Combinational Logic Circuits
Part III -Theoretical Foundations

Overview

- Simplifying Boolean Functions
  - Algebraic Manipulation
  - Karnaugh Map Manipulation (simplifying functions of 2, 3, 4 variables)
- Systematic Approach for Simplifying Functions using K-maps
  - Implicants, Prime Implicants (PIs), and Essential Prime Implicants
  - Simplifying Functions using Essential and Nonessential PIs
- Don’t-care Conditions and Simplification using Don’t Cares

Boolean Functions as Equations

- Truth table and K-map of a Boolean function are unique representations
- However, representing a Boolean function as an equation can be done in many different ways
  - Canonical and Standard forms
- Example:
  - F1(X,Y,Z) = X'•Y'•Z' + X'•Y•Z' + X•Y•Z'
  - F2(X,Y,Z) = X'•Y'•Z' + Y•Z'
  - F3(X,Y,Z) = X'•Z' + X•Y•Z'
  - F4(X,Y,Z) = X'•Z' + Y•Z'
- The corresponding truth tables for F1 to F4 are identical!
- Thus, F1 = F2 = F3 = F4
- However, F2 and F3 are simpler than F1 and F4 is simpler than the others.

X	Y	Z		F1	F2	F3	F4
0	0	0		**1**	**1**	**1**	**1**
0	0	1		0	0	0	0
0	1	0		**1**	**1**	**1**	**1**
0	1	1		0	0	0	0
1	0	0		0	0	0	0
1	0	1		0	0	0	0
1	1	0		**1**	**1**	**1**	**1**
1	1	1		0	0	0	0

How do we simplify Boolean functions?

Simplifying a Boolean Function

- Why simplifying Boolean functions?
  - Boolean functions are used to design digital logic circuits
  - Simpler Boolean function can mean cheaper, smaller, faster circuit
- Three main approaches to simplify Boolean functions:
  - Algebraic Manipulations
    - using the Boolean Algebra as a tool for simplifications
  - Karnaugh Map Manipulations
    - very easy graphical method to simplify Boolean functions
    - it works for functions of up to 4 variables!
  - Algorithmic Techniques
    - used to program a computer to do the simplifications

Algebraic Manipulation

- We use basic identities, properties, and theorems of the Boolean Algebra to manipulate and simplify Boolean functions
- <u>Example1:</u> Simplify F = X'YZ + X'YZ' + XZ

  F = X'YZ + X'YZ' + XZ -- apply identity 14
  = X'Y(Z+Z') + XZ -- apply identity 7
  = X'Y•1 + XZ -- apply identity 2
  = X'Y + XZ
- <u>Example2:</u> Simplify G = X'Y'Z' + X'YZ' + XYZ'

  F = X'Y'Z' + X'YZ' + XYZ' -- apply identity 5
  = X'Y'Z' + X'YZ' + X'YZ' + XYZ' -- apply identity 14
  = X'Z'(Y'+Y) + YZ'(X'+X) -- apply identity 7
  = X'Z'•1 + YZ'•1 -- apply identity 2
  = X'Z' + YZ'

Karnaugh Map Manipulations

- We can use a K-map to simplify a Boolean function of 2, 3, or 4 variables as Sum-Of-Products
- Procedure:
  - Enter 1s in the K-map for each minterm (product term) in the function
  - Group *adjacent* K-map cells containing 1s to obtain a product term with fewer variables
    - The number of cells in a group must be a power of 2 (2, 4, 8, …)!
  - Try to group as many as possible cells containing 1s in a group
    - Such group corresponds to a simpler product term!
  - Try to make as less as possible groups to cover all cells containing 1s
    - This corresponds to fewer product terms in the simplified function!
  - Do not forget to handle boundary cells for K-maps of 3 or 4 variables when you do the grouping
  - Important: The result after the simplification may not be unique!

Simplifying a Boolean Function using 2-variable K-map (examples)

Given functions:

Simplified functions:

$F1(X,Y) = \Sigma m(0,1) =$
$= X'Y' + X'Y$

X \ Y	0	1
0	1	1
1		

$F1(X,Y) = X'$

$F2(X,Y) = \Sigma m(0,3) =$
$= X'Y' + XY$

X \ Y	0	1
0	1	
1		1

$F2(X,Y) = X'Y' + XY$

$F3(X,Y) = \Sigma m(0,2,3) =$
$= X'Y' + XY' + XY$

X \ Y	0	1
0	1	
1	1	1

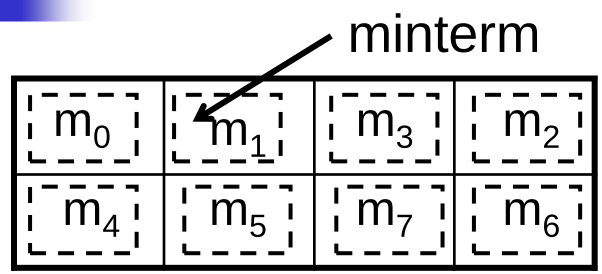
$F3(X,Y) = X + Y'$

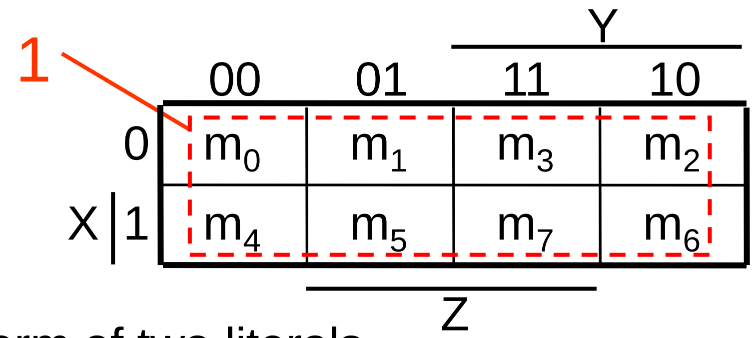
$F4(X,Y) = \Sigma m(0,1,2,3) =$
$= X'Y' + X'Y + XY' + XY$

