TEKNIK DIGITAL (A) (TI 2104)

Materi Kuliah ke-6

LOGIC SIMPLICATION

Karnaugh Maps

- Karnaugh maps (K-maps) are *graphical* representations of boolean functions.
- One ***map cell*** corresponds to a row in the truth table.
- Also, one map cell corresponds to a minterm or a maxterm in the boolean expression
- Multiple-cell areas of the map correspond to standard terms.

Two-Variable Map

$x_1 \backslash x_2$	0	1
0	0 m_0	1 m_1
1	2 m_2	3 m_3

OR

$x_2 \backslash x_1$	0	1
0	0 m_0	2 m_2
1	1 m_1	3 m_3

NOTE: ordering of variables is IMPORTANT for $f(x_1,x_2)$, x_1 is the row, x_2 is the column.

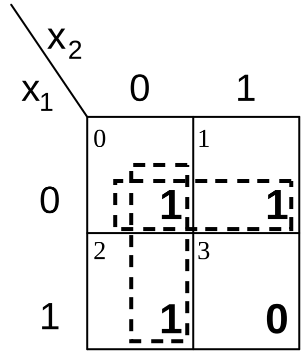
Cell 0 represents $x_1'x_2'$; Cell 1 represents $x_1'x_2$; etc. If a minterm is present in the function, then a 1 is placed in the corresponding cell.

Two-Variable Map (cont.)

- Any two adjacent cells in the map differ by ONLY one variable, which appears complemented in one cell and uncomplemented in the other.
- Example:
  m_0 ($=x_1'x_2'$) is adjacent to m_1 ($=x_1'x_2$) and m_2 ($=x_1x_2'$) but NOT m_3 ($=x_1x_2$)

2-Variable Map -- Example

- $f(x_1,x_2) = x_1'x_2' + x_1'x_2 + x_1x_2'$
  $= m_0 + m_1 + m_2$
  $= x_1' + x_2'$
- 1s placed in K-map for specified minterms m_0, m_1, m_2
- Grouping (ORing) of 1s allows simplification
- What (simpler) function is represented by each dashed rectangle?
  - $a_1' = m_0 + m_1$
  - $a_2' = m_0 + m_2$
- Note m_0 covered twice

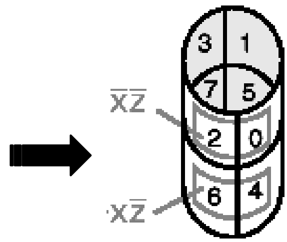


Minimization as SOP using K-map

- Enter 1s in the K-map for each product term in the function
- Group *adjacent* K-map cells containing 1s to obtain a product with fewer variables. Groups must be in power of 2 (2, 4, 8, …)
- Handle "boundary wrap" for K-maps of 3 or more variables.
- Realize that answer may not be unique

Three-Variable Map

x \ yz	00	01	11	10
0	0 m_0	1 m_1	3 m_3	2 m_2
1	4 m_4	5 m_5	7 m_7	6 m_6



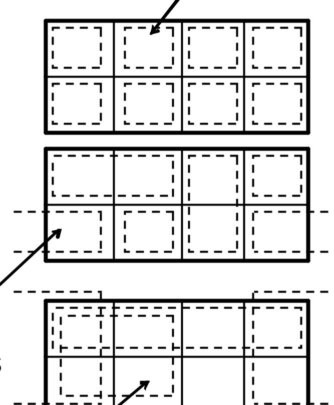
-Note: variable ordering is (x,y,z); yz specifies column, x specifies row.

-Each cell is adjacent to ***three*** other cells (left or right or top or bottom or edge wrap)

Three-Variable Map (cont.)

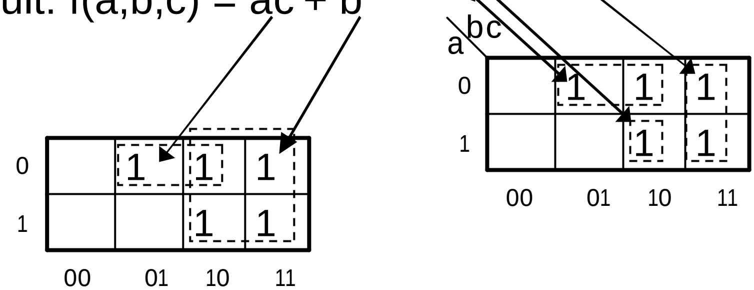
The types of structures that are either minterms or are generated by repeated application of the minimization theorem on a three variable map are shown at right.

