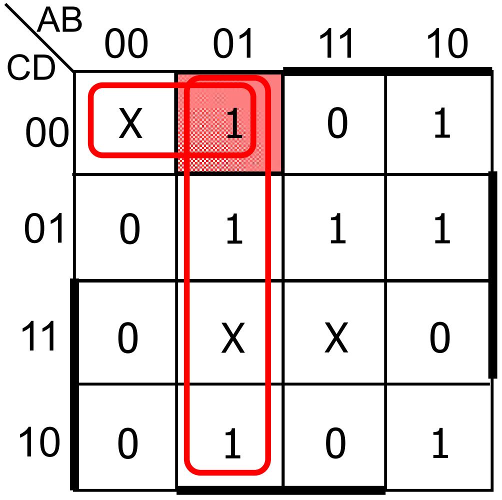
An essential prime implicant (largest grouping possible, that must be included to cover all 1's)

An essential prime implicant

Algorithm for two-level simplification (example)

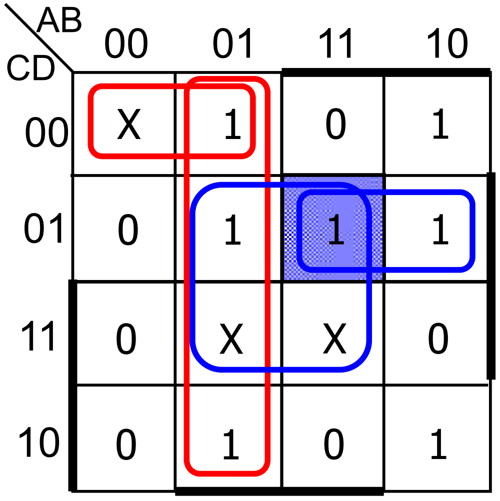


CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1



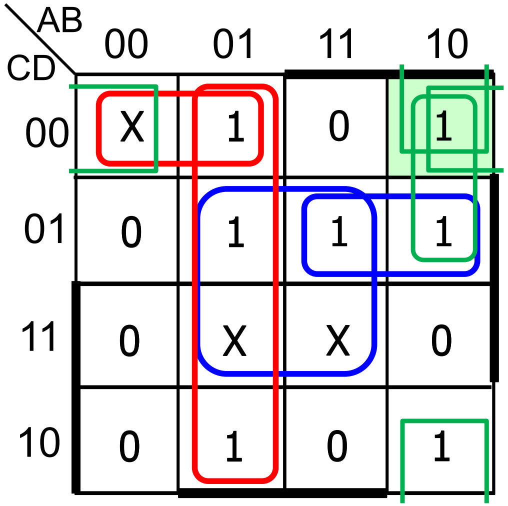
CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

2 primes around A'BC'D'



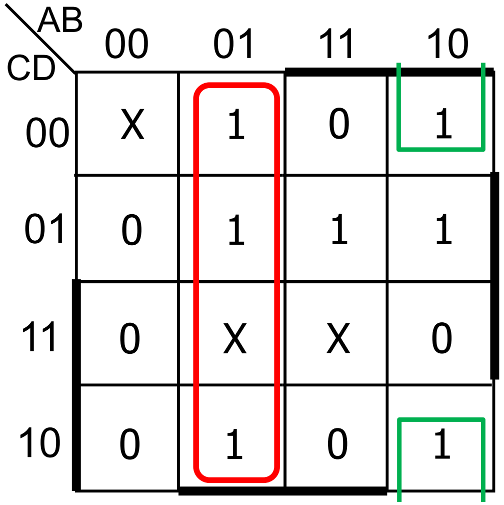
CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

2 primes around ABC'D



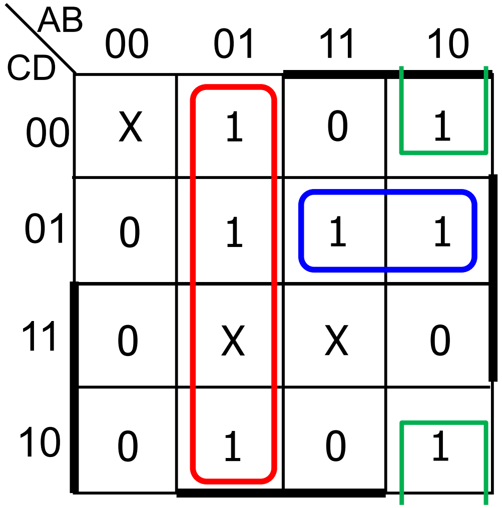
CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

3 primes around AB'C'D'



CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

2 essential primes

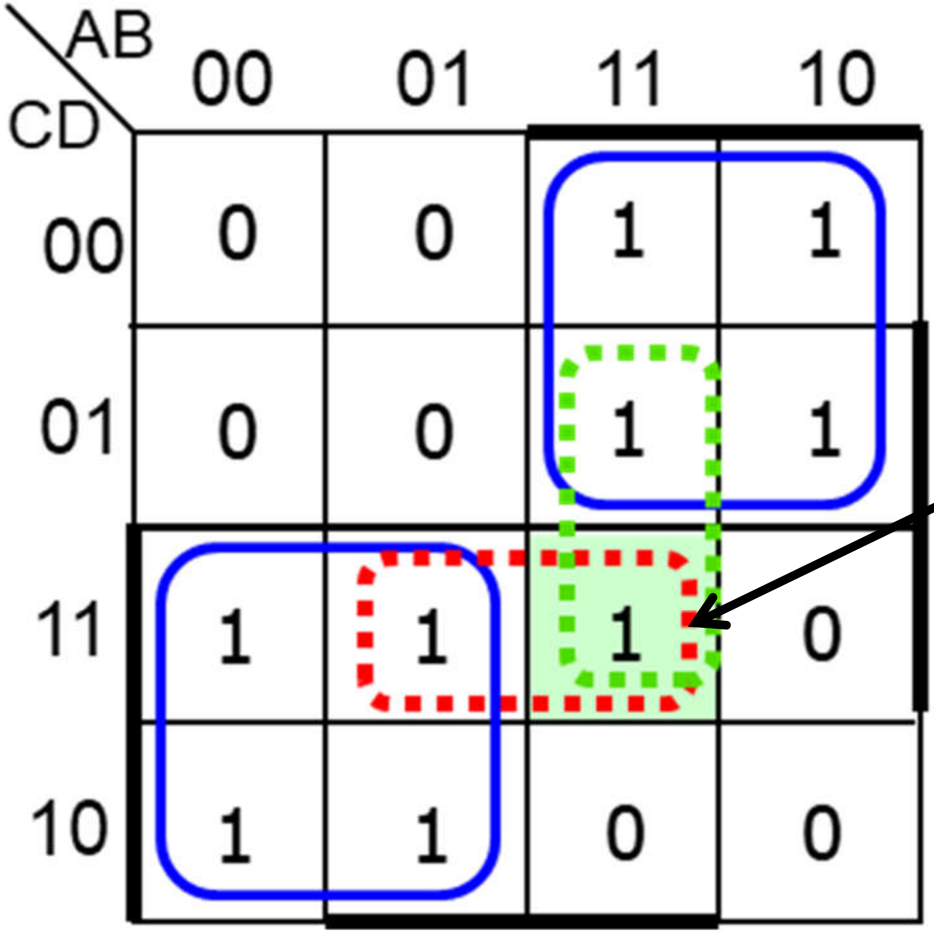


CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

minimum cover (3 primes)

Not always a single minimal expression

- When multiple minimal expressions can be found, one can select any of the minimal covers.



CD \ AB	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	1	0
10	1	1	0	0

How to group this “1”?

Notes on K-maps

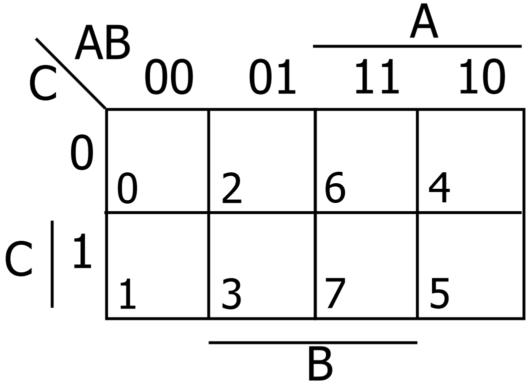
a) Form the largest possible groups.

b) Construct the smallest possible number of groups provided that rule (c) is obeyed.

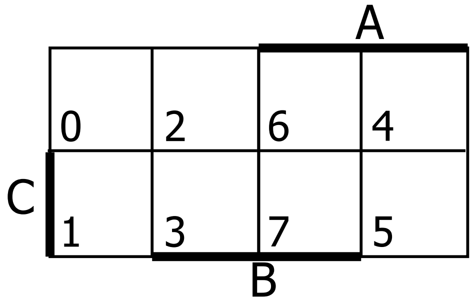
c) All squares with a “1” entry must be included in at least one group.

d) Squares should not be included in more than one group unless it enables a small group to be replaced by a larger one.