X \ Y	0	1
0	1	1
1	1	1

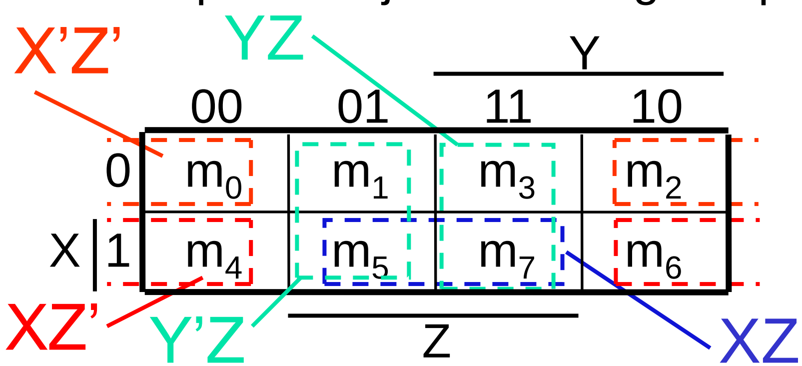
$F4(X,Y) = 1$

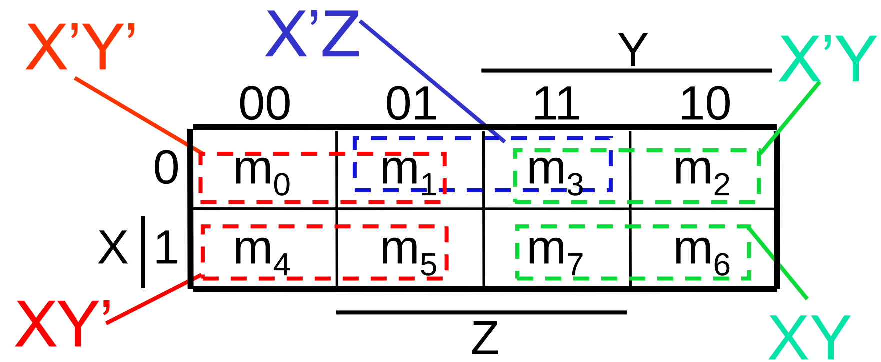
Simplifying a Boolean Function using 3-variable K-map (groupings)



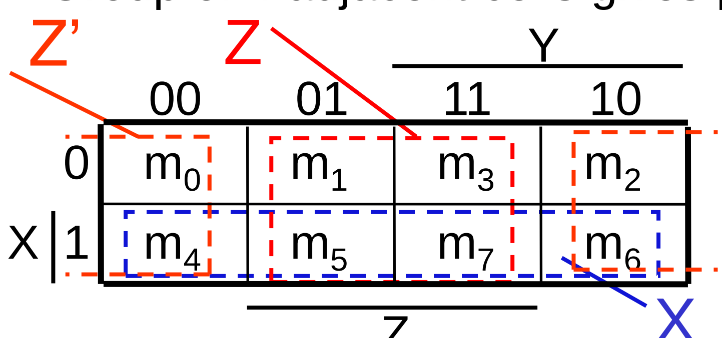


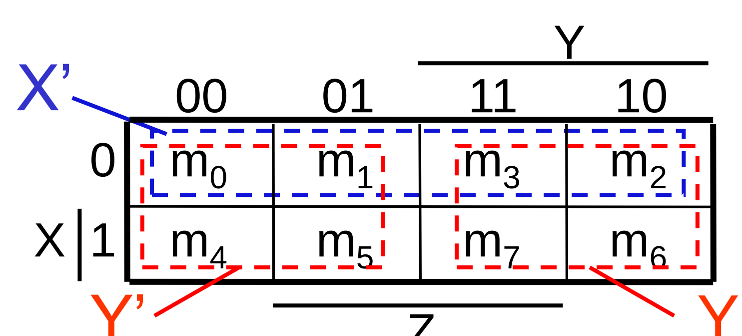
- Group of 2 adjacent cells gives product term of two literals.





- Group of 4 adjacent cells gives product term of one literal.

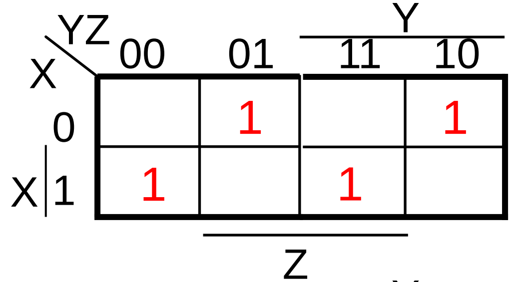




Simplifying a Boolean Function using 3-variable K-map (examples)

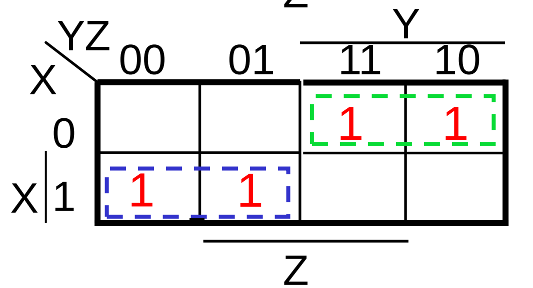
Given functions: | **Simplified functions:**

