Third Week
Combinational Logic Design

Logic Synthesis

Overview

- Problem solution methodology
- Minimization techniques
- Algebraic Minimization
- Karnaugh Map Minimization
- Quine-McCluskey Minimization

Design Methodology

- We start with some form of a problem statement
  - Usually just text; ambiguous, poorly stated
- We must produce a design representation
  - S.A. expressions, minimized
- The primary problem we have is first to concisely define the true problem we are to solve
  - Define the “system” requirements

Design Methodology (cont.)

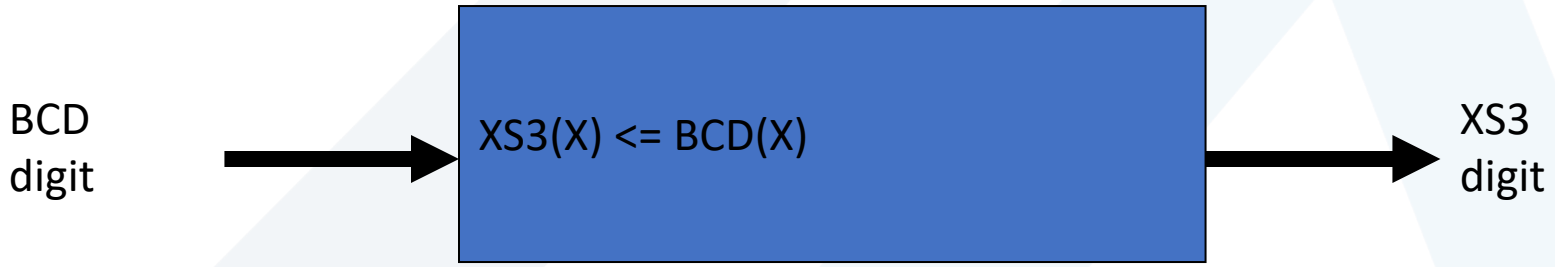
- Step 1) - Break down the problem statement
  - Identify system inputs and outputs
  - Extract the stated input-output relationship(s)
  - State the above as system (black-box) level requirements. BE PRECISE!
- Step 2) - Perform initial system definition
  - Define interface variables and representation
    - If representation is not defined in problem statement, make preliminary assignment
  - Restate I/O relationship as algorithm, equation, simulation, etc.

Design Methodology (cont.)

- Step 3) - Translate relationships to logic representation
  - Construct truth table, generate S.A. expressions
  - This is the formal statement of the I/O relationship(s)
- Step 4) - Generate minimal set of logic expressions
  - Rapid prototyping development tools
  - Karnough maps, Quine-McCluskey, etc.

BCD to XS3 Example

- Initial Statement: "Design a circuit to convert BCD to XS3."
  - 1) We need to restate and translate this to specific requirements
    - R1: The circuit shall input one BCD digit
    - R2: The circuit shall output one XS3 digit
    - R3: The XS3 output shall be the equivalent decimal value as the BCD input value



BCD to XS3 Example (cont.)

- 2) Now we need to define the interfaces in detail
  - We know that the input is one decimal digit in BCD representation, i.e. 4 bits, b_3, b_2, b_1, b_0
  - The output is one XS3 decimal digit, which is also 4 bits, x_3, x_2, x_1, x_0
  - Since these are well known representations, we don't need to explicitly define them
- 3) Now we use a truth table to define the logical input/output relationship
  - Only 10 out of 16 possible input combinations are defined
  - We'll assume last 6 won't occur; are don't cares

BCD to XS3 Example (cont.)

b_3	b_2	b_1	b_0	x_3	x_2	x_1	x_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	-	-	-	-
1	0	1	1	-	-	-	-
1	1	0	0	-	-	-	-
1	1	0	1	-	-	-	-
1	1	1	0	-	-	-	-
1	1	1	1	-	-	-	-

Note: Don't cares can work to our advantage during minimization; we can assign either 0 or 1 as needed.

BCD to XS3 Example (cont.)