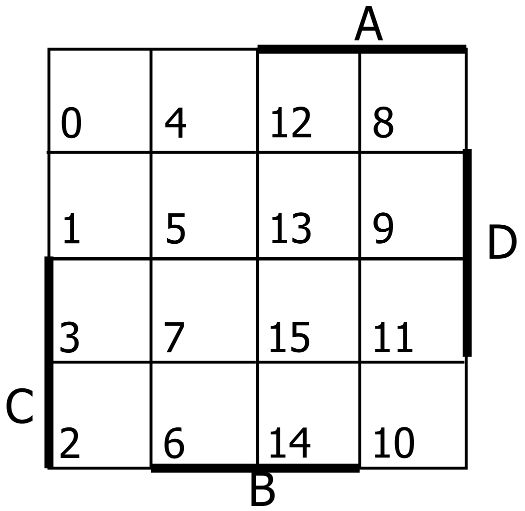
Alternate labelings for Karnaugh maps



C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5



0	2	6	4
1	3	7	5



0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

13 = 1101= ABC′D

Karnaugh Map Construction

- Every square represents 1 input combination
- Must label axes in Gray code order:
- Fill in squares with given function values

F(X, Y, Z)=Σm(1,4,5,6)

Z \ XY	00	01	11	10
0	0 0	2 0	6 1	4 1
1	1 1	3 0	7 0	5 1

3 Variable Karnaugh Map

G(W, X ,Y, Z)=Σm(1,2,3,5,6,7,9,10,11,14,15)

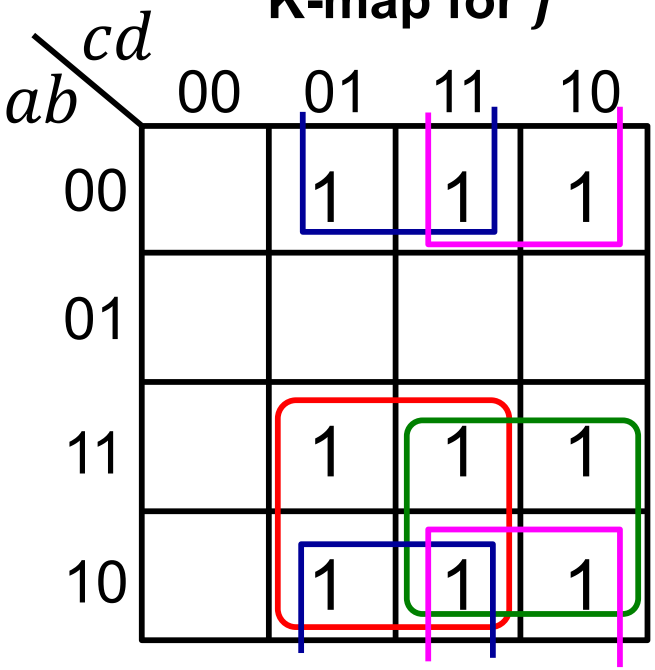
YZ \ WX	00	01	11	10
00	0 0	4 0	12 0	8 0
01	1 1	5 1	13 0	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 1	14 1	10 1

4 Variable Karnaugh Map

Product-of-Sums Minimization – Method 1

- Construct the K-map for f' to find a minimal SOP expression.
  - You can do this by simply replacing the 0's in the K-map of f with 1's in the K-map of f'.
  - Complement ($f=(f')'$) the minimal SOP expression of f' and apply DeMorgan's theorem to find a minimal POS expression for f.
- Example: $f(a,b,c,d) = \sum m(1,2,3,9,10,11,13,14,15)$

K-map for f

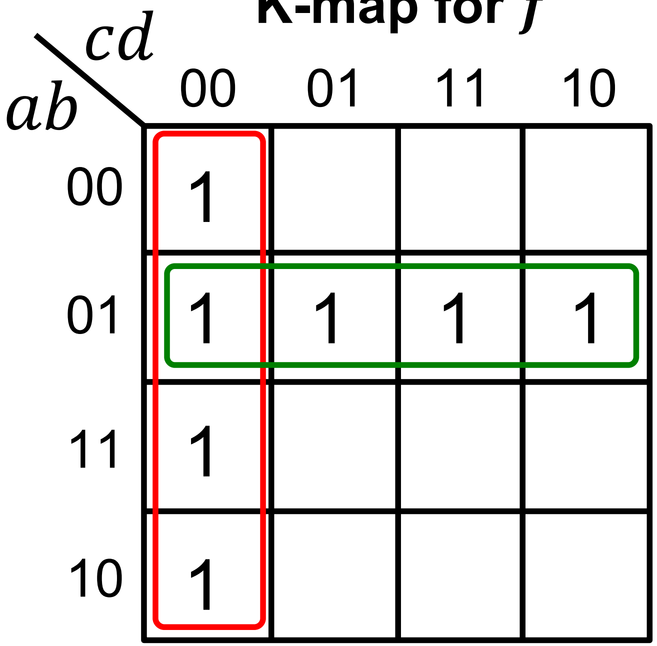


$ab \backslash cd$	00	01	11	10
00		1	1	1
01				
11		1	1	1
10		1	1	1

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

Minimal SOP (8 literals)

K-map for f'



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

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

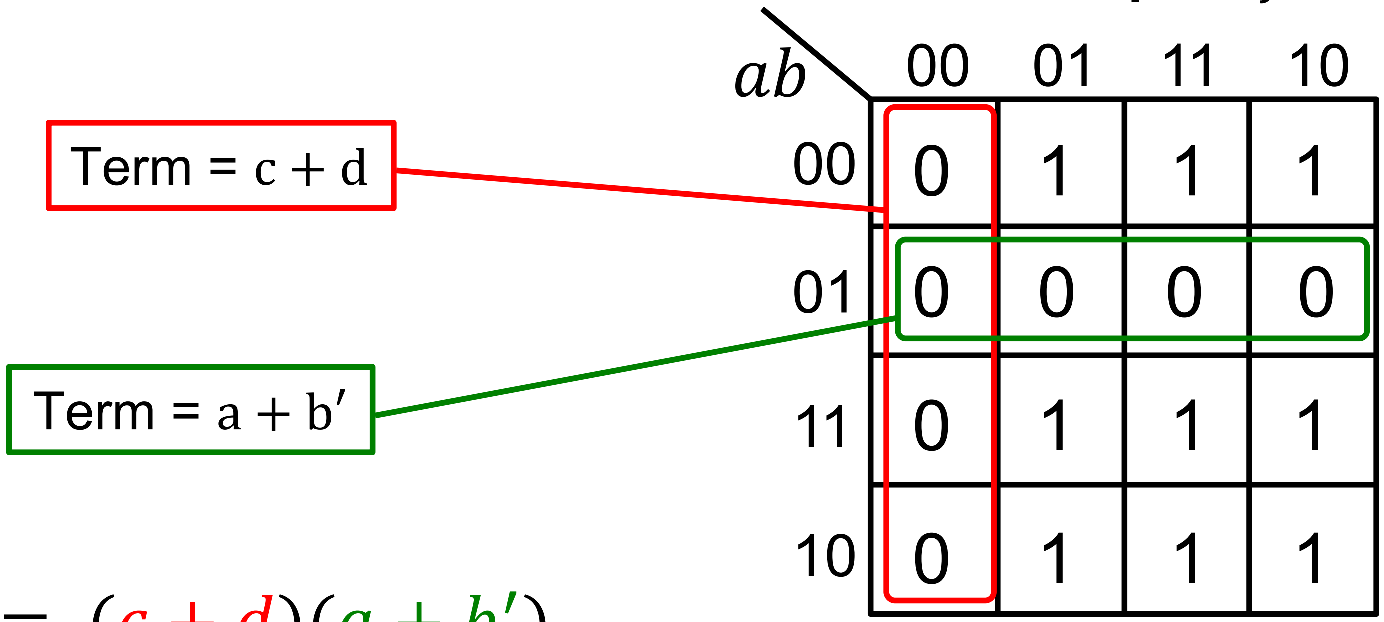
$f = (f')' = (c+d)(a+b')$

Minimal POS (4 literals)

Product-of-Sums Minimization – Method 2

- Construct the K-map for f and find a minimal POS expression by:
  - Grouping 0s as you did with the 1's for SOP.
  - Replacing all the maxterms represented by the squares in the group by a single expression formed by the OR of the complement of the variables which remain constant every square of the grouping. Thus, all those variables changing in the grouping are eliminated.
- Example: $f(a,b,c,d) = \sum m(1,2,3,9,10,11,13,14,15)$

K-map for f



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

Term = $\mathrm{c} + \mathrm{d}$

Term = $\mathrm{a} + \mathrm{b}'$

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

Minimal POS (4 literals)

POS or SOP Minimization?

