CS2100 COMPUTER ORGANISATION

http://www.comp.nus.edu.sg/~cs2100/

Lecture #15

Simplification

Lecture #15: Simplification

1. Function Simplification
2. Algebraic Simplification
3. Half Adder
4. Gray Code
5. K-maps
   - 5.1 Introduction
   - 5.2 How to use K-maps
   - 5.3 Converting to Minterms Form
   - 5.4 Prime Implicants (PIs) and Essential Prime Implicants (EPIs)
   - 5.5 Finding Simplified SOP Expression
   - 5.6 Finding Simplified POS Expression
   - 5.7 Don't-care Conditions
6. More Examples

1. Function Simplification

- Why simplify?
  - Simpler expression leads to circuit that uses fewer logic gates.
  - Thus cheaper, uses less power, (sometimes) faster.
- Techniques
  - Algebraic
    - Using theorems
    - Open-ended; requires skills
  - Karnaugh Maps
    - Easy to use
    - Limited to no more than 6 variables
  - Quine-McCluskey (non-examinable)
    - Suitable for automation
    - Can handle many variables (but computationally intensive)

2. Algebraic Simplification (1/3)

- Aims to minimise
  - Number of literals, and
  - Number of terms
- But sometimes conflicting, so let's aim at reducing the number of literals for the examples in the next few slides.
- Challenging – requires good algebraic manipulation skills.

2. Algebraic Simplification (2/3)

- Example 1: Simplify (x+y)·(x+y')·(x'+z)

(x+y)·(x+y')·(x'+z)	
= (**x·x** + x·y' + x·y + y·y') · (x'+z)	(distributivity)
= (x + **x·y' + x·y** + y·y') · (x'+z)	(idempotency)
= (x + x·(**y'+y**) + **y·y'**) · (x'+z)	(distributivity)
= (x + x·(1) + 0) · (x'+z)	(complement)
= (**x + x**) · (x'+z)	(identity)
= **x · (x'+z)**	(idempotency)
= **x·x'** + x·z	(distributivity)
= **0 + x·z**	(complement)
= x·z	(identity)

Number of literals reduced from 6 to 2.

2. Algebraic Simplification (3/3)

- Example 2: Find the simplified SOP expression of

  F(a,b,c,d) = a·b·c + a·b·d + a'·b·c' + c·d + b·d'

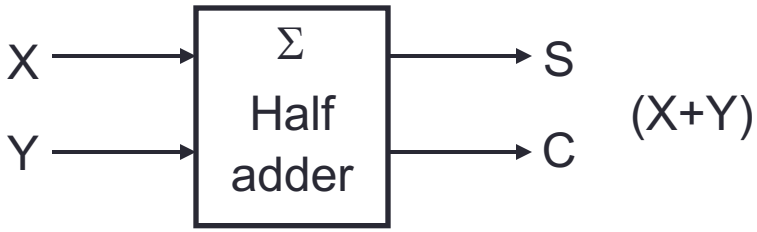
a·b·c + **a·b·d** + a'·b·c' + c·d + **b·d'**	
= a·b·c + **a·b + a'·b·c'** + c·d + b·d'	(absorption 2)
= **a·b·c + a·b** + b·c' + c·d + b·d'	(absorption 2)
= a·b + **b·c'** + c·d + **b·d'**	(absorption 1)
= a·b + c·d + b·(**c'+d'**)	(distributivity)
= a·b + **c·d + b·(c·d)'**	(DeMorgan's)
= **a·b** + c·d + **b**	(absorption 1)
= b + c·d	(absorption 1)

a·b·d + **b·d'**
= **b·** (**a·d** + **d'**)
= **b·** (**a** + **d'**)
= **a·b** + **b·d'**

Number of literals reduced from 13 to 3.

3. Half Adder (1/2)

- **Half adder** is a circuit that adds 2 single bits (X, Y) to produce a result of 2 bits (C, S).
- The black-box representation and truth table for half adder are shown below.



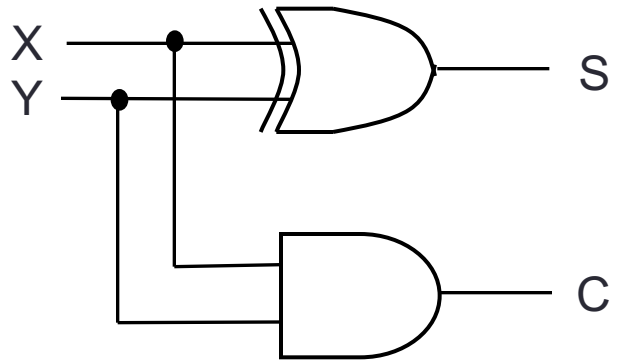
(X+Y)

Inputs		*Outputs*	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

3. Half Adder (2/2)

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- In canonical form (sum-of-minterms):
  - C = X·Y
  - S = X'·Y + X·Y'
- Output S can be simplified further (though no longer in SOP form):
  - S = X'·Y + X·Y' = X ⊕ Y
- Implementation of a half adder



4. Gray Code (1/3)

- Unweighted (not an arithmetic code)
- Only a single bit change from one code value to the next.
- Not restricted to decimal digits: *n* bits → 2^n values.
- Good for error detection.
- Named after Frank Gray; also called reflected binary code.
- Example: 4-bit standard Gray code

Decimal	Binary	Gray Code	Decimal	Binary	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

4. Gray Code (2/3)

- There are many Gray code sequences.
- Example: For 3 bits, here are some possible Gray code sequences:

000	000	110
001	010	111
011	110	101
010	111	100
110	011	000
111	001	001
101	101	011
100	100	010

This is the standard Gray Code.

These are NOT Gray codes (why?)

000	010	010
001	110	011
010	101	111
011	001	110
100	100	111
101	111	101
110	000	001
111	011	000

4. Gray Code (3/3)

- Generating a 4-bit standard Gray code sequence.

```
0 0 0 0
0 0 0 1
0 0 1 1
0 0 1 0

0 1 1 0
0 1 1 1
0 1 0 1
0 1 0 0

1 1 0 0
1 1 0 1
1 1 1 1
1 1 1 0
1 0 1 0
1 0 1 1
1 0 0 1
1 0 0 0
```

- How to generate 5-bit standard Gray code sequence? 6-bit standard Gray code sequence?

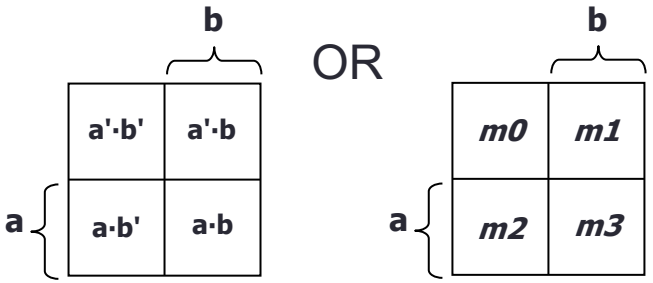
5.1 Introduction to K-maps

- Systematic method to obtain simplified sum-of-products (SOP) expressions.
- Objective: *Fewest* possible product terms and literals.
- Diagrammatic technique based on a special form of *Venn diagram*.
- Advantage: Easy to use.
- Disadvantage: Limited to 5 or 6 variables.
- Karnaugh-map (K-map) is an abstract form of Venn diagram, organised as a matrix of squares, where
  - Each square represents a minterm
  - Two adjacent squares represent minterms that differ by exactly one literal

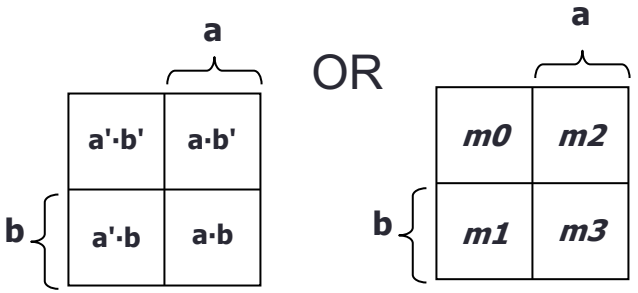
5.1 2-Variable K-maps (1/3)

- Let the 2 variables be a and b. The K-map can be drawn as…
- Alternative layouts of a 2-variable (a, b) K-map:

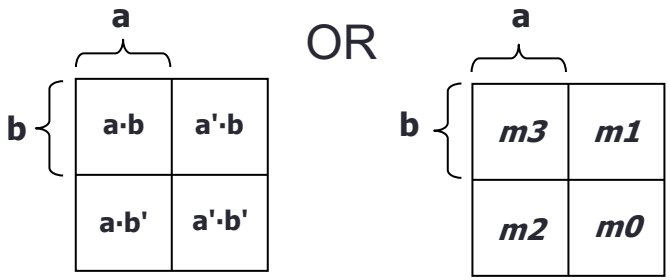
Alternative 1:



Alternative 2:



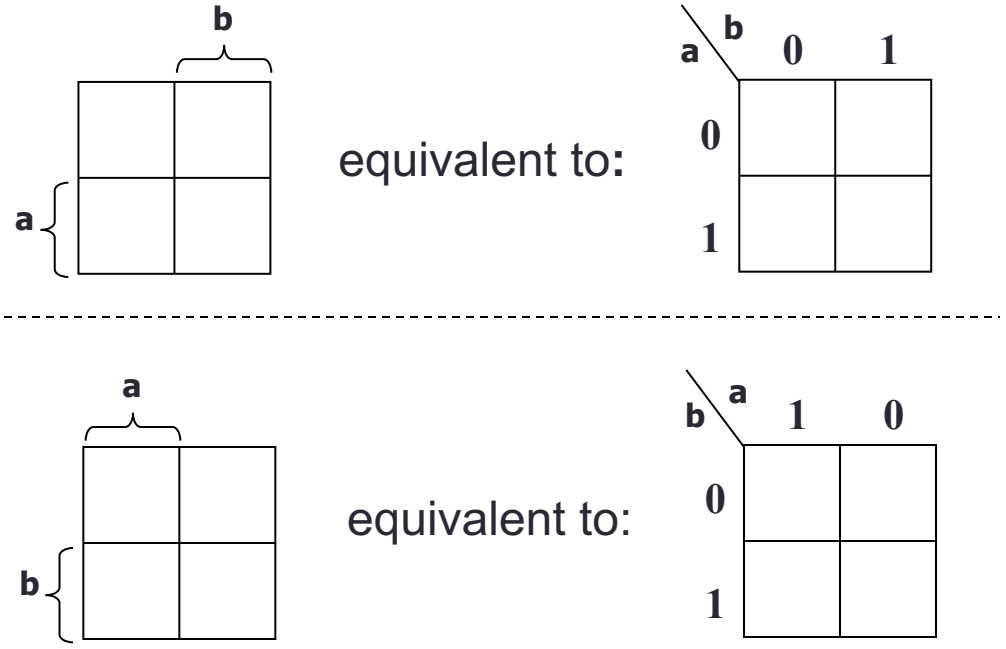
Alternative 3:



and others…

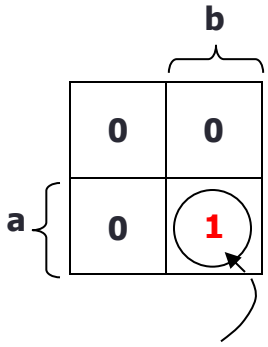
5.1 2-Variable K-maps (2/3)

- Alternative labelling of a 2-variable (a, b) K-map:

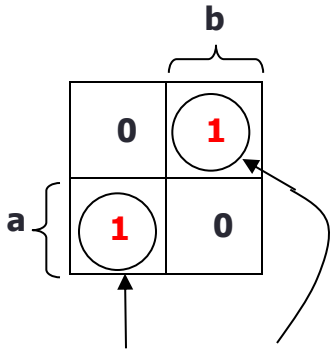


5.1 2-Variable K-maps (3/3)

- The K-map for a function is filled by putting
  - A ‘1’ in the square the corresponds to a minterm of the function
  - A ‘0’ otherwise
- Example: Half adder.



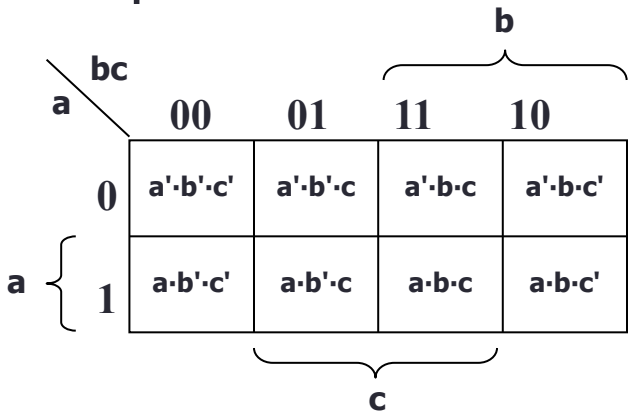
C = a·b



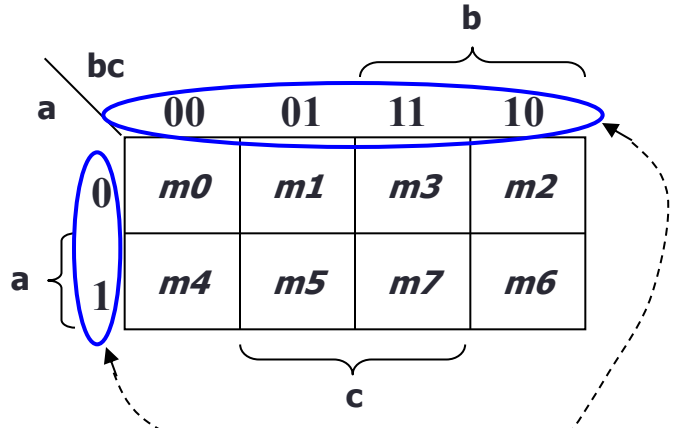
S = a·b' + a'·b

5.1 3-Variable K-maps (1/2)

- As there are 8 minterms for 3 variables, so there are 8 squares in a 3-variable K-map.
- Example: Let the variables be a, b, c.



OR

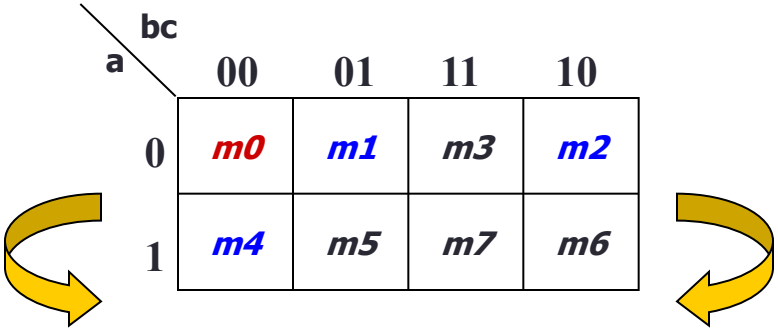


Above arrangement ensures that minterms of adjacent cells differ by only *ONE literal*. (Other arrangements which satisfy this criterion may also be used.)

Note *Gray code* sequence
A Gray code sequence is one where the value differs from its previous by 1 bit.

5.1 3-Variable K-maps (2/2)

- There is wrap-around in the K-map:
  - a'·b'·c' (*m0*) is adjacent to a'·b·c' (*m2*)
  - a·b'·c' (*m4*) is adjacent to a·b·c' (*m6*)



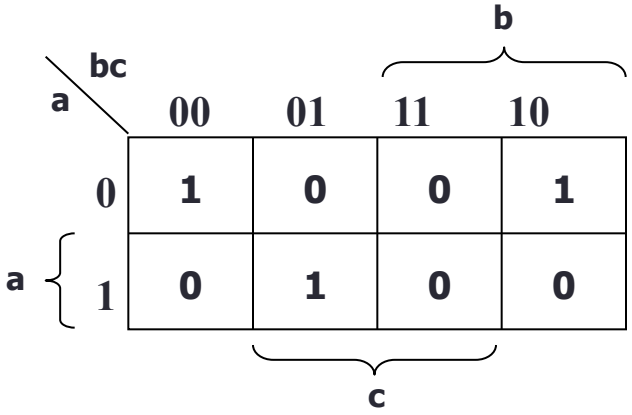
Each cell in a 3-variable K-map has 3 adjacent neighbours. For example, *m0* has 3 adjacent neighbours: *m1*, *m2* and *m4*.

In general, each cell in an *n*-variable K-map has *n* adjacent neighbours.

Quick Review Questions #1

- DLD page 106, questions 5-1 to 5-2.

5-1. The K-map of a 3-variable function *F* is shown below. What is the sum-of-minterms expression of *F*?