F1(X,Y,Z) = Σm(1,2,4,7)



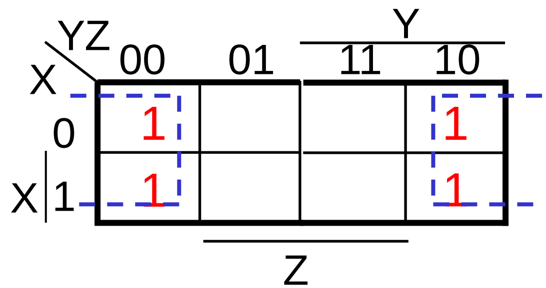
Simplification is not possible

F2(X,Y,Z) = Σm(2,3,4,5)



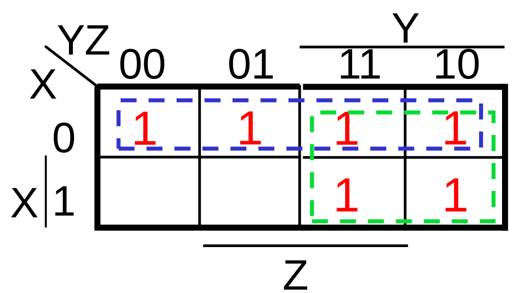
F2(X,Y,Z) = XY’ + X’Y

F3(X,Y,Z) = Σm(0,2,4,6)



F3(X,Y,Z) = Z’

F4(X,Y,Z) = Σm(0,1,2,3,6,7)



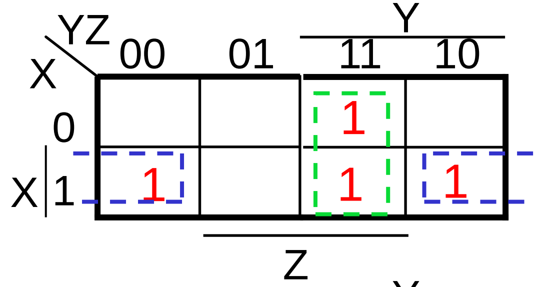
F4(X,Y,Z) = X’ + Y

Simplifying a Boolean Function using 3-variable K-map (more examples)

Given functions:

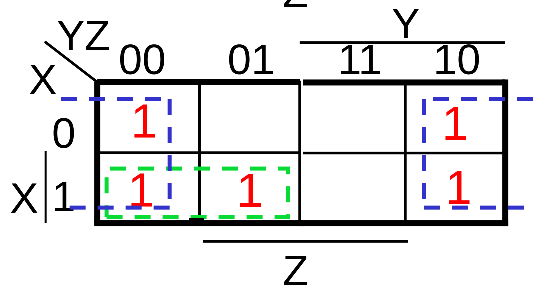
Simplified functions:

F5(X,Y,Z) = Σm(3,4,6,7)



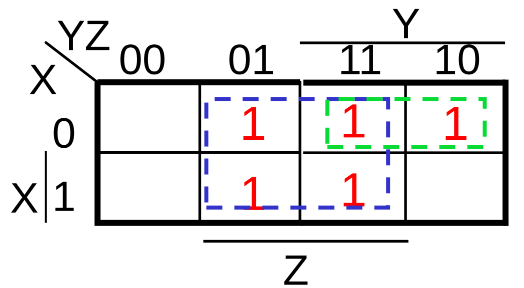
F5(X,Y,Z) = XZ' + YZ

F6(X,Y,Z) = Σm(0,2,4,5,6)



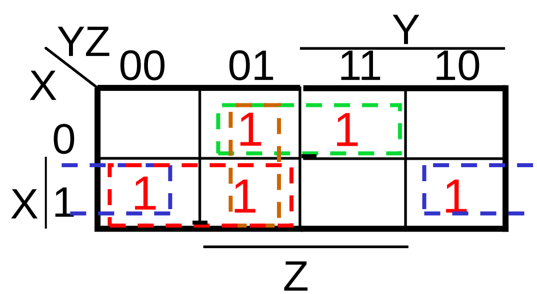
F6(X,Y,Z) = Z' + XY'

F7(X,Y,Z) = Σm(1,2,3,5,7)



F7(X,Y,Z) = Z + X'Y

F8(X,Y,Z) = Σm(1,3,4,5,6)



F8(X,Y,Z) = XZ'+X'Z +Y'Z

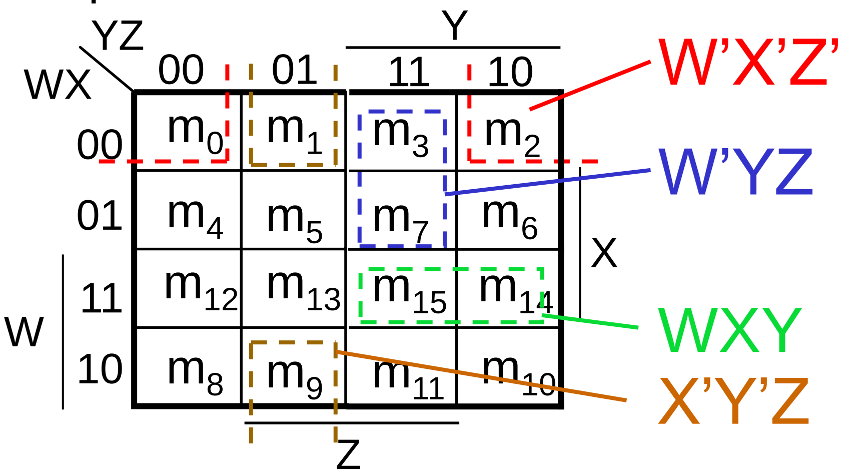
or

F8(X,Y,Z) = XZ'+X'Z +XY'