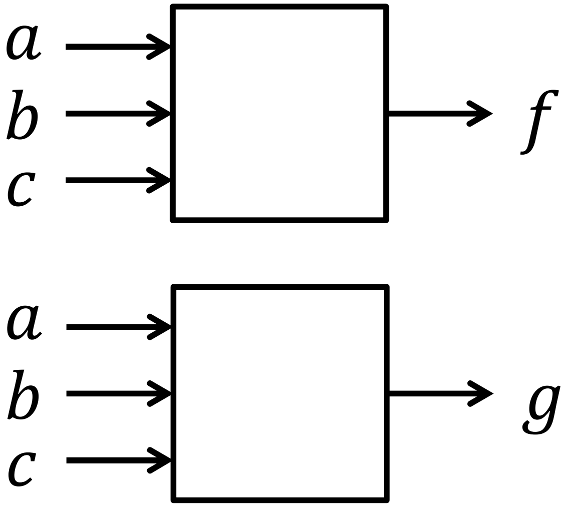
- In general, reductions in both forms should be attempted since one form may lead to fewer gates and/or fewer inputs.

- “Don’t care” states can be grouped with either 1’s to obtain a simplified sum-of-products, or grouped with 0’s to obtain a minimum product-of-sums.

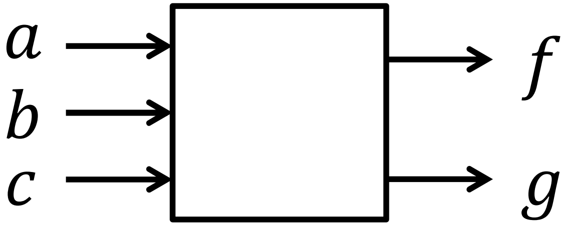
Multiple Outputs

Multiple Outputs

- Consider the 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: $f(a,b,c) = \sum \mathrm{m}(0,2,6,7)$ and $g(a,b,c) = \sum \mathrm{m}(1,3,6,7)$
- 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

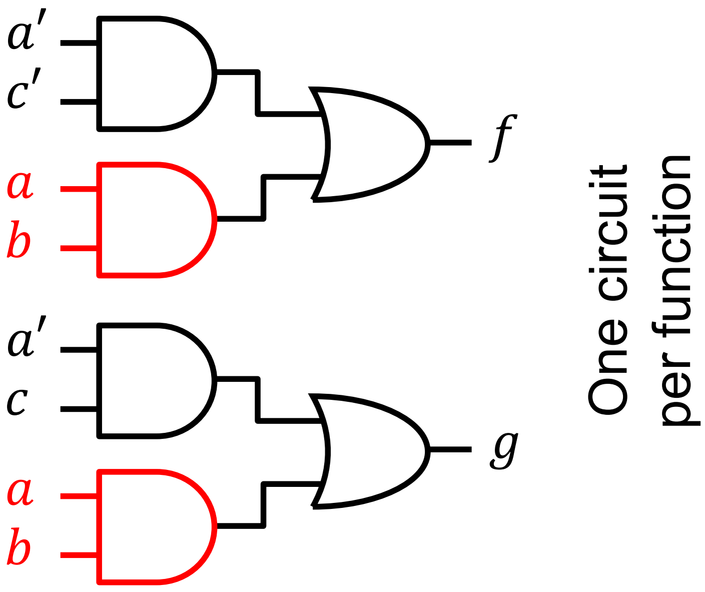
$$f = a'c' + ab$$

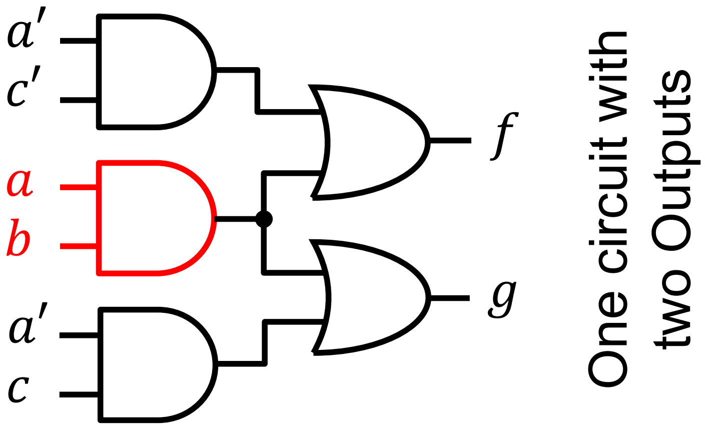
K-Map of *g*

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

$$g = a'c + ab$$

Common Term = ab





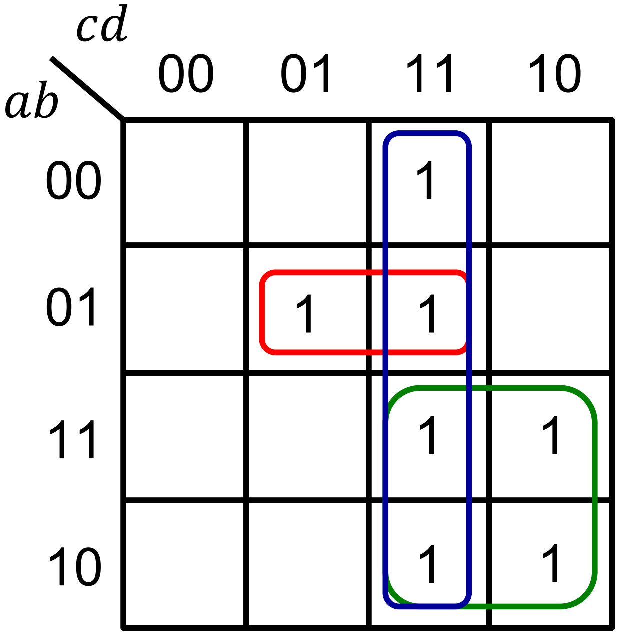
Multiple Outputs – Example 2

- $f(a,b,c,d) = \sum \mathrm{m}(3,5,7,10,11,14,15)$, $g(a,b,c,d) = \sum \mathrm{m}(1,3,5,7,10,14)$
- Draw the K-map and write minimal SOP expressions of f and g