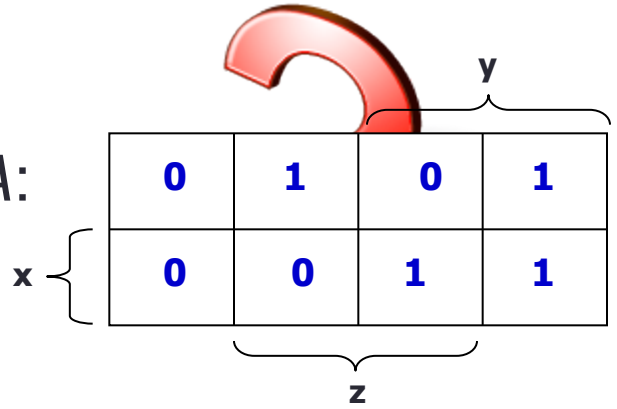


$\Sigma m(0, 2, 5)$

= a'·b'·c' + a'·b·c' + a·b'·c

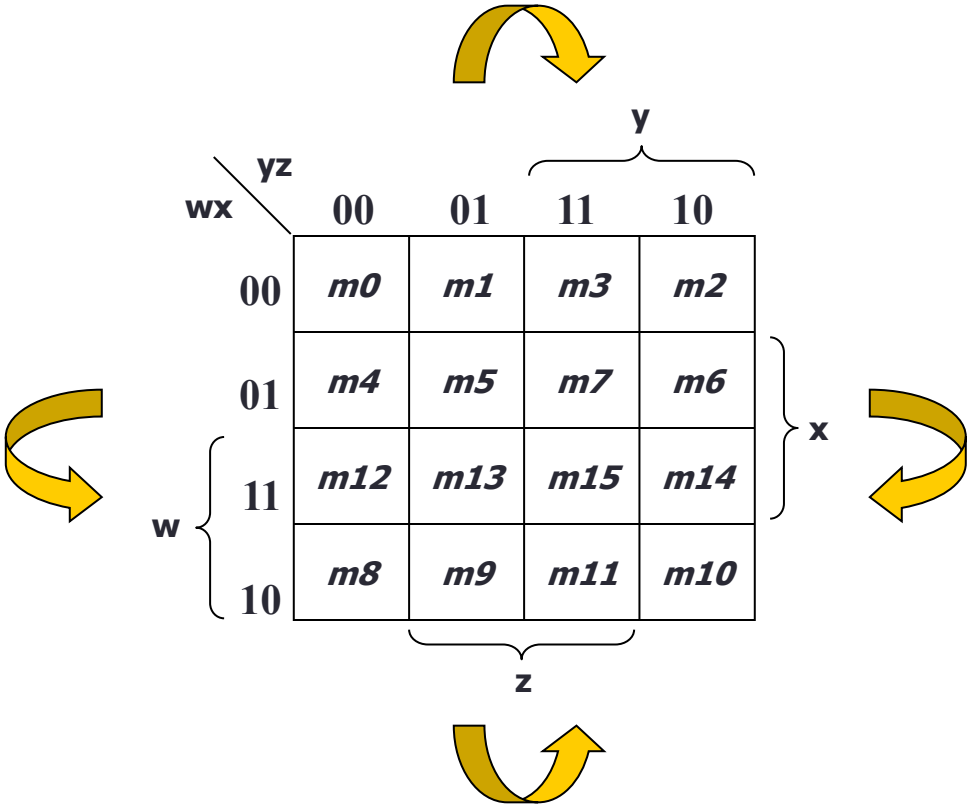
5-2. Draw the K-map for this function *A*:

$A(x, y, z) = x \cdot y + y \cdot z' + x' \cdot y' \cdot z$



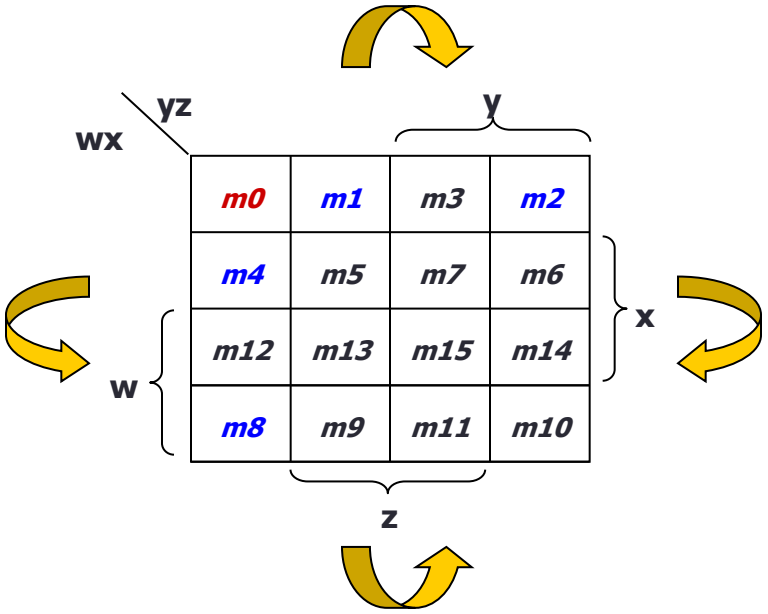
5.1 4-Variable K-maps (1/2)

- There are 16 square cells in a 4-variable K-map.
- Example: Let the variables be w, x, y, z.



5.1 4-Variable K-maps (2/2)

- There are 2 wrap-arounds.
- Every cell has 4 neighbours.
  - Example: The cell corresponding to minterm *m0* has neighbours *m1*, *m2*, *m4* and *m8*.

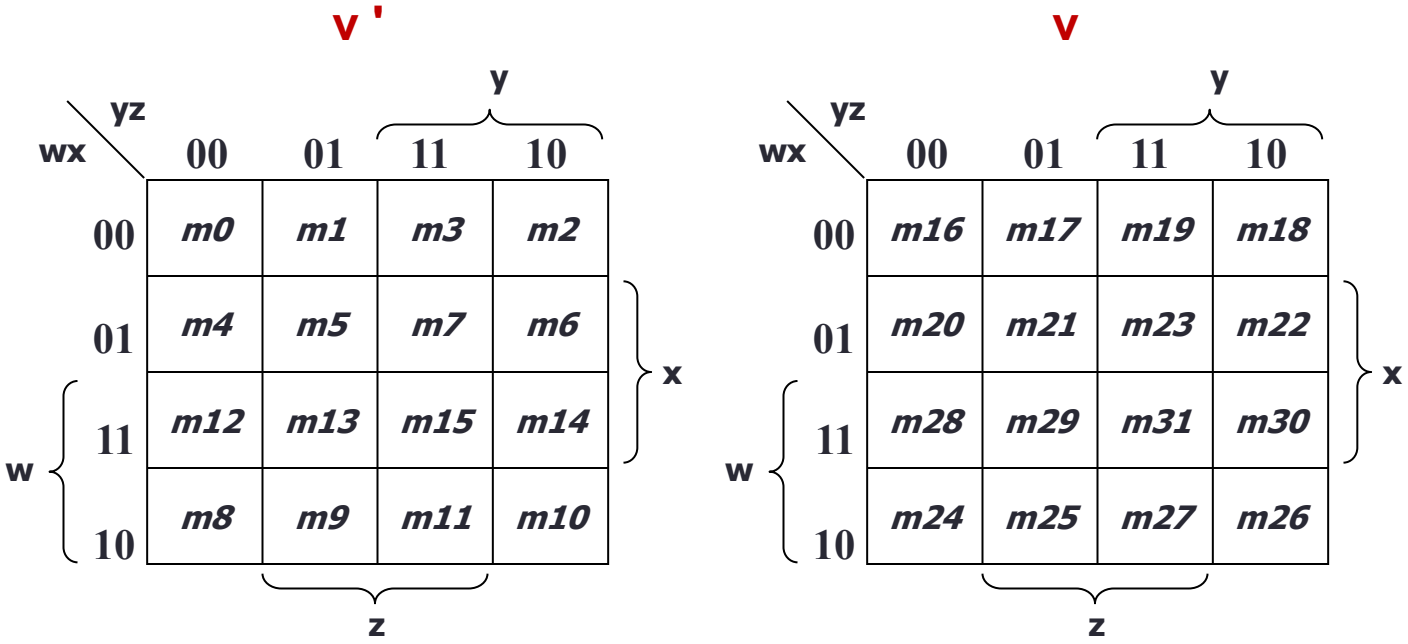


5.1 5-Variable K-maps (1/2)

- K-maps of more than 4 variables are more difficult to use because the geometry (hypercube configurations) for combining adjacent squares becomes more involved.
- For 5 variables, e.g. v, w, x, y, z, we need $2^5 = 32$ squares.
- Each square has 5 neighbours.

5.1 5-Variable K-maps (2/2)

- Organised as two 4-variable K-maps. One for v' and the other for v.