- 4) Now we can generate the logical expressions for the outputs; later we will minimize them

$$x_3 = \overline{b_3}b_2\overline{b_1}b_0 + \overline{b_3}b_2b_1\overline{b_0} + \overline{b_3}b_2b_1b_0 + b_3\overline{b_2}\,\overline{b_1}\,\overline{b_0} + b_3\overline{b_2}\,\overline{b_1}b_0$$

$$x_2 = \overline{b_3}\,\overline{b_2}\,\overline{b_1}b_0 + \overline{b_3}\,\overline{b_2}b_1\overline{b_0} + \overline{b_3}\,\overline{b_2}b_1b_0 + \overline{b_3}b_2\overline{b_1}\,\overline{b_0} + b_3\overline{b_2}\,\overline{b_1}b_0$$

$$x_1 = \overline{b_3}\,\overline{b_2}\,\overline{b_1}\,\overline{b_0} + \overline{b_3}\,\overline{b_2}b_1b_0 + \overline{b_3}b_2\overline{b_1}\,\overline{b_0} + \overline{b_3}b_2b_1b_0 + b_3\overline{b_2}\,\overline{b_1}\,\overline{b_0}$$

$$x_0 = \overline{b_3}\,\overline{b_2}\,\overline{b_1}\,\overline{b_0} + \overline{b_3}\,\overline{b_2}b_1\overline{b_0} + \overline{b_3}b_2\overline{b_1}\,\overline{b_0} + \overline{b_3}b_2b_1\overline{b_0} + b_3\overline{b_2}\,\overline{b_1}\,\overline{b_0}$$

Logic Expression Minimization

- Goal is to find an equivalent of an original logic expression that:
  - a) has fewer variables per term
  - b) has fewer terms
  - c) needs less logic to implement
- There are three main manual methods
  - Algebraic minimization
  - Karnaugh Map minimization
  - Quine-McCluskey (tabular) minimization

Algebraic Minimization

- Process is to apply the switching algebra postulates, laws, and theorems to transform the original expression
  - Hard to recognize when a particular law can be applied
  - Difficult to know if resulting expression is truly minimal
  - Very easy to make a mistake
    - Incorrect complementation
    - Dropped variables

Switching Algebra Laws and Theorems

Involution:

$$x = \left(\overline{\overline{x}}\right)$$

Switching Algebra Laws and Theorems

Identity:

$$X + 1 = 1 \qquad X \cdot 0 = 0$$

$$X + 0 = X \qquad X \cdot 1 = X$$

Switching Algebra Laws and Theorems

Idempotence:

$$X + X = X \qquad X \cdot X = X$$

Switching Algebra Laws and Theorems

Associativity:

$$X + (Y + Z) = (X + Y) + Z$$

$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$$

Switching Algebra Laws and Theorems

DeMorgan's Theorem:

$$\overline{X + Y} = \overline{X} \cdot \overline{Y}$$

$$\overline{X \cdot Y} = \overline{X} + \overline{Y}$$

General form:

$$\overline{F(\cdot, +, X_1, \ldots X_n)} = G(+, \cdot, \overline{X_1}, \ldots \overline{X_n})$$

Switching Algebra Laws and Theorems

Adjacency:

$$X \cdot Y + X \cdot \overline{Y} = X$$

$$(X + Y) \cdot (X + \overline{Y}) = X$$

Switching Algebra Laws and Theorems

Absorption:

$$X + (X \cdot Y) = X$$

$$X \cdot (X + Y) = X$$

Switching Algebra Laws and Theorems

Simplification:

$$X + (\overline{X} \cdot Y) = X + Y$$

$$X \cdot (\overline{X} + Y) = X \cdot Y$$

Switching Algebra Laws and Theorems

Consensus:

$$X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$$

$$(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z) = (X + Y) \cdot (\overline{X} + Z)$$

DeMorgan's Theorem

Very useful for complementing function expressions:

e.g.

$$F = X + Y \cdot Z; \qquad \overline{F} = \overline{X + Y \cdot Z}$$

$$\overline{F} = \overline{X} \cdot \overline{Y \cdot Z} \qquad F = \overline{X} \cdot \left(\overline{Y} + \overline{Z}\right)$$

$$\overline{F} = \overline{X} \cdot \overline{Y} + \overline{X} \cdot \overline{Z}$$

Minimization via Adjacency

- Adjacency is easy to use; very powerful
  - Look for two terms that are identical except for one variable

$$\text{e.g. } A \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D$$

  - Application removes one term and one variable from the remaining term

$$A \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D = A \cdot B \cdot C$$

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

$$(A \cdot B \cdot C) \cdot (\overline{D} + D) = (A \cdot B \cdot C) \cdot 1 = A \cdot B \cdot C$$

Example of Adjacency Minimization

$$x_3 = \overline{b_3} b_2 \overline{b_1} b_0 + \overline{b_3} b_2 b_1 \overline{b_0} + \overline{b_3} b_2 b_1 b_0 + b_3 \overline{b_2} \overline{b_1} \overline{b_0} + b_3 \overline{b_2} \overline{b_1} b_0$$