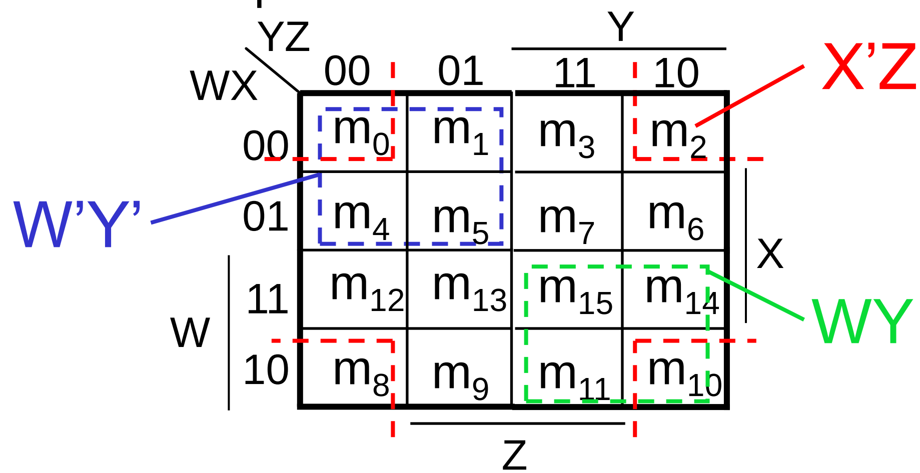
Not unique solution

Simplifying a Boolean Function using 4-variable K-map (grouping examples)

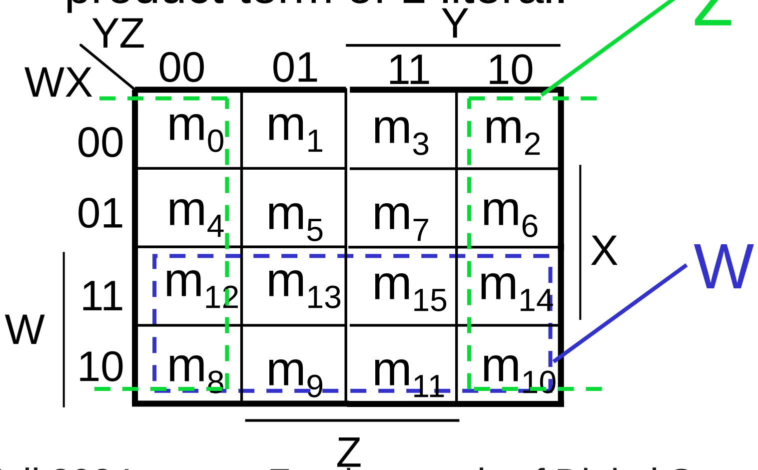
- Group of 2 adjacent cells gives product term of 3 literals.



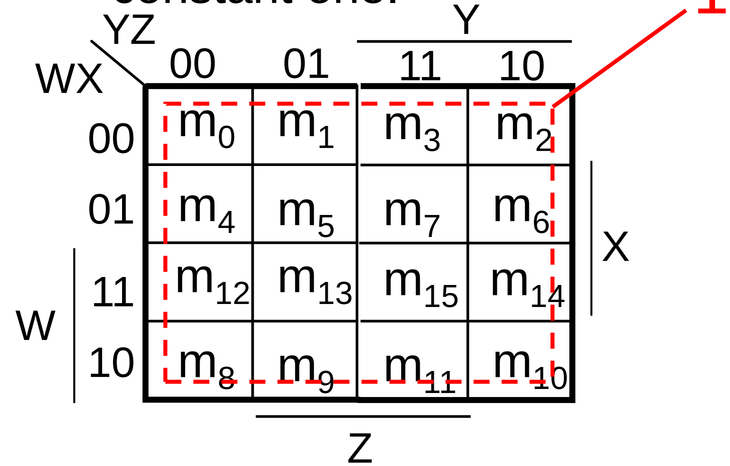
- Group of 4 adjacent cells gives product term of 2 literals.



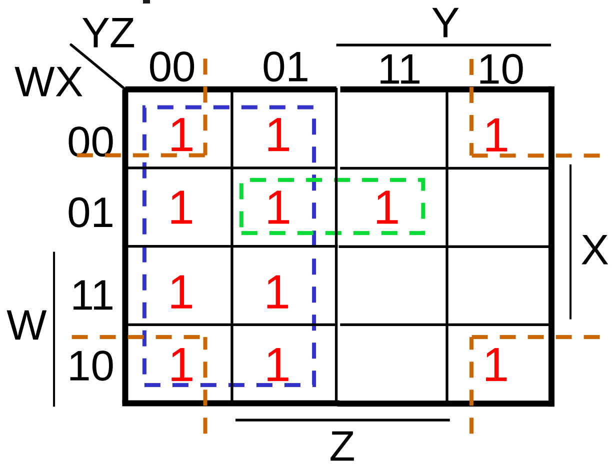
- Group of 8 adjacent cells gives product term of 1 literal.



- Group of all cells gives constant one.