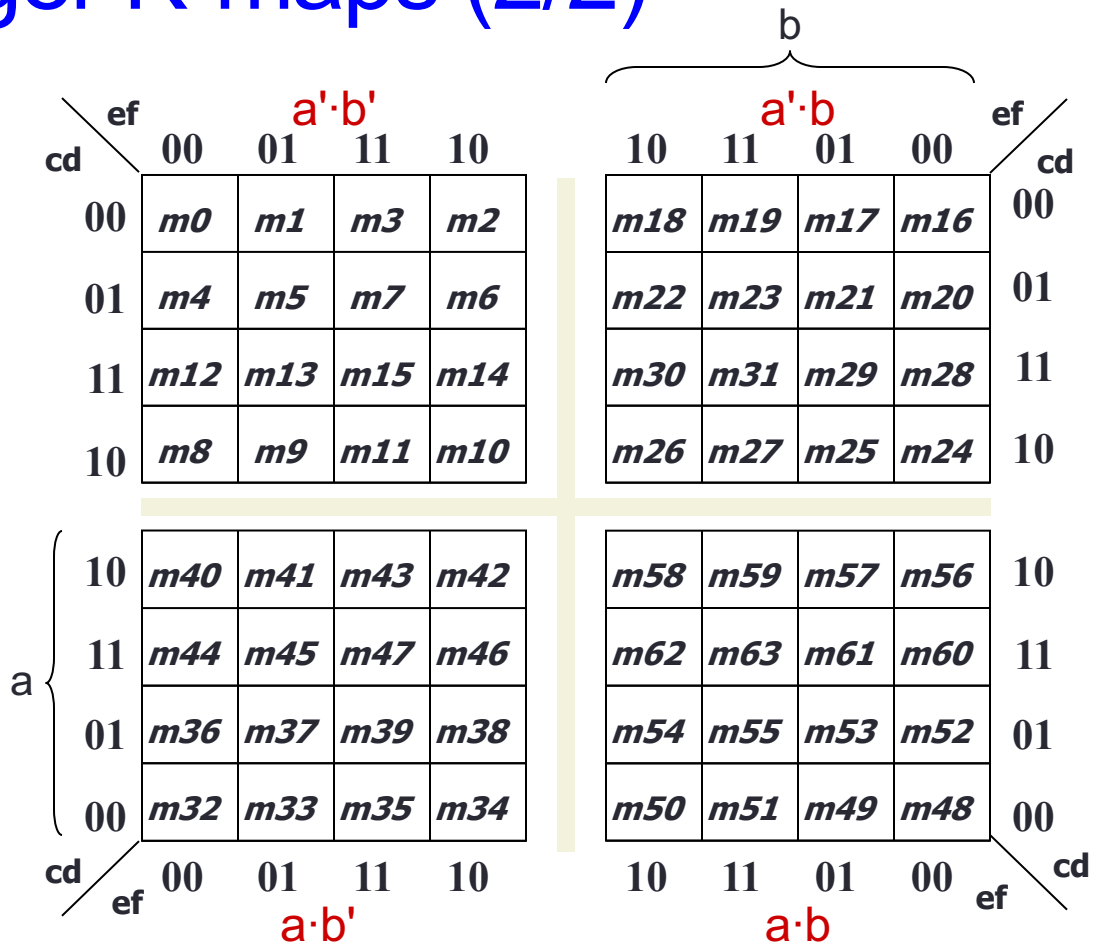


Corresponding squares of each map are adjacent.
Can visualise this as *one 4-variable K-map* being on TOP *of the other 4-variable K-map*.

5.1 Larger K-maps (1/2)

- 6-variable K-map is pushing the limit of human's "pattern-recognition" capability.
- K-maps larger than 6 variables are practically unheard of!
- Normally, a 6-variable K-map is organised as four 4-variable K-maps, mirrored along two axes.

5.1 Larger K-maps (2/2)



Try stretching your recognition capability by finding simplest sum-of-products expression for Σ m(6,8,14,18,23,25,27,29,41,45,57,61).

5.2 How to Use K-maps (1/7)

- Based on the Unifying Theorem (complement law):

  A + A' = 1

- In a K-map, each cell containing a '1' corresponds to a minterm of a given function *F* where the output is 1.
- Each valid grouping of adjacent cells containing '1' then corresponds to a simpler product term of *F*.
  - A group must have size in powers of two: 1, 2, 4, 8, …
  - Grouping 2 adjacent cells eliminates 1 variable from the product term; grouping 4 cells eliminates 2 variables; grouping 8 cells eliminates 3 variables, and so on. In general, grouping 2^n cells eliminates *n* variables.

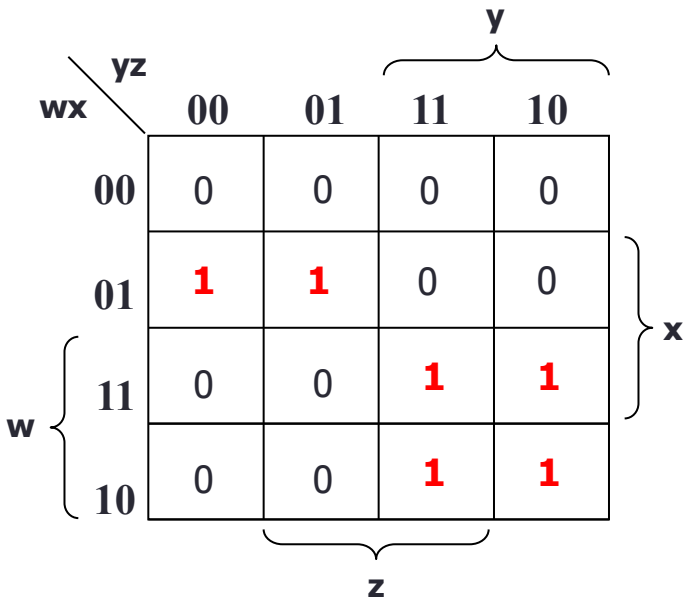
5.2 How to Use K-maps (2/7)

- Group as many cells as possible
  - The larger the group, the fewer the number of literals in the resulting product term.
- Select as few groups as possible to cover all the cells (minterms) of the function
  - The fewer the groups, the fewer is the number of product terms in the simplified SOP expression.

5.2 How to Use K-maps (3/7)

- Example:

$$F(w,x,y,z) = w'\cdot x\cdot y'\cdot z' + w'\cdot x\cdot y'\cdot z + w\cdot x'\cdot y\cdot z' + w\cdot x'\cdot y\cdot z + w\cdot x\cdot y\cdot z' + w\cdot x\cdot y\cdot z$$
$$= \Sigma\, m(4, 5, 10, 11, 14, 15)$$



5.2 How to Use K-maps (4/7)

- Each group of adjacent minterms corresponds to a possible product term of the given function.
- Here, there are 2 groups of minterms, A and B:

A = w'.x.y'.z' + w'.x.y'.z = w'.x.y'.(z' + z) = **w'.x.y'**

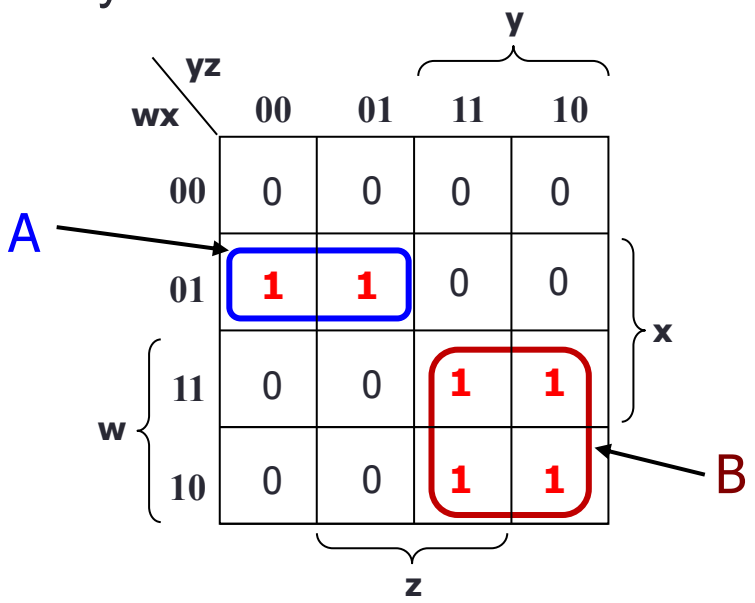
B = w.x'.y.z' + w.x'.y.z + w.x.y.z' + w.x.y.z

= w.x'.y.(z' + z) + w.x.y.(z' + z)

= w.x'.y + w.x.y

= w.(x'+x).y

= **w.y**



5.2 How to Use K-maps (5/7)

- Each product term that corresponds to a group, w'·x·y' and w·y, represents the sum of minterms in that group.
- Boolean expression is therefore the sum of product terms (SOP) that represent all groups of the minterms of the function:

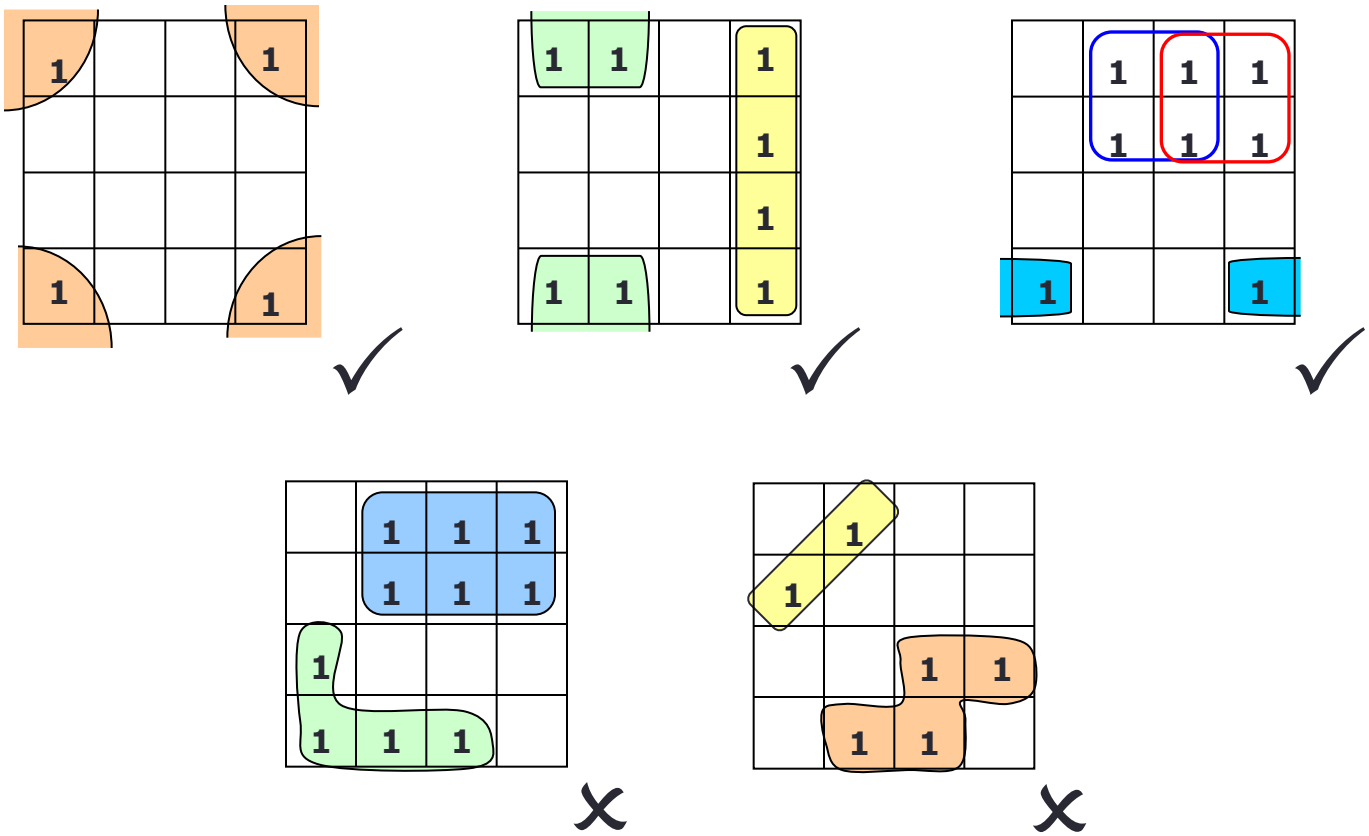
  $F(w,x,y,z)$ = group A + group B = w'·x·y' + w·y

5.2 How to Use K-maps (6/7)

- The larger the group (the more minterms it contains), the fewer is the number of literals in the associated product term.
  - Recall that a group must have size in powers of two.
  - Example: For a 4-variable K-map with variables w, x, y, z
    1 cell = 4 literals. Examples: w·x·y·z, w'·x·y'·z
    2 cells = 3 literals. Examples: w·x·y, w·y'·z'
    4 cells = 2 literals. Examples: w·x, x'·y
    8 cells = 1 literal. Examples: w, y', z
    16 cells = no literal (i.e. logical constant 1). Example: 1

5.2 How to Use K-maps (7/7)

- Examples of valid and invalid groupings.



5.3 Converting to Minterms Form (1/2)

- The K-map of a function can be easily filled in when the function is given in sum-of-minterms form.
- What if it is not in sum-of-minterms form?
  - Convert it into sum-of-products (SOP) form
  - Expand the SOP expression into sum-of-minterms expression, or fill in the K-map directly based on the SOP expression.

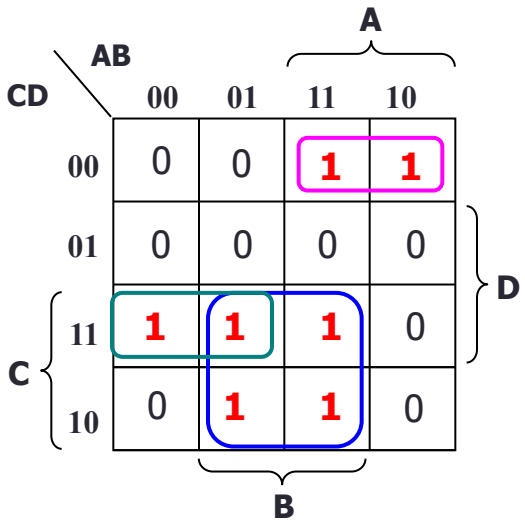
5.3 Converting to Minterms Form (2/2)

- Example:

F(A,B,C,D) = A.(C+D)'.(B'+D') + C.(B+C'+A'.D)
= A.(C'.D').(B'+D') + B.C + C.C' + A'.C.D
= A.B'.C'.D' + A.C'.D' + B.C + A'.C.D

Expanding it to sum of minterms (unnecessary):

A.B'.C'.D' + A.C'.D' + B.C + A'.C.D
= A.B'.C'.D' + A.C'.D'.(B+B') + B.C + A'.C.D
= A.B'.C'.D' + A.B.C'.D' + A.B'.C'.D' + B.C.(A+A') + A'.C.D
= A.B'.C'.D' + A.B.C'.D' + A.B.C + A'.B.C + A'.C.D
= A.B'.C'.D' + A.B.C'.D' + A.B.C.(D+D') + A'.B.C.(D+D') + A'.C.D.(B+B')
= A.B'.C'.D' + A.B.C'.D' + A.B.C.D + A.B.C.D' + A'.B.C.D + A'.B.C.D' + A'.B'.C.D

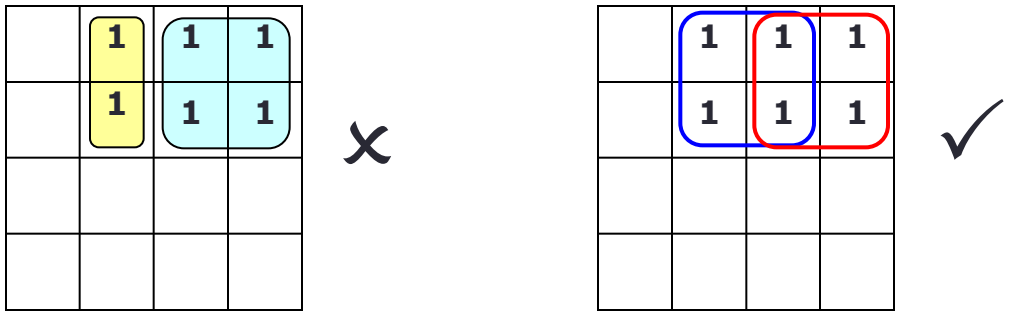


5.4 PIs and EPIs (1/3)

- To find the simplest (minimal) SOP expression from a K-map, you need to obtain:
  - Minimum number of literals per product term; and
  - Minimum number of product terms.
- Achieved through K-map using
  - *Bigger groupings* of minterms (prime implicants) where possible; and
  - *No redundant groupings* (look for essential prime implicants)
- Implicant: a product term that could be used to cover minterms of the function.

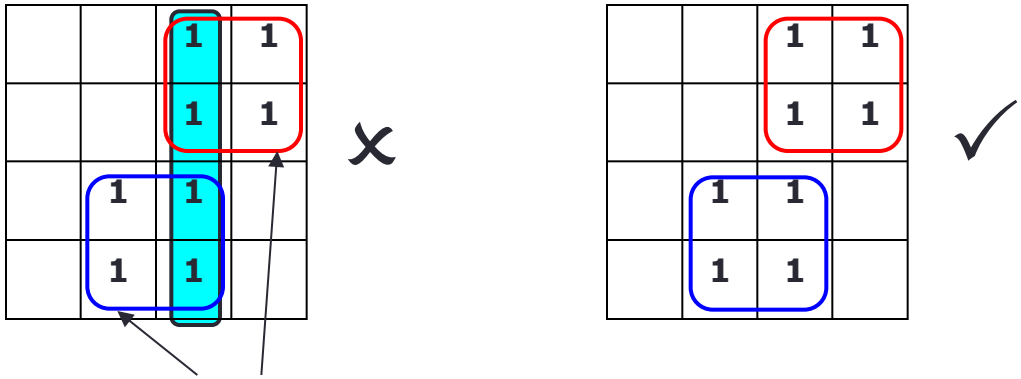
5.4 PIs and EPIs (2/3)

- Prime implicant (PI): a product term obtained by combining the *maximum possible number of minterms* from *adjacent* squares in the map. (That is, it is the biggest grouping possible.)
- Always look for prime implicants in a K-map.



5.4 PIs and EPIs (3/3)

- No redundant groups:

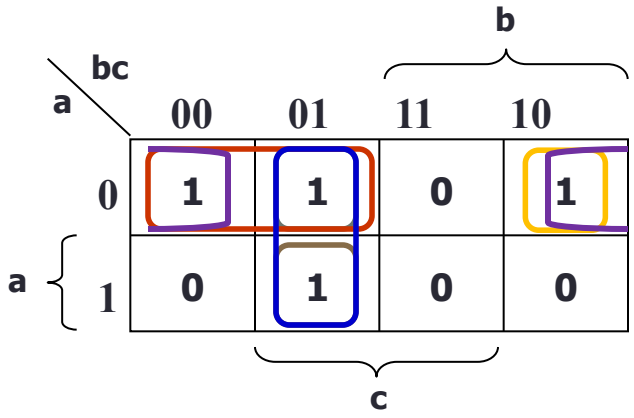


- Essential prime implicant (EPI): a prime implicant that includes at least one minterm that is not covered by any other prime implicant.