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

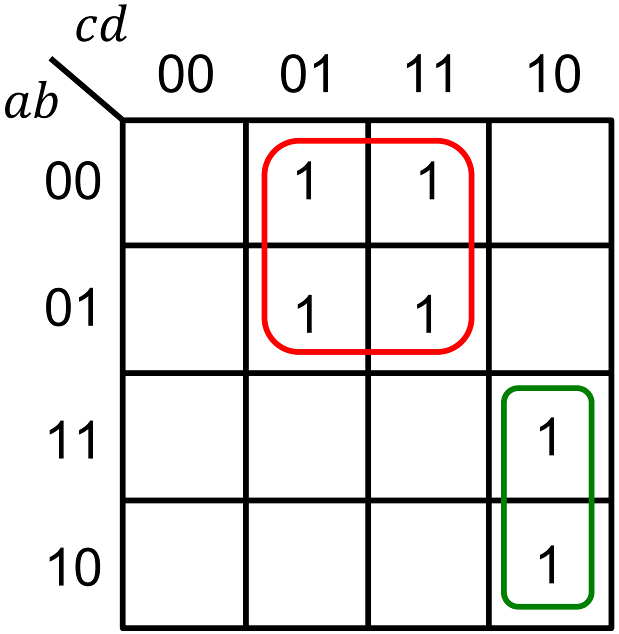
- Extract the common terms of f and g

K-Map of *f*



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

K-Map of *g*



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

Common Terms

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

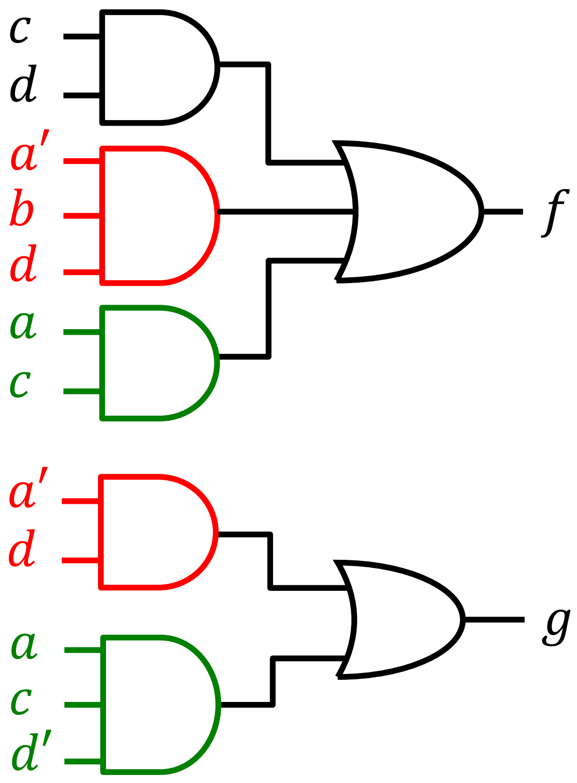
Minimal f and g

$f = T_1 b + T_2 + cd$

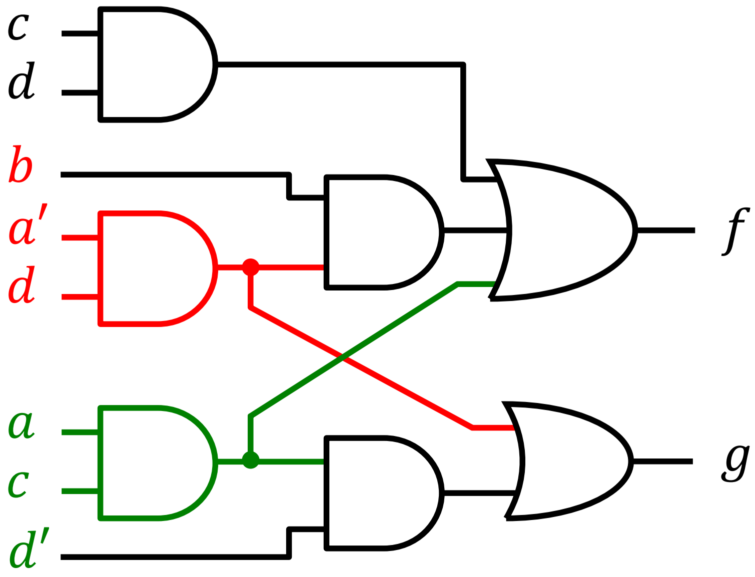
$g = T_1 + T_2 d'$

Multiple Outputs – Example 2

- 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

Five and six-variables K-maps

K-maps

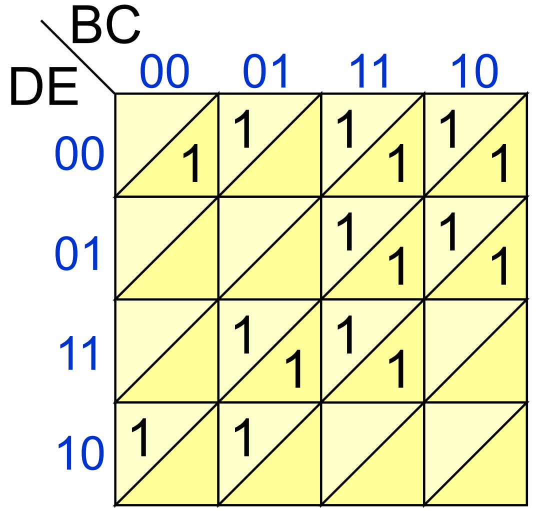
- The K-map method is not limited to four variables or less

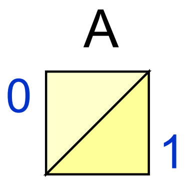
- However, using it to visualize functions with more than four variables becomes more challenging as does trying to map a 5- or 6-dimensional cube to a 2-dimensional paper.

- It is important to remember that within an *n*-variable map we must check for adjacencies in *n* directions.

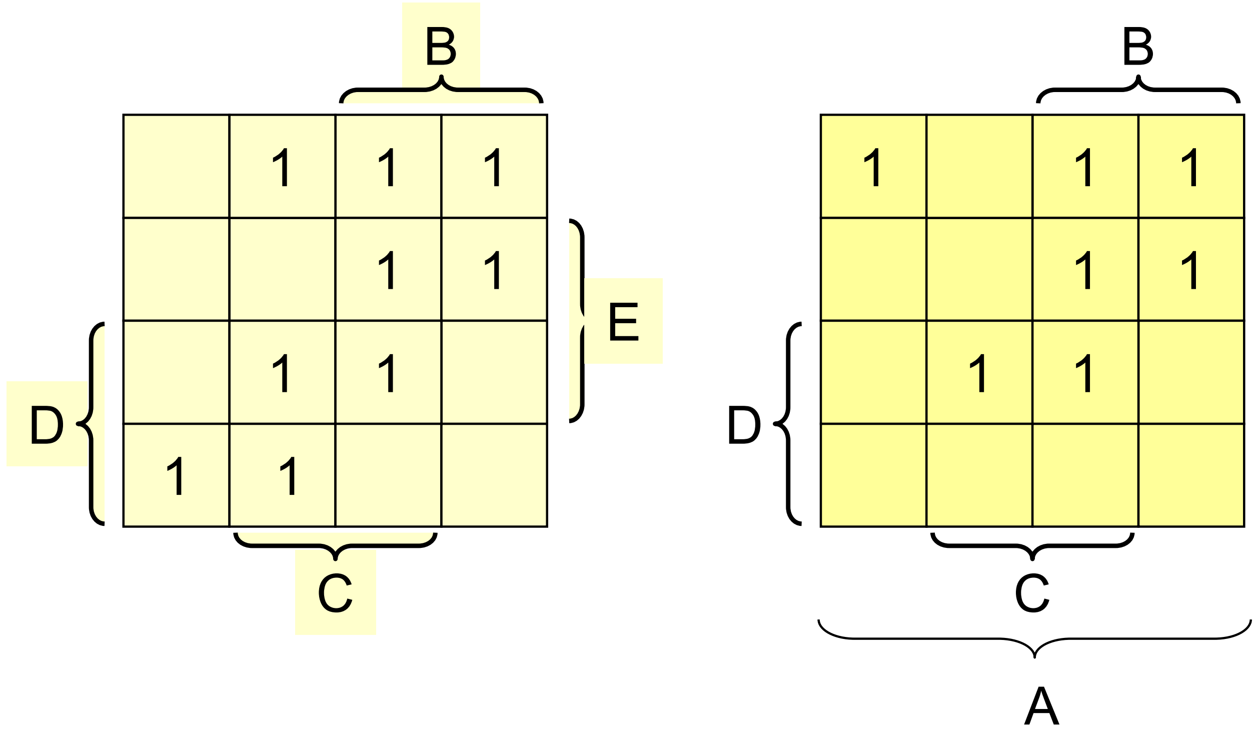
- We can handle the adjacencies for up to six variables.

5-Variable K-maps





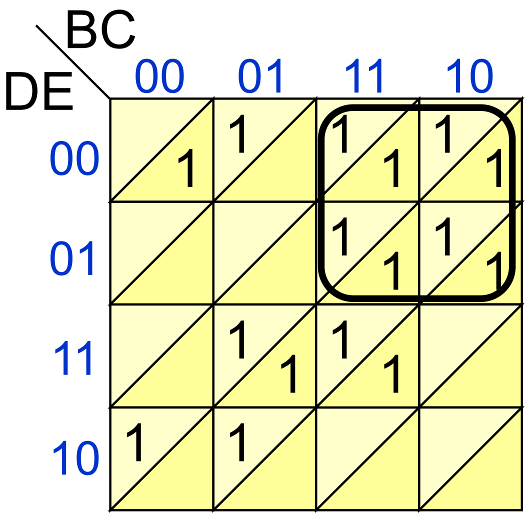
F =



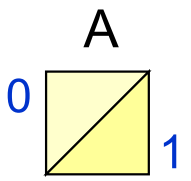
	1	1	1
		1	1
	1	1	
1	1		

1		1	1
		1	1
	1	1	

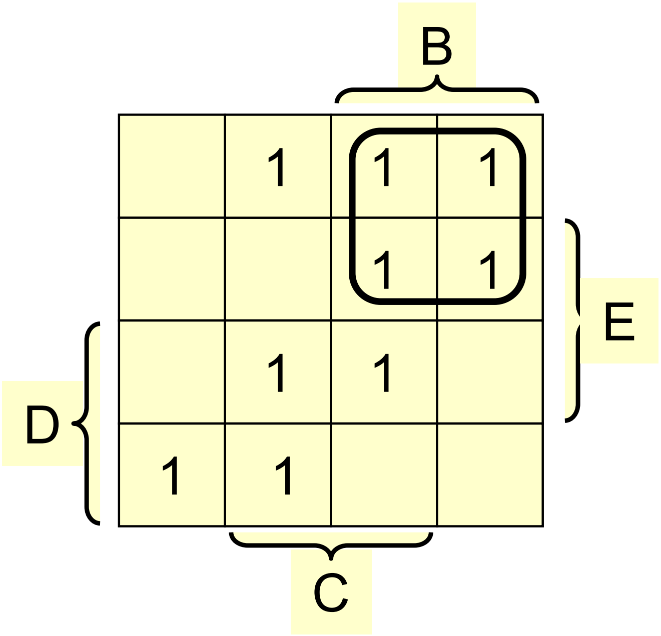
5-Variable K-maps



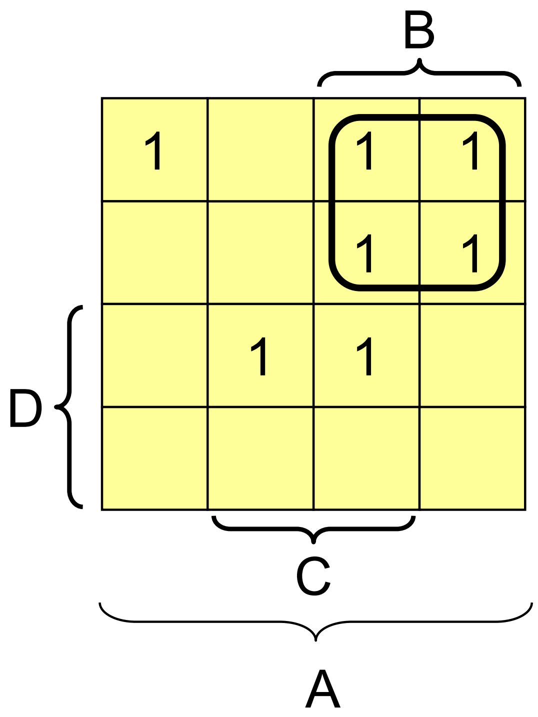
DE \ BC	00	01	11	10
00	 1	1 	1 1	1 1
01			1 1	1 1
11		1 1	1 1	
10	1 	1 		