Simplifying a Boolean Function using 4-variable K-map (examples)



Given function:

F1(W,X,Y,Z) =
= Σm(0,1,2,4,5,7,8,9,10,12,13)

Simplified function:

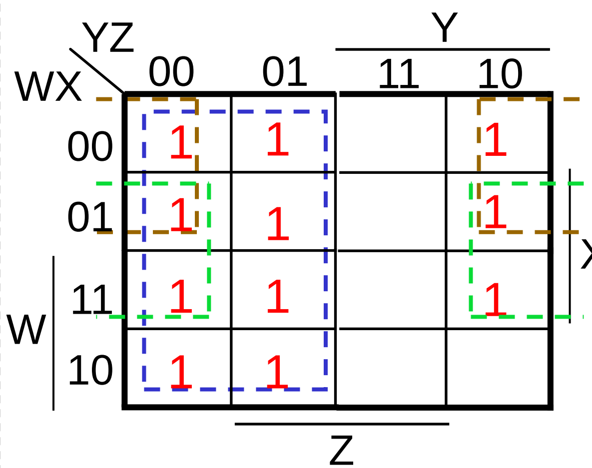
F1(W,X,Y,Z) =
= Y' + X'Z' + W'XZ

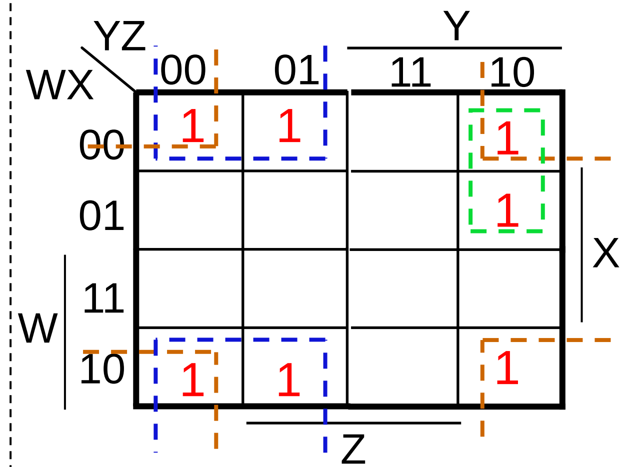
Given function:

F2(W,X,Y,Z) =
= Σm(0,1,2,4,5,6,8,
9,12,13,14)

Simplified function:

F2(W,X,Y,Z) =
= Y' + W'Z' + XZ'





Given function:

F3(W,X,Y,Z) = W'X'Y' + X'YZ' + WX'Y' + W'XYZ'

Simplified function:

F3(W,X,Y,Z) =
= X'Y' + X'Z' + W'YZ'

Simplifying with K-maps Systematically

- You have seen intuitive procedure on how to group cells and simplify Boolean functions!
- Can we have more systematic procedure?
- YES, if we introduce the terms:
  - implicant
  - **prime** implicant
  - **essential prime** implicant

- An **Implicant** **I** of a function F() is a product term which implies F(), i.e., F() = 1 whenever **I** = 1
  - All minterms of a function F are implicants of F
  - All rectangles in a K-map made up of cells containing 1s correspond to implicants

Prime Implicant (PI)

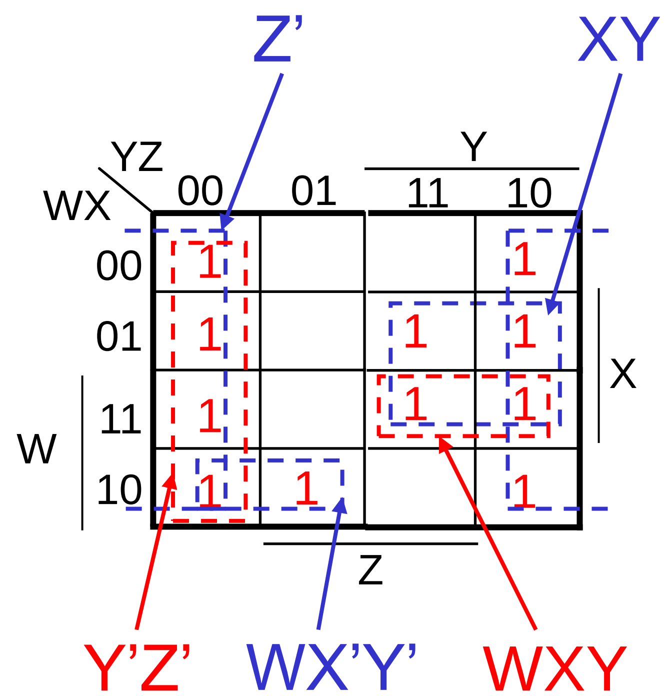
- An implicant **I** of F is called a **Prime Implicant** (PI) if the removal of any literal from **I** results in a product term that is NOT an implicant of F
  - The above should hold for all literals in **I**

- Thus, a prime implicant is not contained in any simpler implicant

- The set of prime implicants corresponds to
  - all rectangles, in a K-map, made up of cells containing 1s that satisfy the following condition:
  - each rectangle is not contained in a larger rectangle