Quick Review Questions #2

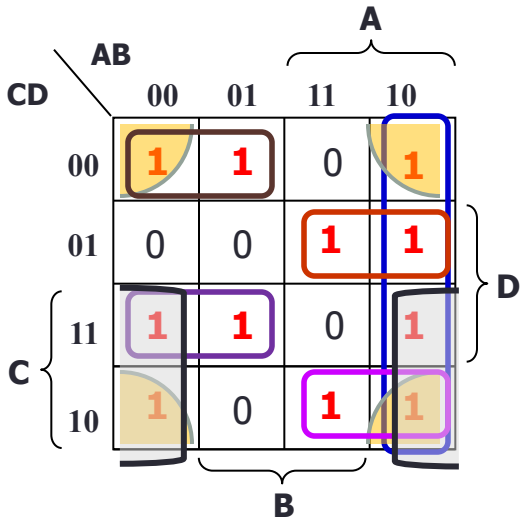
- DLD page 106, question 5-3.

  5-3. Identify the prime implicants and essential prime implicants of the two K-maps below.



How many PIs? 3 PIs
How many EPIs? 2 EPIs

How many PIs? 7 PIs
How many EPIs? 4 EPIs



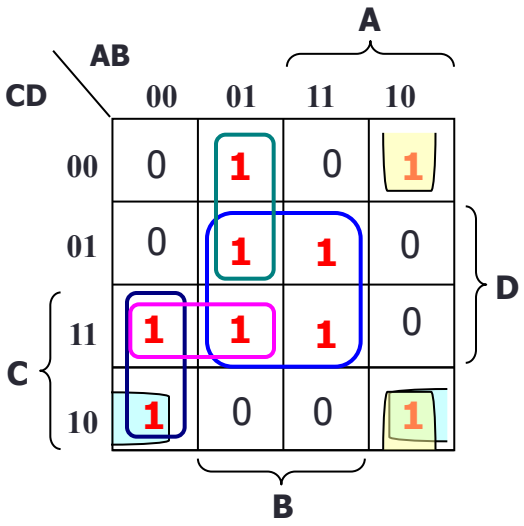
5.5 Finding Simplified SOP Expression (1/4)

- Algorithm
  1. Circle all prime implicants on the K-map.
  2. Identify and select all essential prime implicants for the cover.
  3. Select a minimum subset of the remaining prime implicants to complete the cover, that is, to cover those minterms not covered by the essential prime implicants.

5.5 Finding Simplified SOP Expression (2/4)

- Example #1:

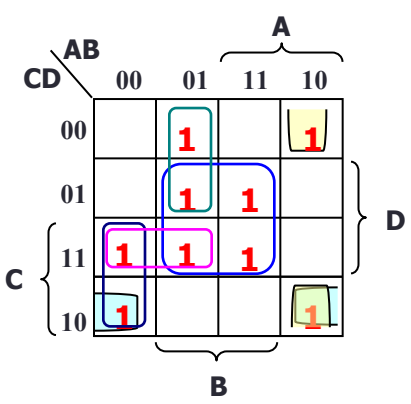
$$F(A,B,C,D) = \Sigma\, m(2,3,4,5,7,8,10,13,15)$$

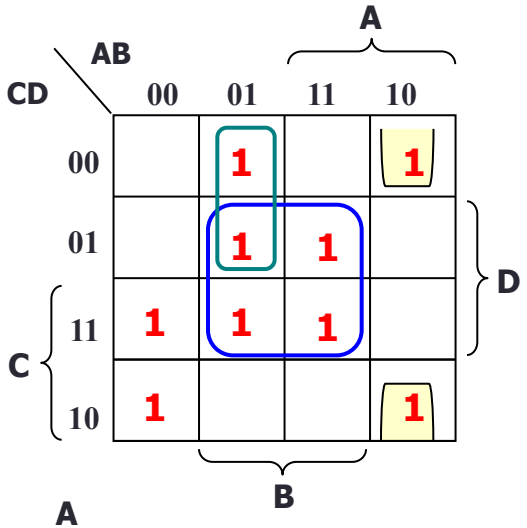


All prime implicants

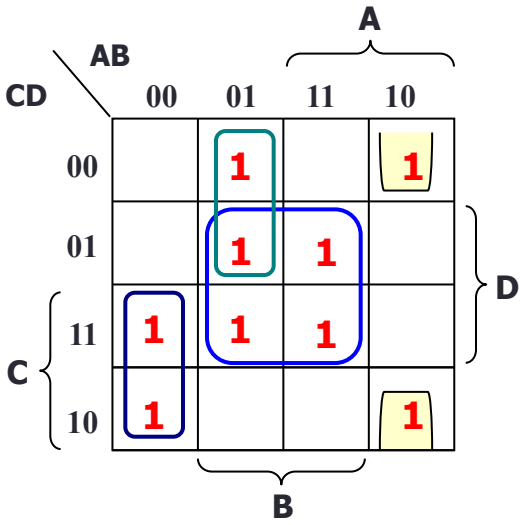
5.5 Finding Simplified SOP Expression (3/4)

All PIs



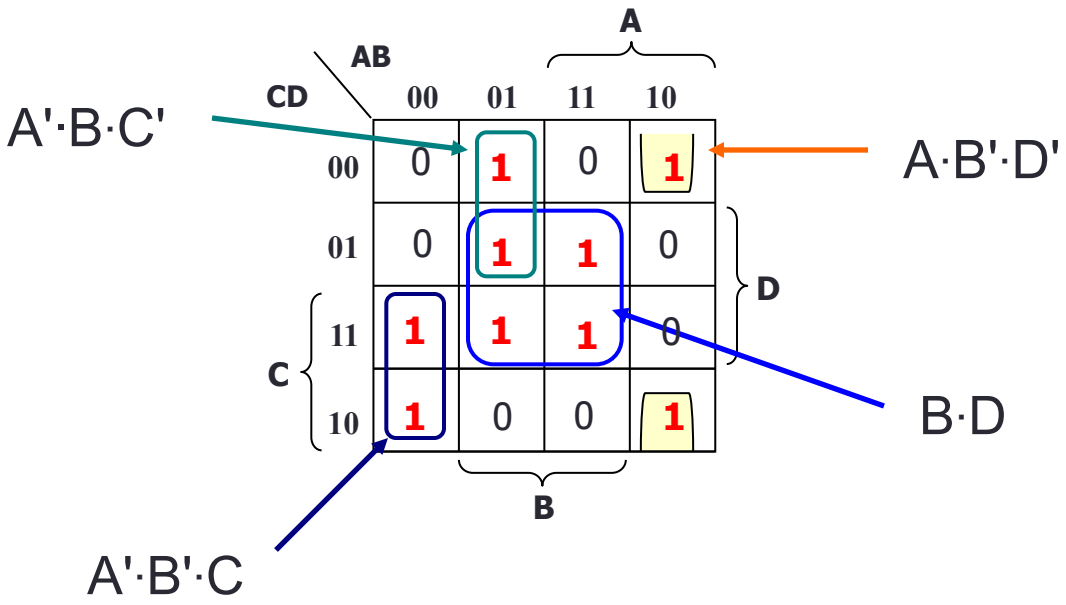


Essential prime implicants



Answer

5.5 Finding Simplified SOP Expression (4/4)



F(A,B,C,D) = B·D + A'·B·C' + A·B'·D' + A'·B'·C

Quick Review Questions #3

- DLD pages 106-107, questions 5-4 to 5-7.

5-4. Find the minimal SOP expression for G(A,B,C,D).

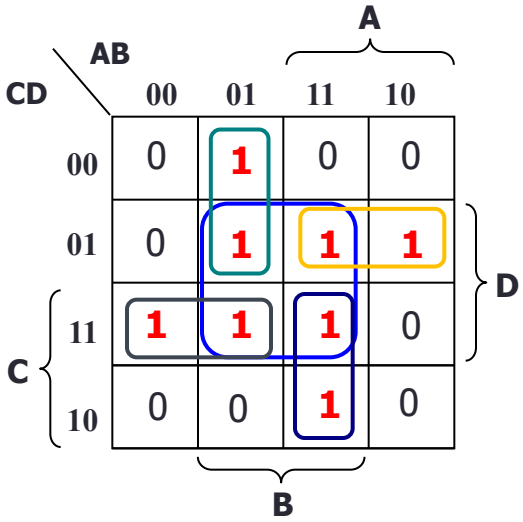
Prime implicants?

B·D, A'·B·C', A·C'·D,

A·B·C, A'·C·D

Essential prime implicants?

All, except B·D, are essential.