Groups of 1, 2, 4, 8 are possible.

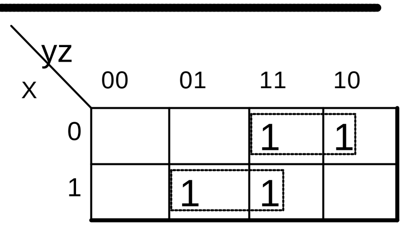


Simplification

- Enter minterms of the Boolean function into the map, then group terms
- Example: f(a,b,c) = ac’ + abc + bc’
- Result: f(a,b,c) = ac’+ b

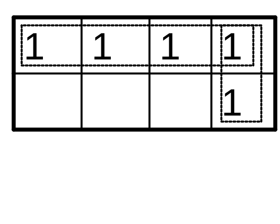


More Examples



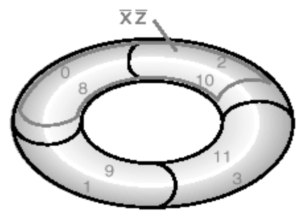
- f_1(x, y, z) = ? m(2,3,5,7)
  - f_1(x, y, z) = x’y + xz

- f_2(x, y, z) = ? m (0,1,2,3,6)
  - f_2(x, y, z) = x’+yz’



Four-Variable Maps

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_{10}	m_{11}



- Top cells are adjacent to bottom cells. Left-edge cells are adjacent to right-edge cells.
- Note variable ordering (WXYZ).

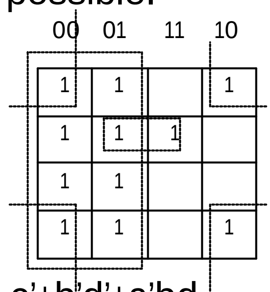
Four-variable Map Simplification

- One square represents a minterm of 4 literals.
- A rectangle of 2 adjacent squares represents a product term of 3 literals.
- A rectangle of 4 squares represents a product term of 2 literals.
- A rectangle of 8 squares represents a product term of 1 literal.
- A rectangle of 16 squares produces a function that is equal to logic 1.

Example

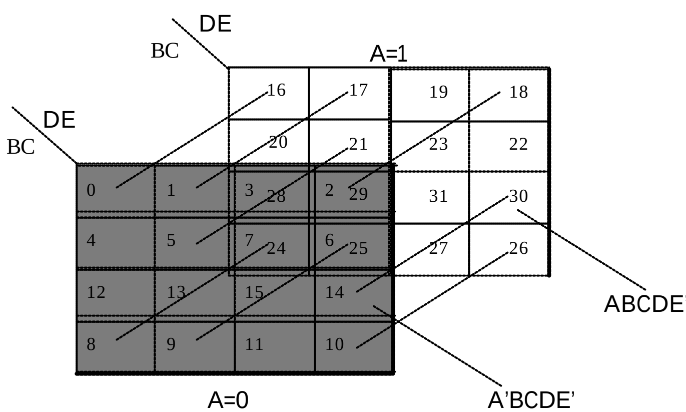
- Simplify the following Boolean function (A,B,C,D) = ? m(0,1,2,4,5,7,8,9,10,12,13).
- First put the function g( ) into the map, and then group as many 1s as possible.

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



g(A,B,C,D) = c'+b'd'+a'bd

5-Variable K-Map

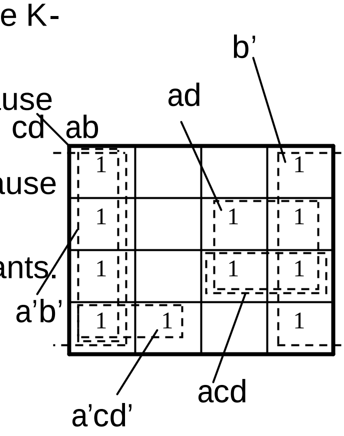


Implicants and Prime Implicants (PIs)

- An **Implicant** (P) of a function F is a product term which implies F, i.e., F(P) = 1.
- An implicant (PI) of F is called a **Prime Implicant** of F if any product term obtained by deleting a literal of PI is NOT an implicant of F
- Thus, a prime implicant is not contained in any "larger" implicant.