F = BD’

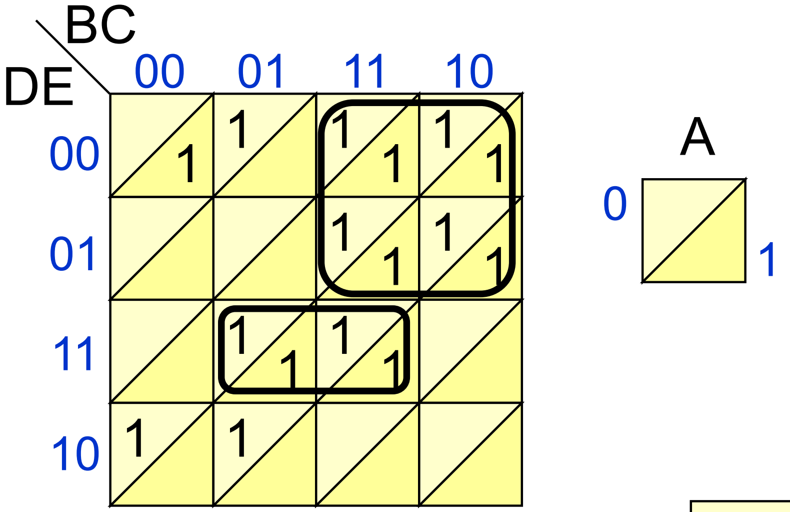


	1	1	1
		1	1
	1	1	
1	1		

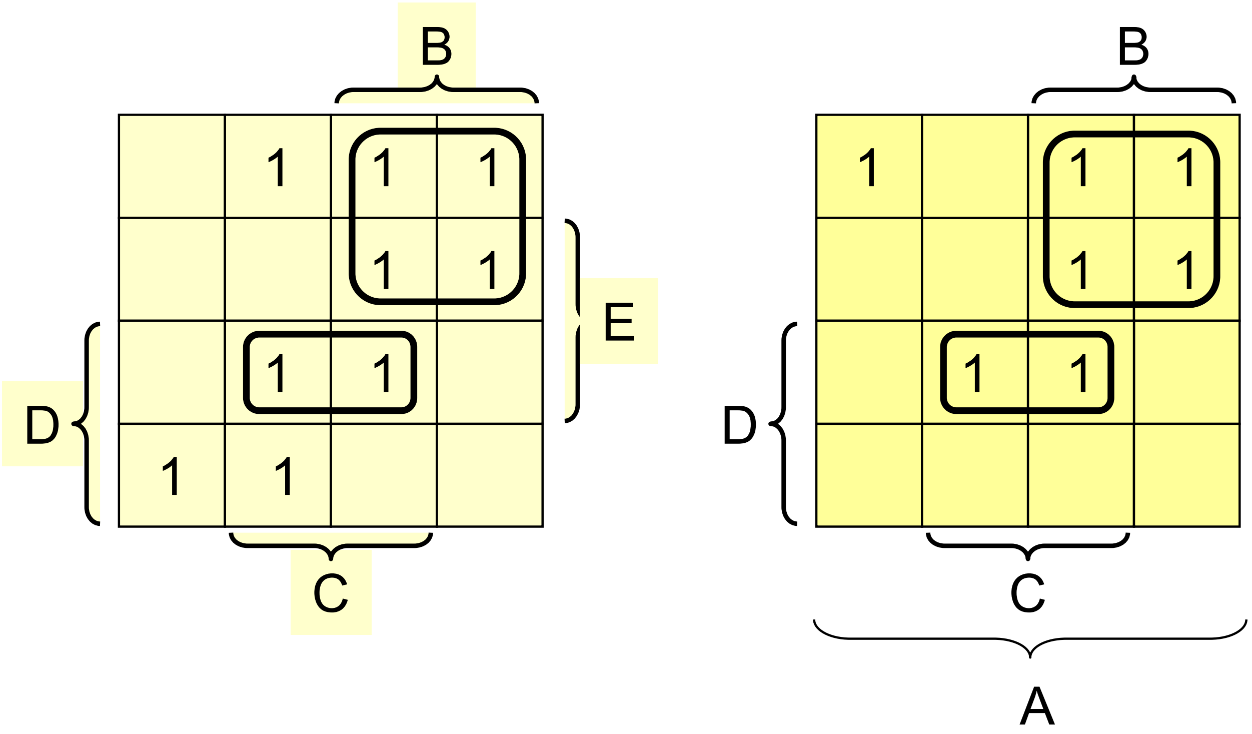


1		1	1
		1	1
	1	1	

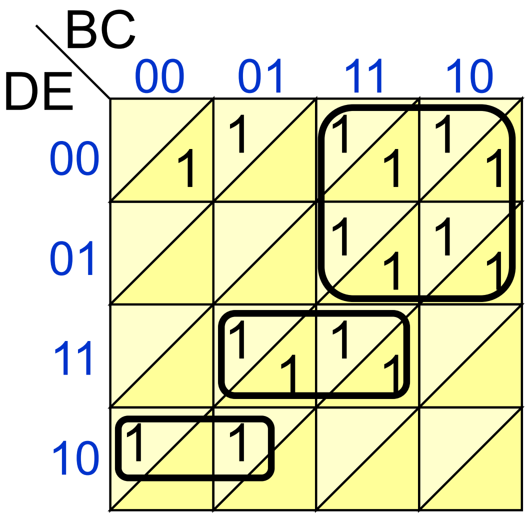
5-Variable K-maps

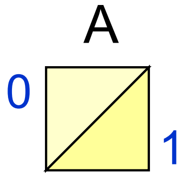


F = BD’ + CDE

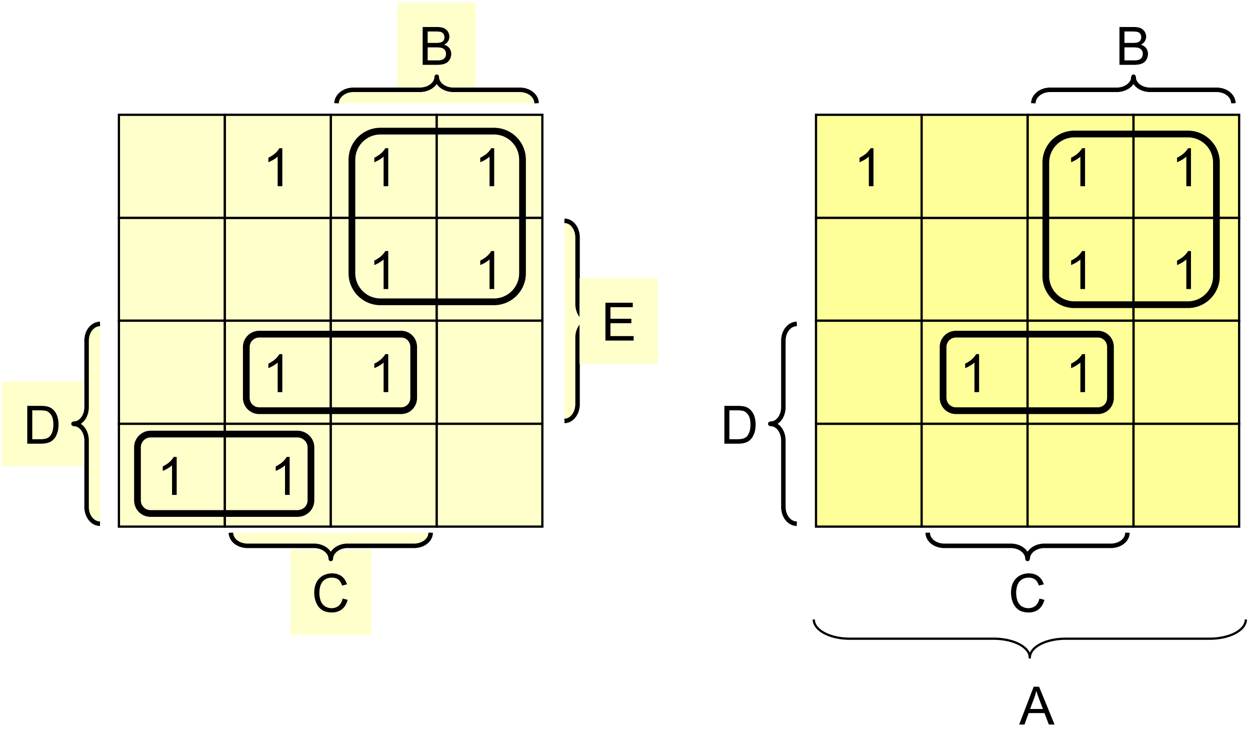