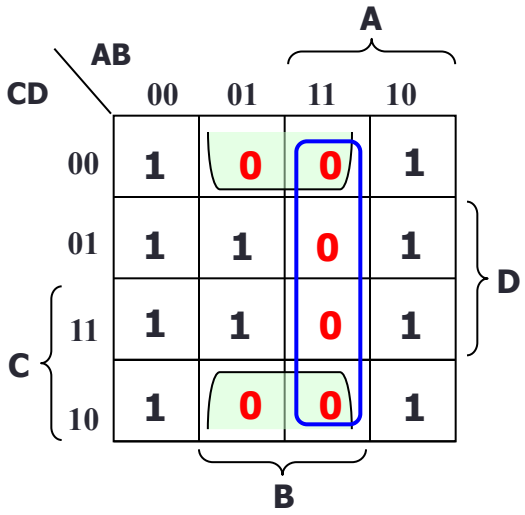


$$G = A'\cdot B\cdot C' + A\cdot C'\cdot D + A\cdot B\cdot C + A'\cdot C\cdot D$$

5.6 Finding Simplified POS Expression (1/2)

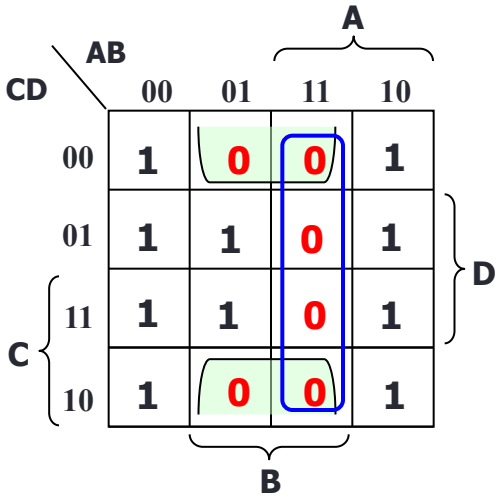
- Simplified POS expression can be obtained by grouping the maxterms (i.e. 0s) of the given function.
- Example:

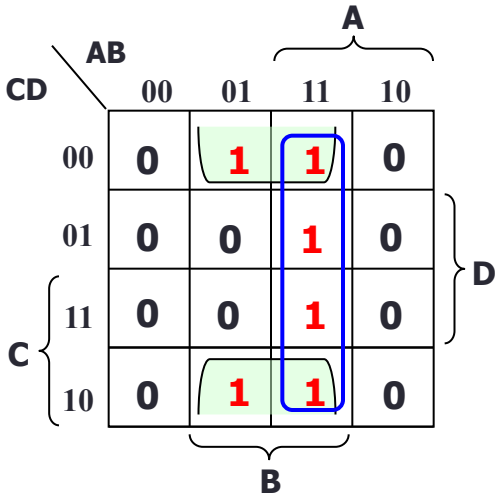
  Given F = Σm(0,1,2,3,5,7,8,9,10,11), we first draw the K-map, then group the maxterms together:



5.6 Finding Simplified POS Expression (2/2)

K-map of **F**





K-map of **F'**

- This gives the SOP of F' to be

  $F' = B \cdot D' + A \cdot B$

- To get POS of F, we have

$$
\begin{aligned}
F &= (B \cdot D' + A \cdot B)' \\
&= (B \cdot D')' \cdot (A \cdot B)' && \text{(DeMorgan)} \\
&= \mathbf{(B'+D) \cdot (A'+B')} && \text{(DeMorgan)}
\end{aligned}
$$

5.7 Don't-Care Conditions (1/5)

- In certain problems, some outputs are not specified or are invalid. Hence, these outputs can be either '1' or '0'.
- They are called don't-care conditions, denoted by X (or d).
- Example: A circuit takes in a 3-bit value ABC and outputs 2-bit value FG which is the sum of the input bits. It is also known that inputs 000 and 111 never occur.

Assuming all inputs are valid.

A	B	C	F	G
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**1**	**1**

Assuming inputs 000 and 111 are invalid.

A	B	C	F	G
0	0	0	**X**	**X**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**X**	**X**

5.7 Don't-Care Conditions (2/5)

- Don't-care conditions can be used to help simplify Boolean expression further in K-maps.
- They could be chosen to be either '1' or '0', depending on which choice results in a simpler expression.
- We usually use the notation Σd to denote the set of don't-care minterms.
  - Example:

    $F(A,B,C) = \Sigma m(3, 5, 6) + \Sigma d(0, 7)$

    $G(A,B,C) = \Sigma m(1, 2, 4) + \Sigma d(0, 7)$

Assuming inputs 000 and 111 are invalid.

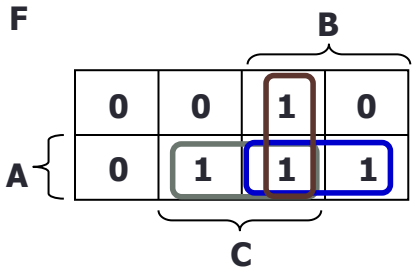
A	B	C	F	G
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	X	X

5.7 Don't-Care Conditions (3/5)

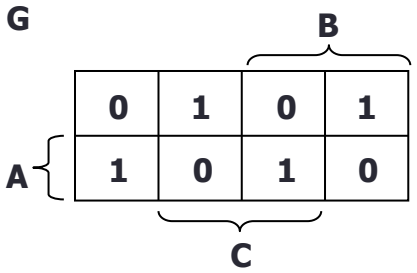
- Comparison
  - Without don't-cares:

    F(A,B,C) = Σm(3, 5, 6, 7)

    G(A,B,C) = Σm(1, 2, 4, 7)



F = A·C + A·B + B·C

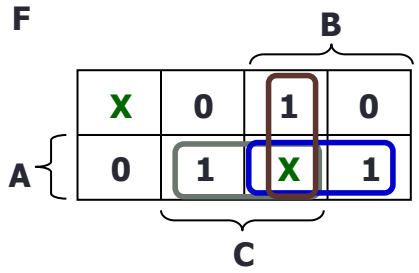


G = A·B'·C' + A'·B'·C + A·B·C + A'·B·C'

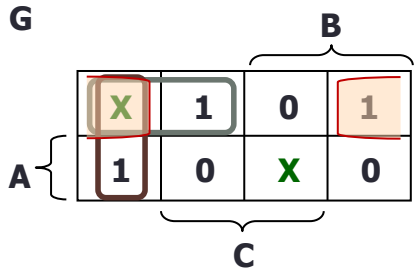
  - With don't-cares:

    F(A,B,C) = Σm(3, 5, 6) + Σd(0, 7)

    G (A,B,C)= Σm(1, 2, 4) + Σd(0, 7)



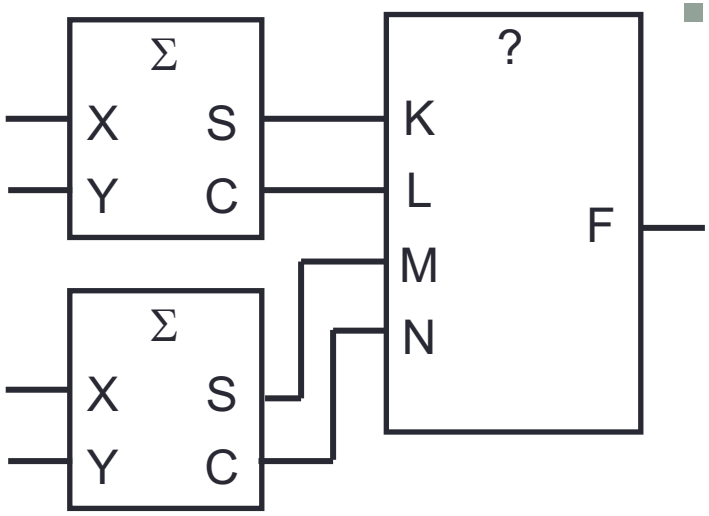
F = A·C + A·B + B·C



G = B'·C' + A'·B' + A'·C'

5.7 Don't-Care Conditions (4/5)

- Suppose you are given the truth table for a function F(K,L,M,N) as follows:
- You are also told that the inputs K, L, M, N are taken from the outputs of two half adders as shown:

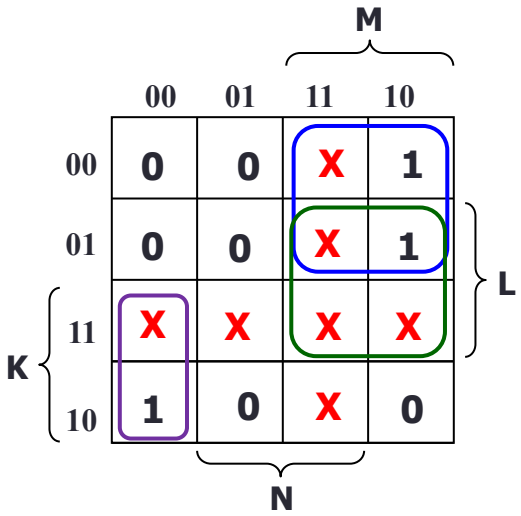


- Then you may revise the truth table:

K	L	M	N	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	X
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
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

5.7 Don't-Care Conditions (5/5)

- K-map of *F*:



- PIs: K'·M, L·M, and K·M'·N'
  (Note: K·L and M·N not considered PIs as they consist of only X's.)
- EPIs: K'·M and K·M'·N'
- Simplified SOP:

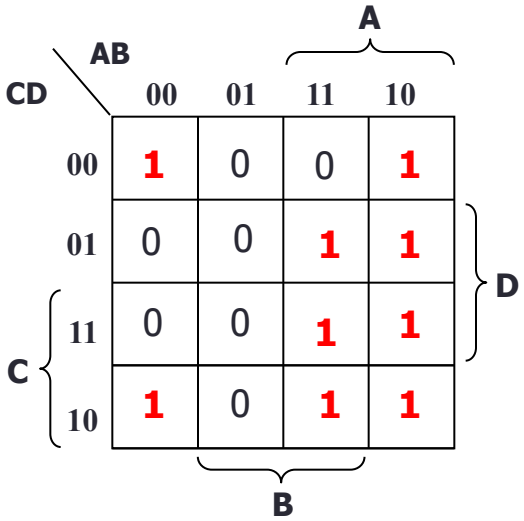
  F = K'·M + K·M'·N'

K	L	M	N	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	X
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
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

6. More Examples (1/6)

- Example #2:

$$F(A,B,C,D) = A \cdot B \cdot C + B' \cdot C \cdot D' + A \cdot D + B' \cdot C' \cdot D'$$

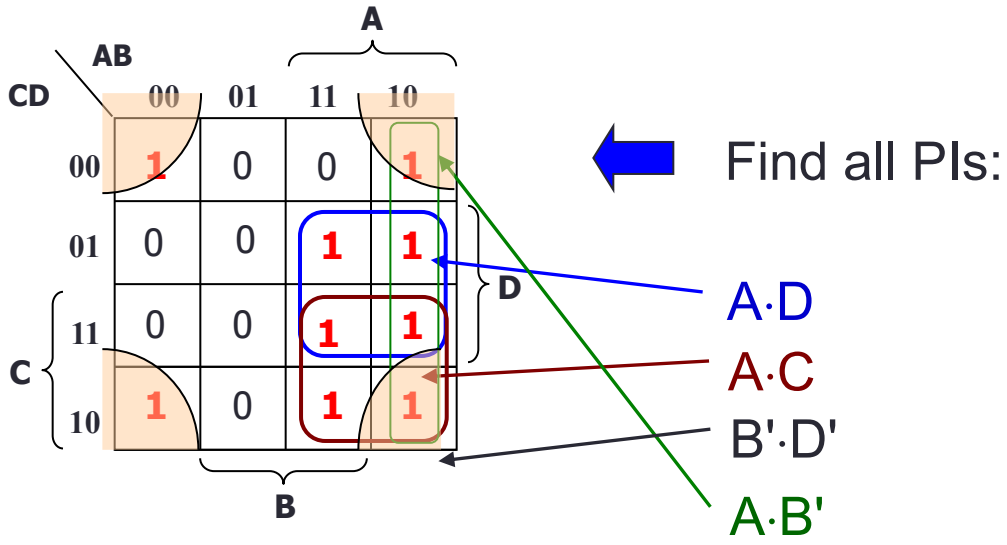


Fill in the 1's and 0's.

6. More Examples (2/6)

- Example #2:

$$F(A,B,C,D) = A{\cdot}B{\cdot}C + B'{\cdot}C{\cdot}D' + A{\cdot}D + B'{\cdot}C'{\cdot}D'$$



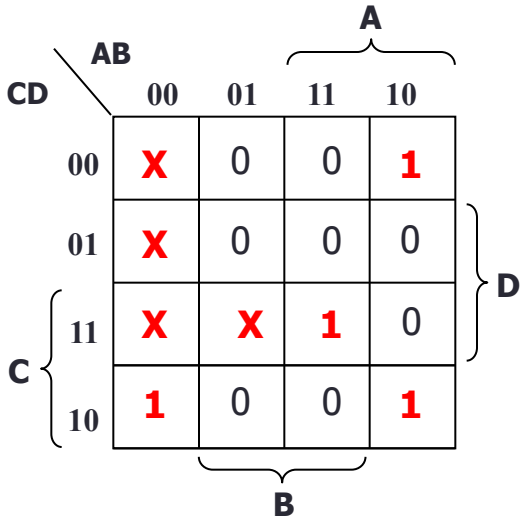
A.D, A.C and B'.D' are EPIs, and they cover all the minterms.

So the answer is: **F(A,B,C,D) = A·D + A·C + B'·D'**

6. More Examples (3/6)

- Example #3 (with don't-cares):

$$F(A,B,C,D) = \Sigma m(2,8,10,15) + \Sigma d(0,1,3,7)$$

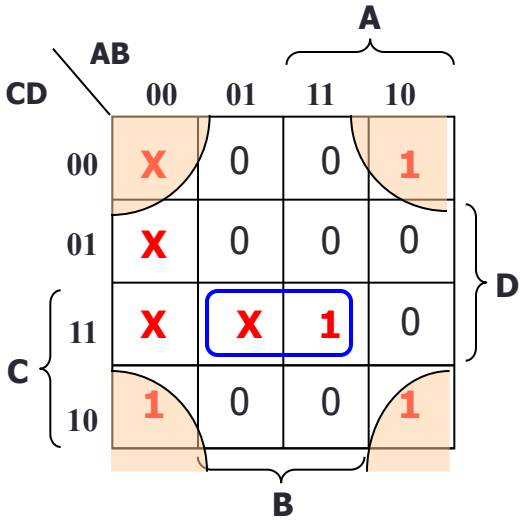


Fill in the 1's and X's.

6. More Examples (4/6)

- Example #3 (with don't-cares):

$$F(A,B,C,D) = \Sigma m(2,8,10,15) + \Sigma d(0,1,3,7)$$



Do we need to have an additional term A'·B' to cover the 2 remaining X's?

No, because all the 1's (minterms) have been covered.

Answer: **F(A,B,C,D) = B'·D' + B·C·D**

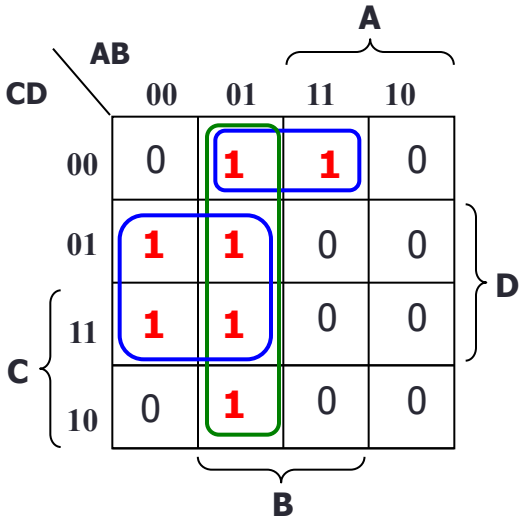
6. More Examples (5/6)

- Find the simplest POS expression for example #2:

  $F(A,B,C,D) = A{\cdot}B{\cdot}C + B'{\cdot}C{\cdot}D' + A{\cdot}D + B'{\cdot}C'{\cdot}D'$

- Draw the K-map of the complement of F, that is, F'.

K-map of F'



From K-map,

$F' = A'{\cdot}B + A'{\cdot}D + B{\cdot}C'{\cdot}D'$

Using DeMorgan's theorem,

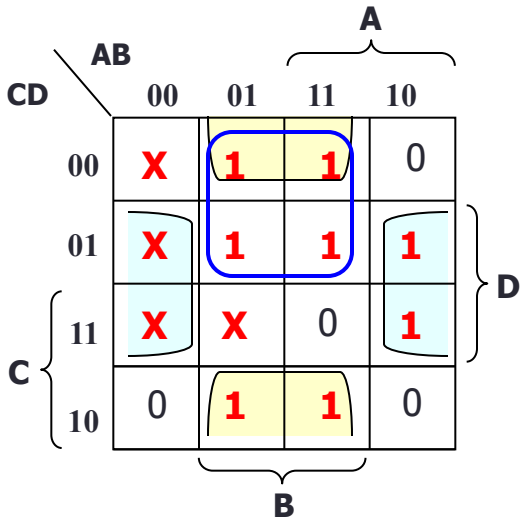
$F = (A'{\cdot}B + A'{\cdot}D + B{\cdot}C'{\cdot}D')'$

$= \mathbf{(A+B')\cdot(A+D')\cdot(B'+C+D)}$

6. More Examples (6/6)

- Find the simplest POS expression for example #3:
  $F(A,B,C,D) = \Sigma m(2,8,10,15) + \Sigma d(0,1,3,7)$
- Draw the K-map of the complement of F, that is, F'.
  $F'(A,B,C,D) = \Sigma m(4,5,6,9,11,12,13,14) + \Sigma d(0,1,3,7)$

K-map of F'



From K-map,

$F' = B \cdot C' + B \cdot D' + B' \cdot D$

Using DeMorgan's theorem,

$F = (B \cdot C' + B \cdot D' + B' \cdot D)'$

$= \mathbf{(B'+C) \cdot (B'+D) \cdot (B+D')}$

Reading

- Alternative Solutions
  - Read up DLD section 5.8, pg 101.
- Quine-McCluskey
  - Not included in syllabus, but helps in further understanding.
  - Read up DLD section 5.10, pg 103 – 105.

End of File