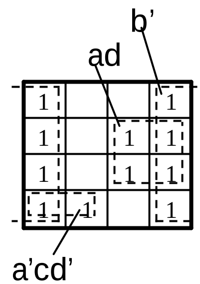
Example

- Consider function f(a,b,c,d) whose K-map is shown at right.
- *a'b'* is **not** a prime implicant because it is contained in *b'*.
- *acd* is **not** a prime implicant because it is contained in *ad*.
- b', ad, and a'cd' are prime implicants.



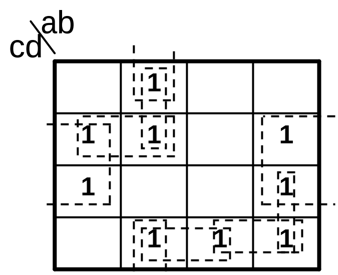
Essential Prime Implicants (EPIs)

- If a minterm of a function F is included in ONLY one prime implicant *p*, then *p* is an **essential prime implicant** of F.
- An essential prime implicant MUST appear in all possible SOP expressions of a function
- 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 b', ad, and a'cd'; all of these are essential.



Another Example

- Consider f_2(a,b,c,d), whose K-map is shown below.
- The only essential PI is b'd.

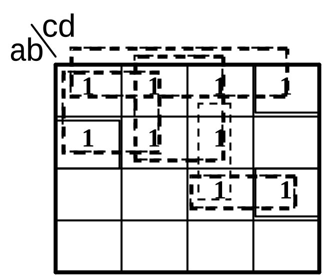


Systematic Procedure for Simplifying Boolean Functions

1. Generate **all** PIs of the function.
2. Include 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(a,b,c,d) = **?** m(0,1,2,3,4,5,7,14,15).
- Five grouped terms, not all needed.
- 3 shaded cells covered by only one term
- 3 EPIs, since each shaded cell is covered by a different term.
- F(a,b,c,d) = a’b’ + a’c’ + a’d + abc



Product of Sums Simplification

- Use sum-of-products simplification on the ***zeros*** of the function in the K-map to get F’.
- Find the complement of 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 Example

ab\cd

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

- F’(a,b,c,d) = ab’ + ac’ + a’bcd’
- Find dual of F’, dual(F’) = (a+b’)(a+c’)(a’+b+c+d’)
- Complement of literals in dual(F’) to get F
  F = (a’+b)(a’+c)(a+b’+c’+d)
  (verify that this is the same as in slide 60)

Don't Care Conditions

- There may be a combination of input values which
  - will **never** occur
  - if they do occur, the output is of no concern.
- The function value for such combinations is called a *don't care*.
- They are usually denoted with **x**. Each **x** may be arbitrarily assigned the value 0 or 1 in an implementation.
- Don't cares can be used to ***further*** simplify a function

Minimization using Don't Cares

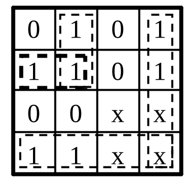
- Treat don't cares as if they are 1s to generate PIs.
- Delete PI's that cover only don't care minterms.
- Treat the covering of remaining don't care minterms as optional in the selection process (*i.e.* they may be, but need not be, covered).

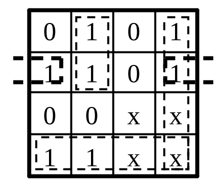
Example

- Simplify the function f(a,b,c,d) whose K-map is shown at the right.
- f = a'c'd+ab'+cd'+a'bc'

  or
- f = a'c'd+ab'+cd'+a'bd'
- 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!)

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



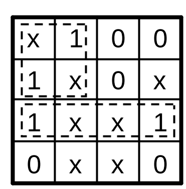


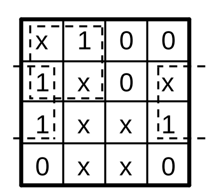
Another Example

- Simplify the function g(a,b,c,d) whose K-map is shown at right.
- g = a'c'+ ab

  or
- g = a'c'+b'd

ab \ cd				
	x	1	0	0
	1	x	0	x
	1	x	x	1
	0	x	x	0





Algorithmic minimization

- What do we do for functions with more than 4-5 variables?
- You can "code up" a minimiser (Computer-Aided Design, CAD)
  - Quine-McCluskey algorithm
  - Iterated consensus
- We won't discuss these techniques here

QUIST

Sederhanakan fungsi Boole berikut ini :

- **F = A' B' C' D' + B C' D' + A C' D'**
- **F = A + A' B C D + A' B C + A' B' C' + A' B' C**
- **G = WX'YZ' + WXY' + WXYZ + X'Y'Z**
- **H = X'YZ' + X'Y'Z + XYZ**