5-Variable K-maps

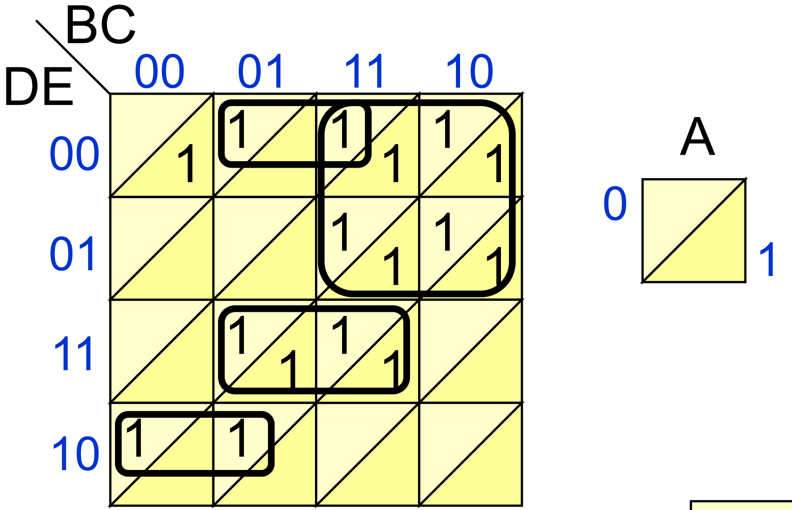




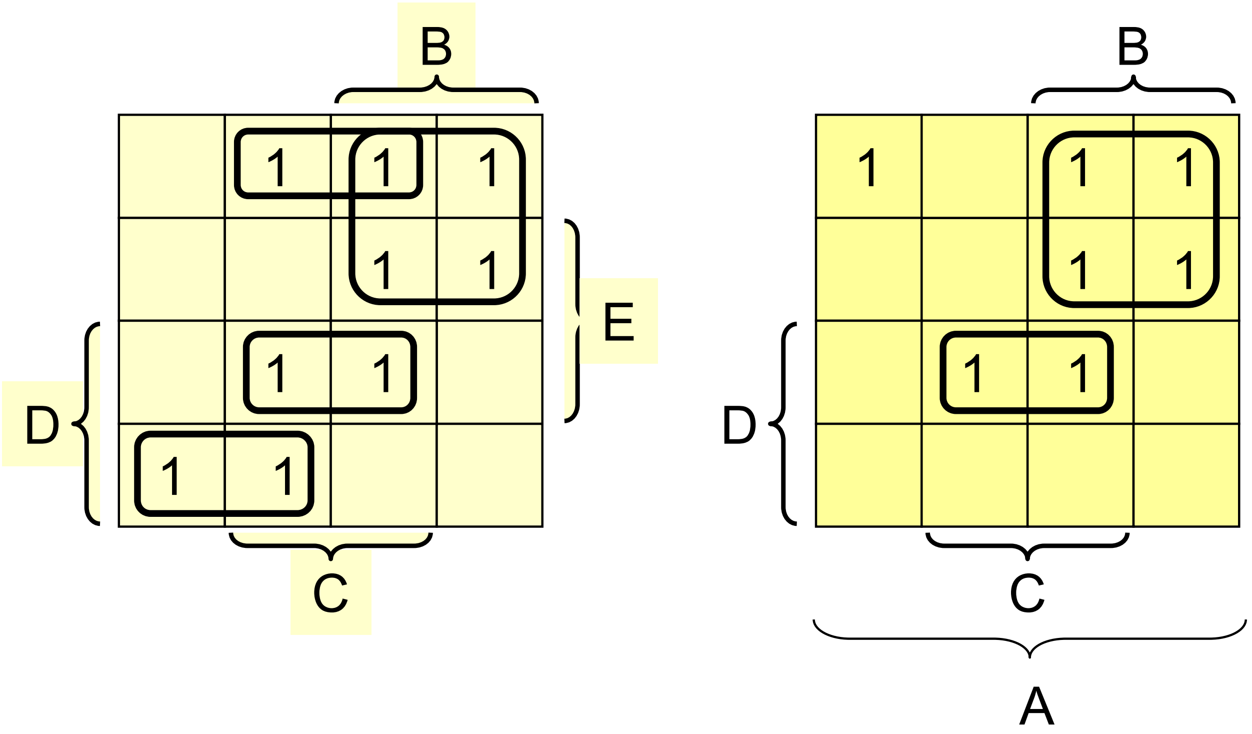
F = BD’ + CDE + A’B’DE’



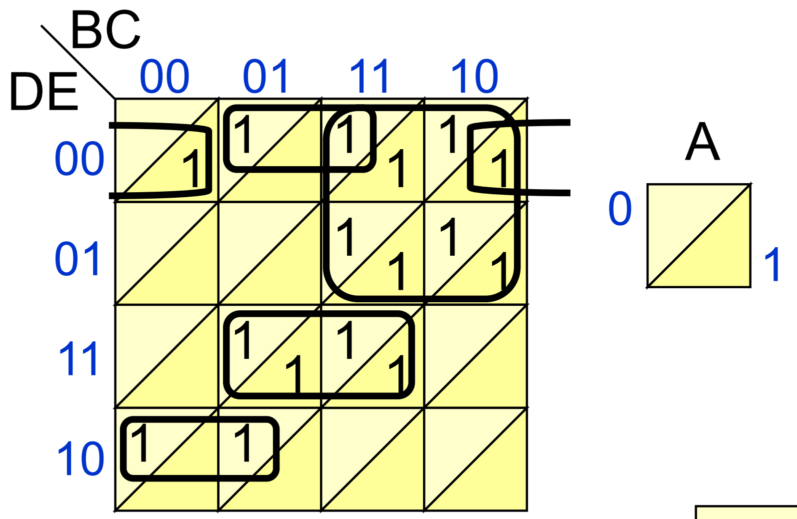
5-Variable K-maps



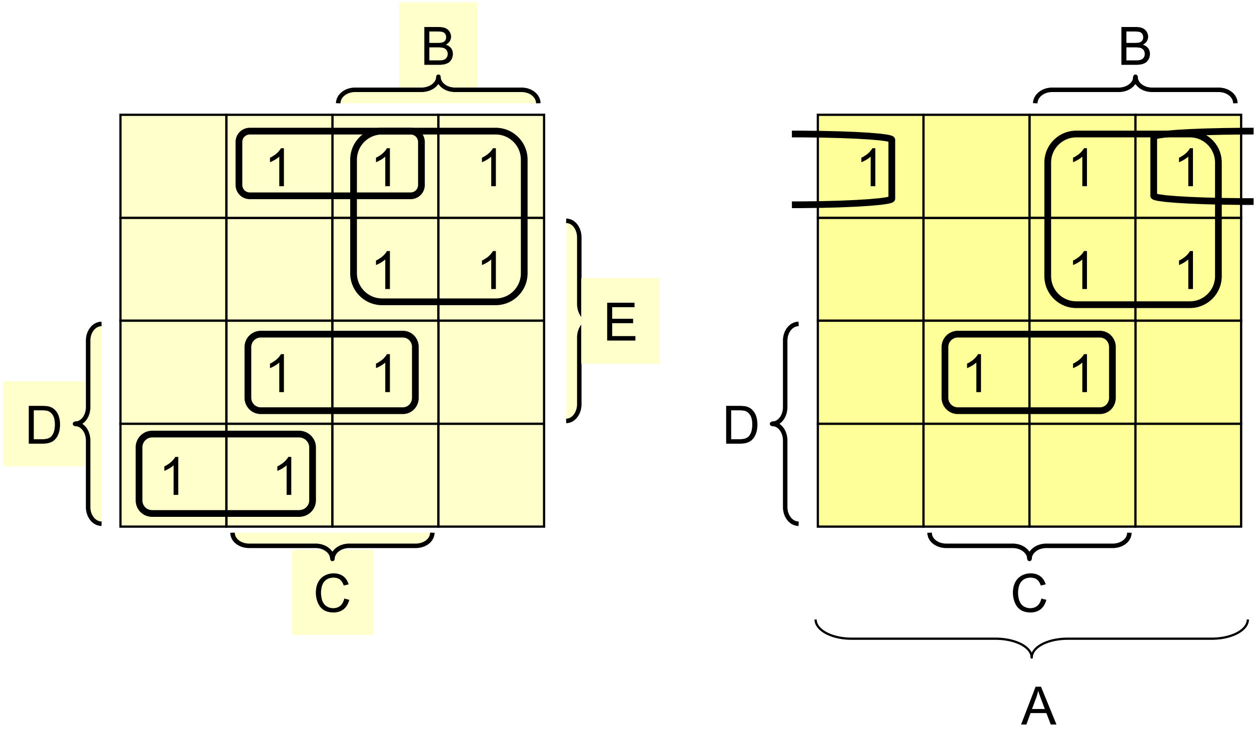
F = BD’ + CDE + A’B’DE’ + A’CD’E’