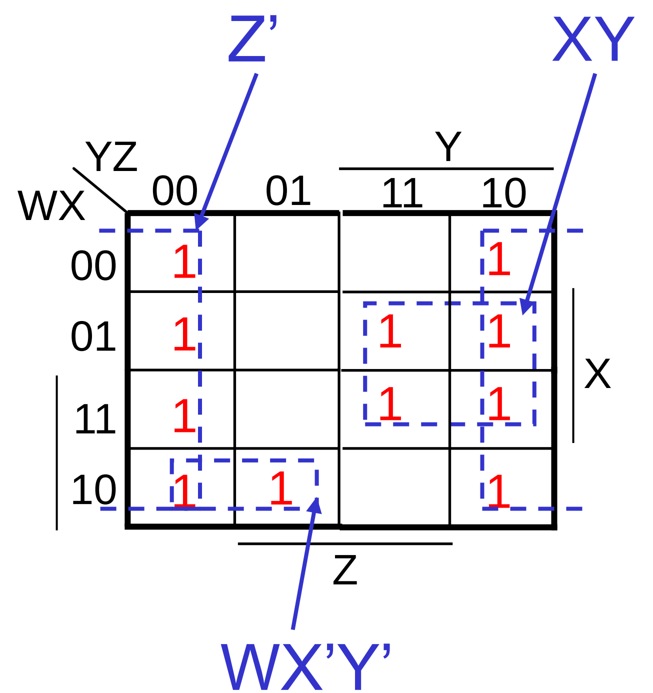
Example of Prime Implicants (PIs)

- Consider function F(W,X,Y,Z) whose K-map is shown at right
- **Y'Z'** is **not** a prime implicant because it is contained in **Z'**
- **WXY** is **not** a prime implicant because it is contained in **XY**
- Product terms **Z'**, **XY**, **WX'Y'** are prime implicants. Why?
  - Consider the term **XY** and obtain terms by deleting any literal:
    - We get two terms: term **X** and term **Y**
    - Both terms are NOT implicants of F
    - Thus, the term **XY** is prime implicant



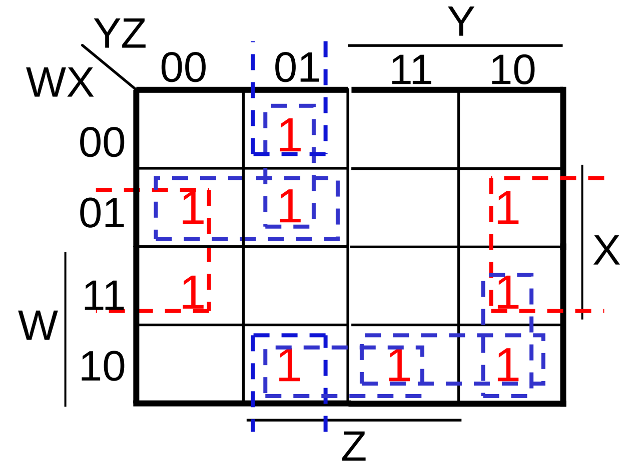
Essential Prime Implicants (EPIs)

- If a minterm of function F is included in ONLY one prime implicant *pi*, then *pi* is an **Essential Prime Implicant** of F
- An essential prime implicant MUST appear in all possible SOP expressions of function F
- To find essential prime implicants:
  - Generate all prime implicants of a function
  - Select those prime implicants that contain at least one 1 that is not covered by any other prime implicant
- For the previous example, the PIs are Z', XY, and WX'Y'; all of these are essential.

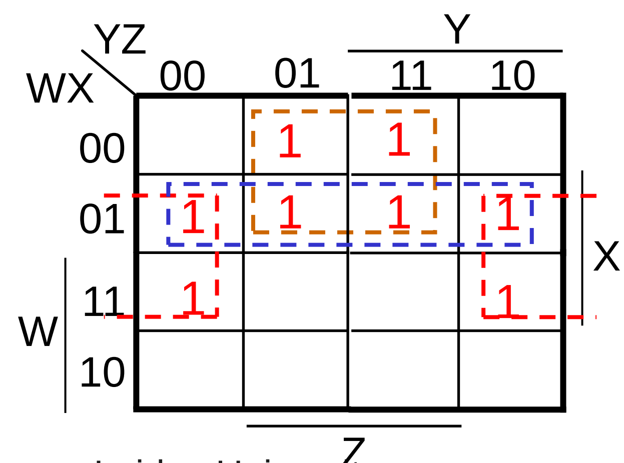


Essential Prime Implicants (examples)

- Consider function F1(W,X,Y,Z) whose K-map is shown below:
  - All Prime Implicants are:
    XZ’, W’XY’, W’Y’Z, X’Y’Z,
    WX’Z, WX’Y, WYZ’
  - Essential Prime Implicants are:
    XZ’



- Consider function F2(W,X,Y,Z) whose K-map is shown below:
  - All Prime Implicants are:
    XZ’, W’Z, W’X
  - Essential Prime Implicants are:
    XZ’ and W’Z



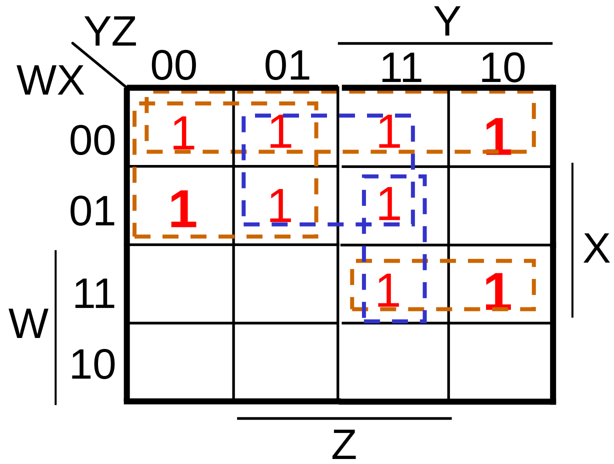
Systematic Procedure for Simplifying Boolean Functions

Given : The K-map of a Boolean function
Obtain: The simplest SOP expression for the function

1. **Find all** *prime implicants* (PIs) of the function
2. **Select all essential** PIs
3. For remaining minterms not included in the essential PIs, **select** a set of other PIs to **cover** them, with **minimal overlap** in the set
4. The resulting simplified function is the logical **OR** of the product terms **selected above**

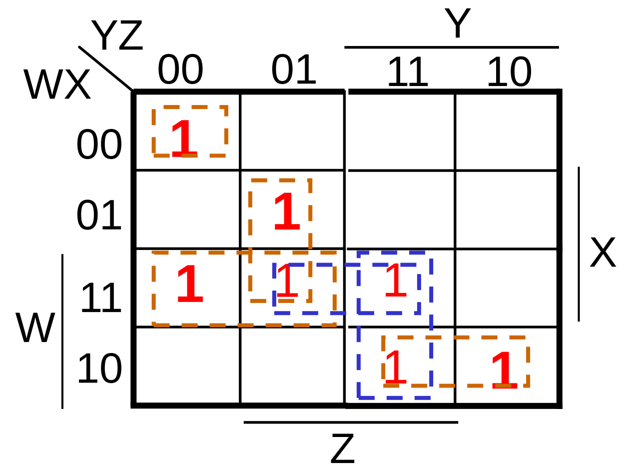
Example

- F(W,X,Y,Z) = ∑m(0,1,2,3,4,5,7,14,15).
- All prime implicants (PI) are: W'X', W'Y', W'Z, XYZ, WXY
- Select all essential PIs: W'X', W'Y', WXY
- Select other PIs to cover all 1s with minimal overlap:
  - Possibilities: W'Z or XYZ
  - We select W'Z because it is simpler.
- F(W,X,Y,Z) = W'X'+W'Y'+WXY+ W'Z



Other Examples

- Consider function F(W,X,Y,Z) whose K-map is shown at right.
- All prime implicants are:
  - W’X’Y’Z’, WXY’, WX’Y, WXZ, WYZ, XY’Z
- Essential prime implicants are:
  - W’X’Y’Z’, WXY’, WX’Y, XY’Z
- Nonessential prime implicants are:
  - WXZ, WYZ
- Simplified function (solution not unique):
  - F = W’X’Y’Z’+WXY’+WX’Y+XY’Z + WXZ
  - F = W’X’Y’Z’+WXY’+WX’Y+XY’Z + WYZ

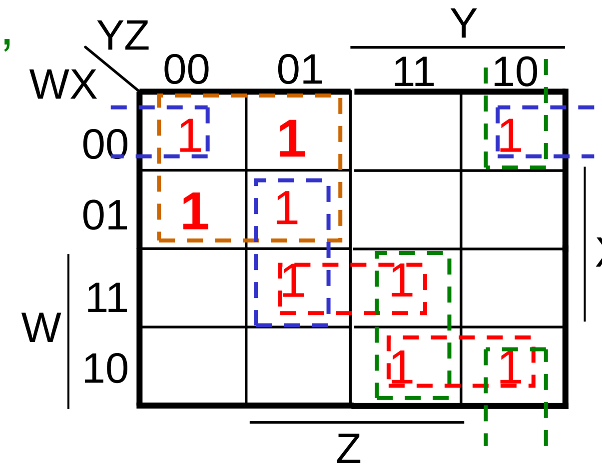


Other Examples (cont.)

- Consider function F(W,X,Y,Z) =∑m(0,1,2,4,5,10,11,13,15) whose K-map is shown at right.
- All prime implicants are:
  - W'Y', XY'Z, WXZ, WYZ, WX'Y, W'X'Z', X'YZ'
- Essential prime implicants are:
  - W'Y'
- Nonessential prime implicants are:
  - XY'Z, WXZ, WYZ, WX'Y, W'X'Z', X'YZ'
- Simplified function (solution not unique):
  - F = W'Y'+WXZ+WX'Y+W'X'Z'
  - F = W'Y'+WXZ+WX'Y+X'YZ'
  - F = W'Y'+WYZ+X'YZ'+XY'Z
  - F = W'Y'+WYZ+X'YZ'+WXZ

WXZ and WX'Y are ***NON-overlapping*** PIs.

WYZ and X'YZ' are ***NON-overlapping*** PIs.



Product-Of-Sums (POS) Simplification

- So far, we have considered simplification of a Boolean function expressed in Sum-Of-Products (SOP) form using a K-map .
- Sometimes the Product-Of-Sums form of a function is simpler than the SOP form.
- Can we use K-maps to simplify a Boolean function in Product-Of-Sums form?
- Procedure:
  - Use sum-of-products simplification on the ***zeros*** of function F in the K-map. In this way you will get the simplified complement of F (F’).
  - Find the complement of F’ which is F, i.e., (F’)’ = F
    - Recall that the complement of a Boolean function can be obtained by (1) taking the dual and (2) complementing each literal.
    - OR, using DeMorgan’s Theorem.