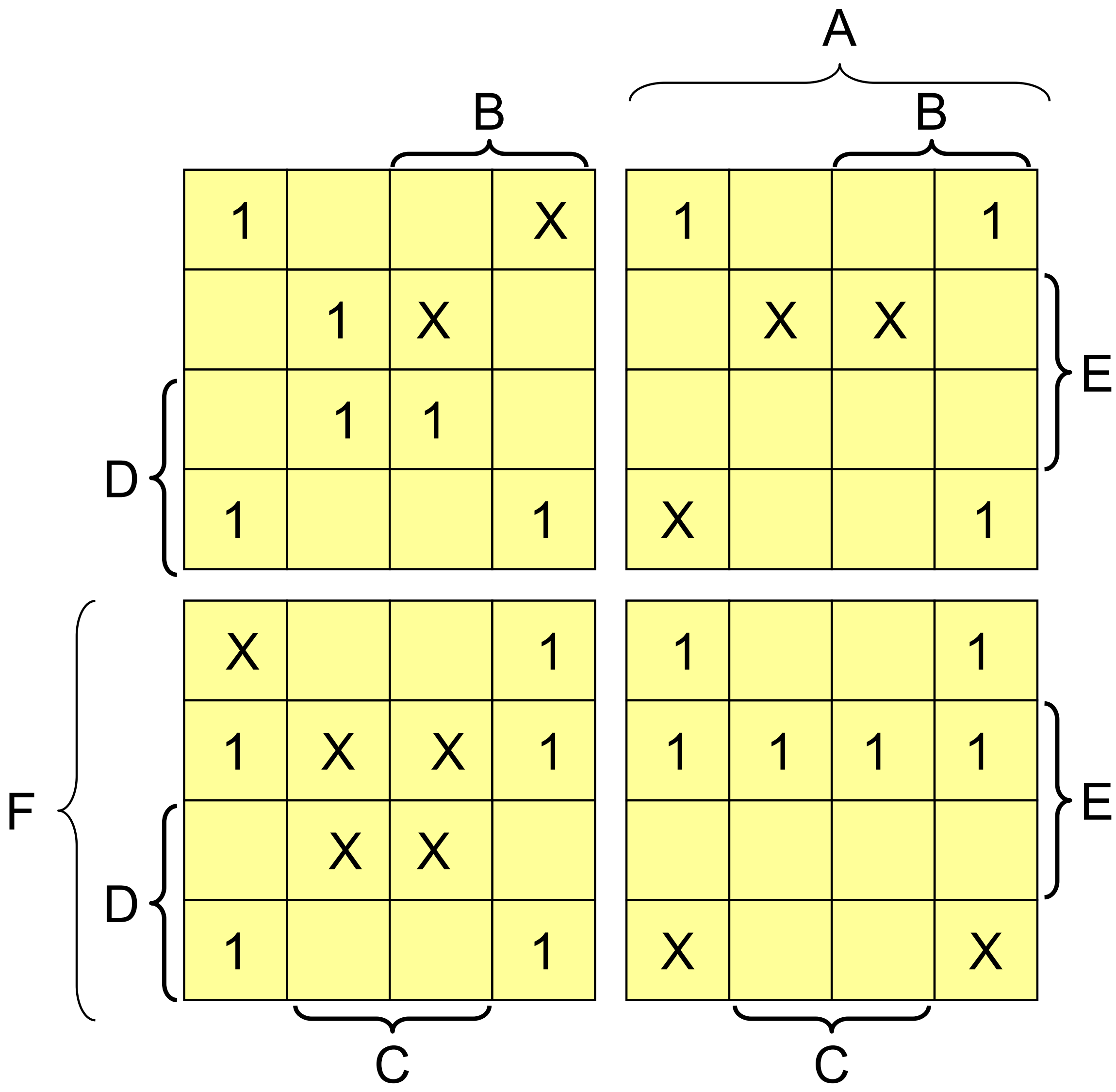
5-Variable K-maps



F = BD’ + CDE + A’B’DE’ + A’CD’E’ + AC’D’E’