POS Simplification Example

$F = \sum m(0,1,2,3,4,5,7,14,15)$

WX \ YZ	00	01	11	10
00	1	1	1	1
01	1	1	1	0
11	0	0	1	1
10	0	0	0	0

The complement of F (F’)

WX \ YZ	00	01	11	10
00				
01				1
11	1	1		
10	1	1	1	1

- Simplify using <u>*zeros*</u>: $F' = WX' + WY' + W'XYZ'$
- Complement F’ to find F, i.e., **$F = (F')'$**
  - First get the dual of F’:
    dual(F’) $= (W+X') \cdot (W+Y') \cdot (W'+X+Y+Z')$
  - Complement each literal in **dual(F’)** to get **F as POS**
    F $= (W'+X) \cdot (W'+Y) \cdot (W+X'+Y'+Z)$

Don’t-Care Conditions

- Sometimes a Boolean function is not specified for some combinations of input values. Why?
  - There may be a combination of input values which will never occur
  - If they do occur, the value of the function is of no concern
- Such combinations is called ***don’t-care condition***
- The function value for such combinations is called a ***don't-care***
- The don’t-care function values are usually denoted with **x**
  - **x** may be arbitrarily set to 0 or 1 in an implementation
- Don’t-cares can be used to **further** simplify a function

Simplification using Don’t-Cares

- Treat don't-cares as if they are 1s to generate prime implicants in order to produce simple expressions
- Delete prime implicants that cover only don't-care minterms
- Treat the covering of remaining don't care minterms as optional in the selection process
  - they may be covered
  - but it is not necessary

Example with Don’t-Care Conditions

- Consider the following incompletely specified function **F** that has three don’t-care minterms **d**:
  - F(A,B,C,D) = ∑m(1,3,7,11,15)
  - d(A,B,C,D) = ∑m(0,2,5)

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

F1 = CD + A’B’

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

F2 = CD + A’D

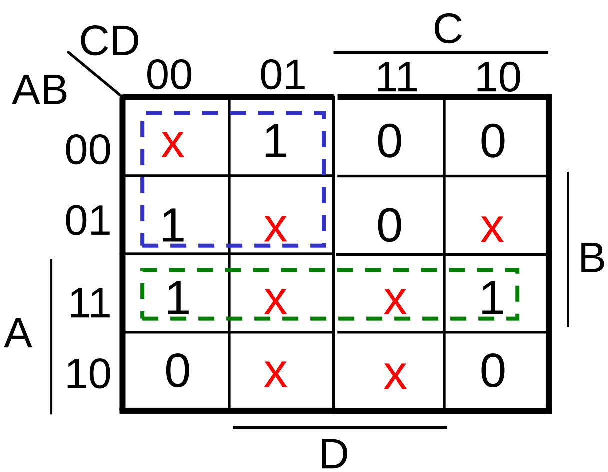
<u>Notice:</u> F1 and **F2** are algebraically ***not equal***. Both include the specified minterms of **F**, but each includes different *don’t-care* minterms.

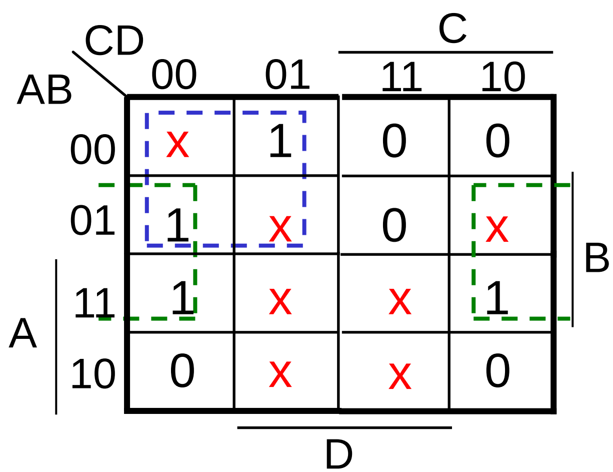
Other Examples with Don’t-Cares (1)

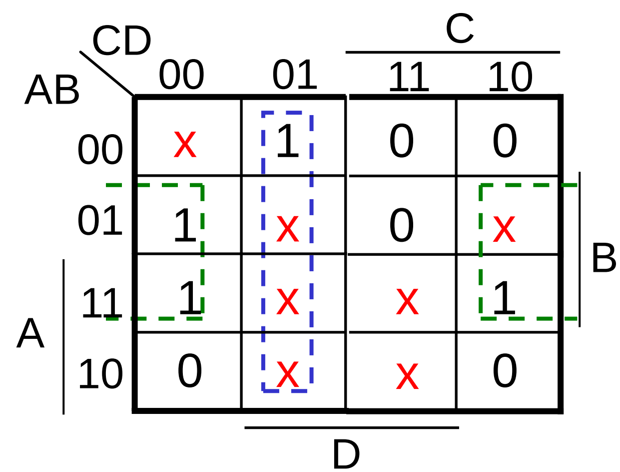
- Simplify the function G(A,B,C,D) whose K-map is shown at right.

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

G = A’C’+ AB   or   G = A’C’+BD’   or   G = BD’ + C’D







Other Examples with Don't-Cares (2)

- Simplify the function F(A,B,C,D) whose K-map is shown at the top-right.
- F = A'BC'+AB'+CD'+A'C'D

  or
- F = A'BD'+AB'+CD'+A'C'D
- The middle two terms are EPIs, while the first and last terms are selected to cover the minterms m_1, m_4, and m_5.
- There's a third solution! Can you find it?

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

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

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

Algorithmic Techniques for Simplification

- Simplification of Boolean functions using K-maps works for functions of up to 4 variables
- What do we do for functions with more than 4 variables?
- You can “code up” a minimizer program
  - Use the Quine-McCluskey algorithm
  - Base on (essential) prime implicants
- We won’t discuss these techniques here
- Search on Internet to find more information about the Quine-McCluskey algorithm