6-Variable K-maps



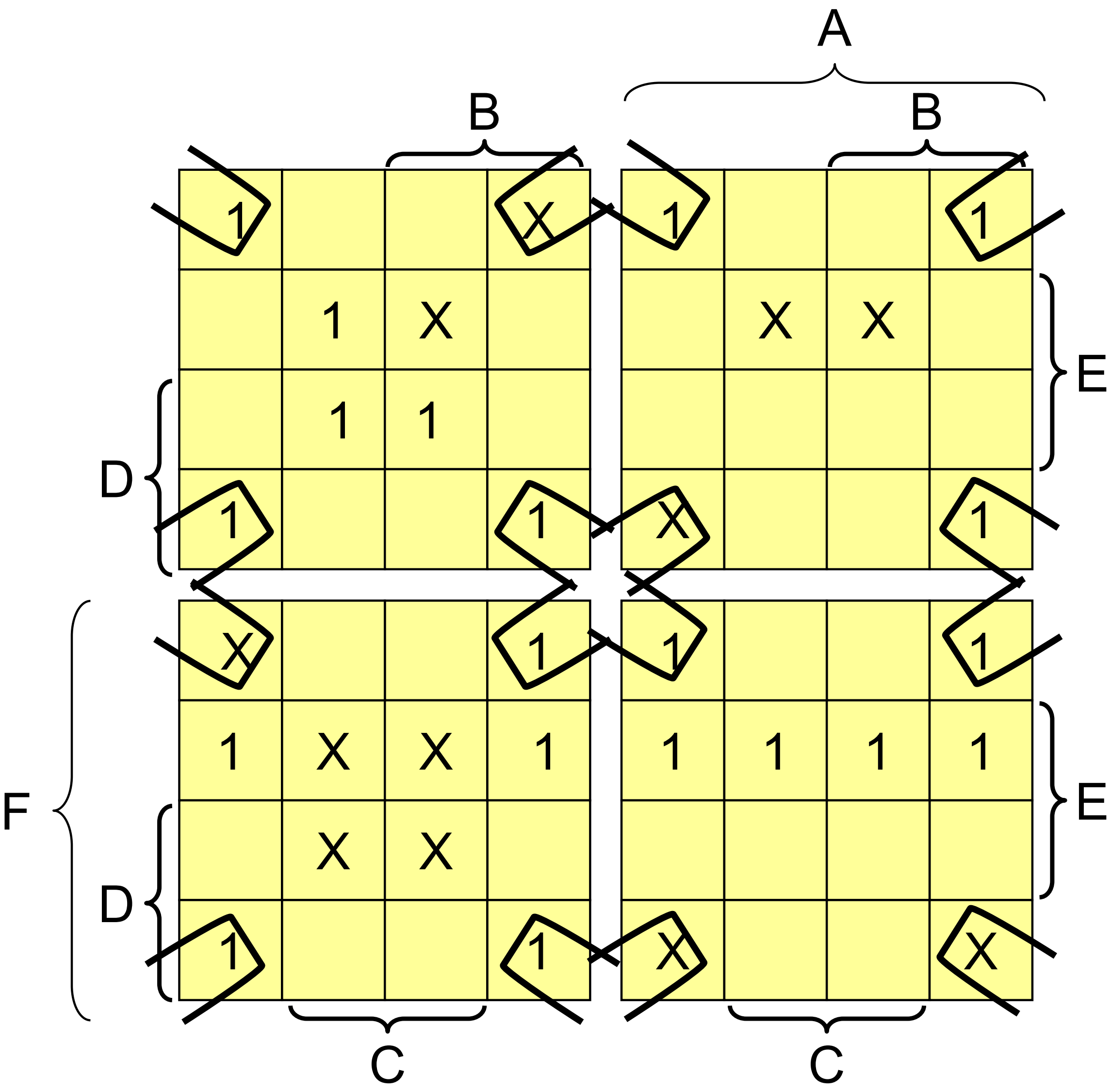
1			X
	1	X	
	1	1	
1			1

1			1
	X	X	
X			1

X			1
1	X	X	1
	X	X	
1			1

1			1
1	1	1	1
X			X

6-Variable K-maps



1			X
	1	X	
	1	1	
1			1

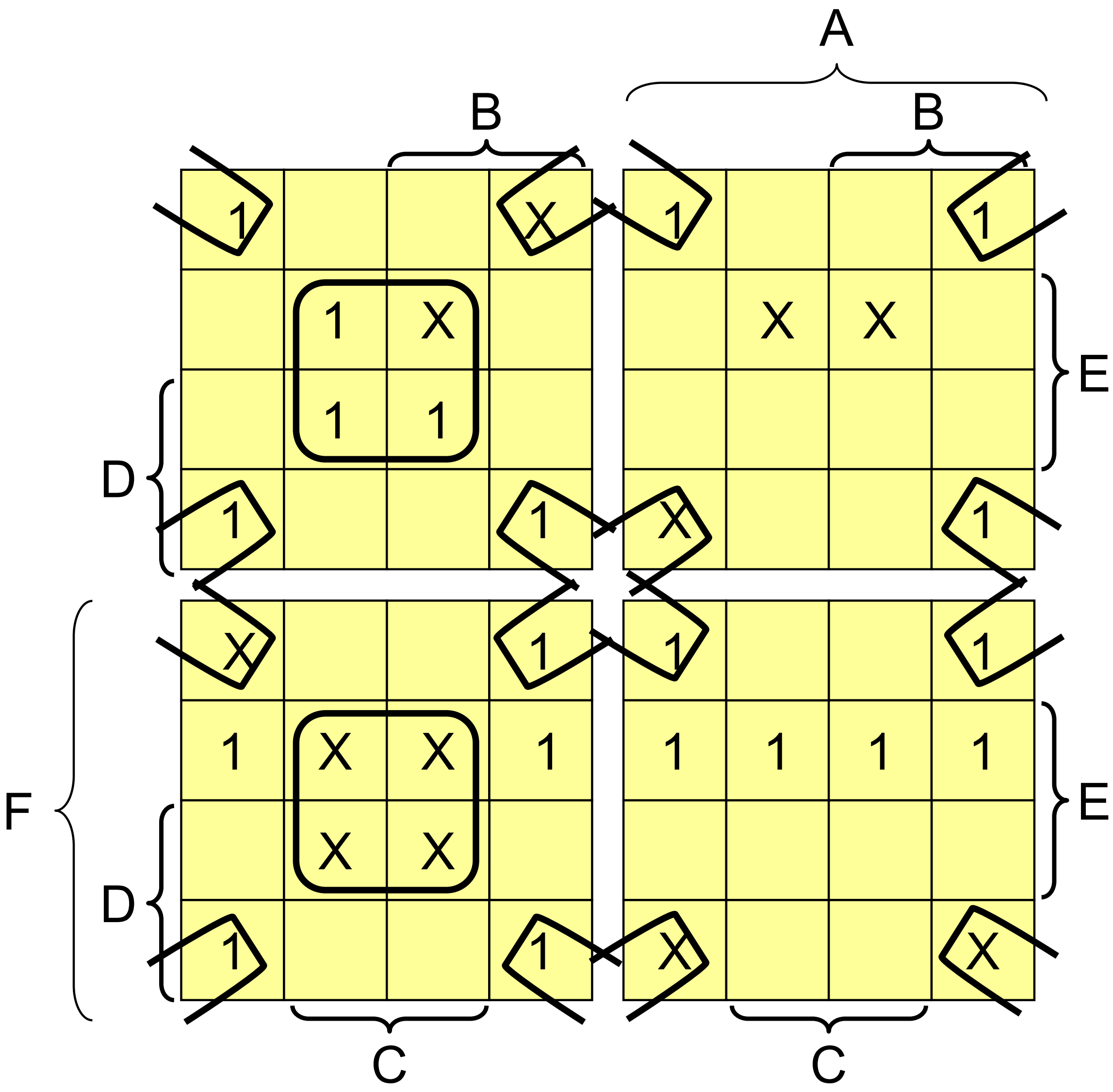
1			1
	X	X	
X			1

X			1
1	X	X	1
	X	X	
1			1

1			1
1	1	1	1
X			X

g = C’E’

6-Variable K-maps



1			X
	1	X	
	1	1	
1			1

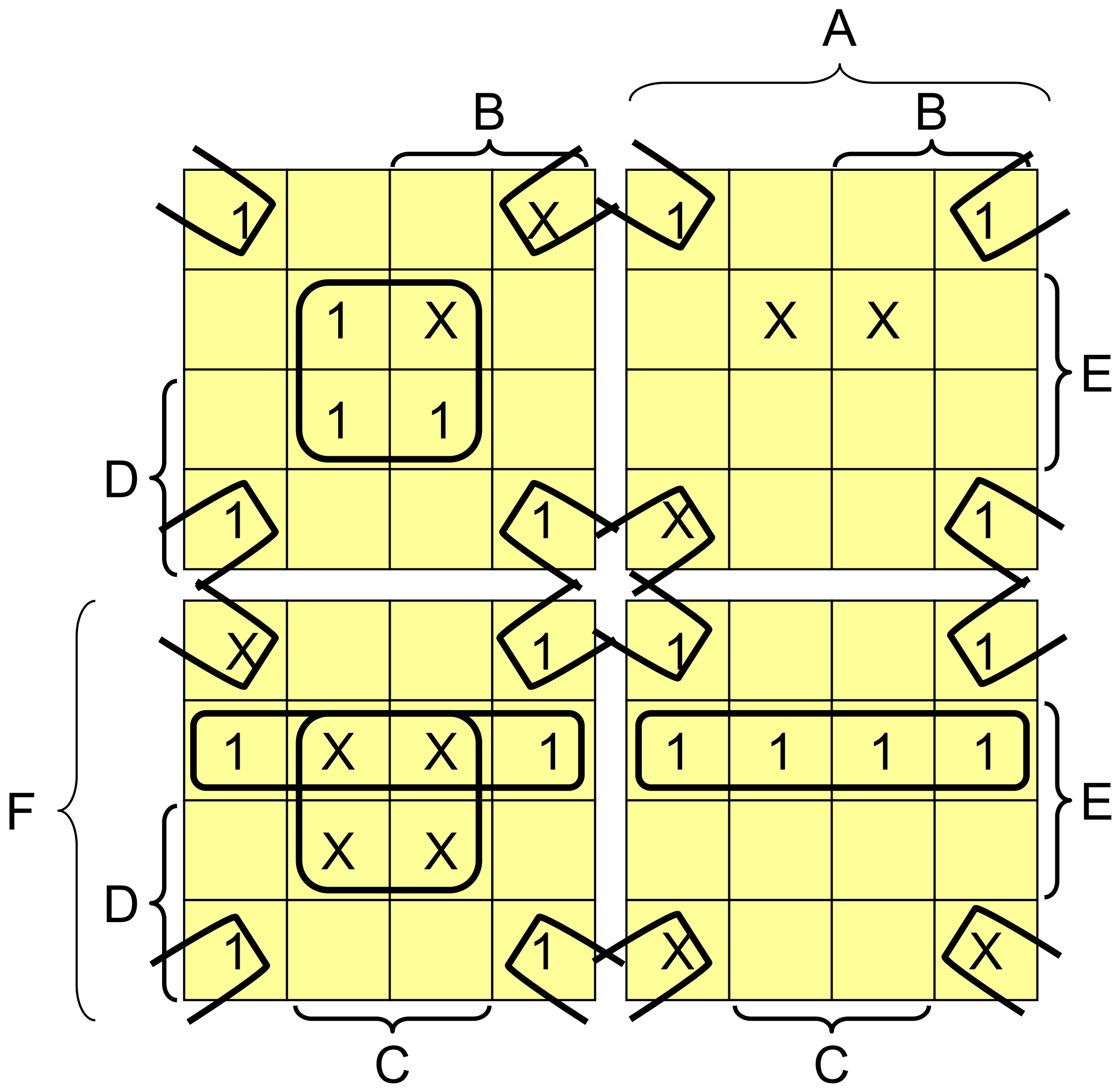
1			1
	X	X	
X			1

X			1
1	X	X	1
	X	X	
1			1

1			1
1	1	1	1
X			X

g = C'E' + A'CE

6-Variable K-maps



1			X	1			1
	1	X			X	X	
	1	1					
1			1	X			1
X			1	1			1
1	X	X	1	1	1	1	1
	X	X					
1			1	X			X

g = C’E’ + A’CE + D’EF

Automating two-level simplification

- K-maps depend on our ability to visually identify prime implicants and select a set of prime implicants that cover all minterms.

- They do not provide a direct algorithm to be implemented in a computer.

- For larger systems, we need a programmable method (e.g. Quine-McCluskey method) to provide a systematic method for generating all prime implicants and then extracting a minimum set of primes covering the on-set.

Acknowledgments

- Credit is acknowledged where credit is due. Please refer to the full list of references.