Duplicate m7 and rearrange

$$x_3 = \overline{b_3} b_2 \overline{b_1} b_0 + \overline{b_3} b_2 b_1 b_0 + \overline{b_3} b_2 b_1 \overline{b_0} + \overline{b_3} b_2 b_1 b_0 + b_3 \overline{b_2} \overline{b_1} \overline{b_0} + b_3 \overline{b_2} \overline{b_1} b_0$$

Apply adjacency on term pairs

$$x_3 = \overline{b_3} b_2 b_0 + \overline{b_3} b_2 b_1 + b_3 \overline{b_2} \overline{b_1}$$

Karnaugh Map Minimization

- Karnaugh Map (or K-map) minimization is a visual minimization technique
  - Is an application of adjacency
  - Procedure guarantees a minimal expression
  - Easy to use; fast
  - Problems include:
    - Applicable to limited number of variables (4 ~ 8)
    - Errors in translation from TT to K-map
    - Errors in reading final expression

K-map Minimization (cont.)

- Basic K-map is a 2-D rectangular array of cells
  - Each K-map represents one bit column of output
  - Each cell contains one bit of output function
- Arrangement of cells in array facilitates recognition of adjacent terms
  - Adjacent terms differ in one variable value; equivalent to difference of one bit of input row values
    - e.g. m6 (110) and m7 (111)

Truth Table Rows and Adjacency

Standard TT ordering doesn't show adjacency

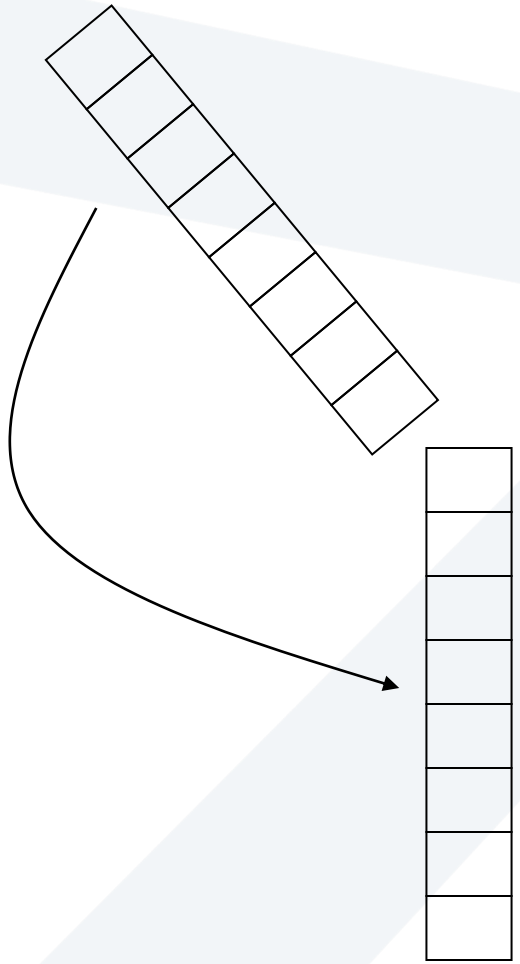
A	B	C	D	minterm
0	0	0	0	m0
0	0	0	1	m1
0	0	1	0	m2
0	0	1	1	m3
0	1	0	0	m4
0	1	0	1	m5
0	1	1	0	m6
0	1	1	1	m7
1	0	0	0	m8
1	0	0	1	m9
1	0	1	0	m10
1	0	1	1	m11
1	1	0	0	m12
1	1	0	1	m13
1	1	1	0	m14
1	1	1	1	m15

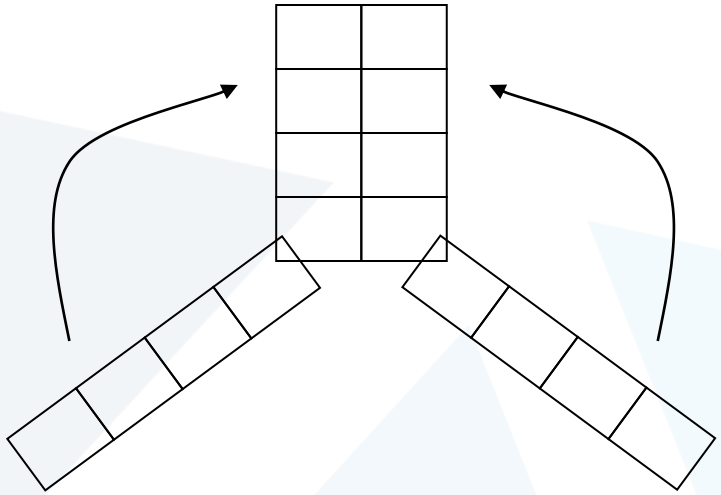
Key is to use gray code for row order

A	B	C	D	minterm
0	0	0	0	m0
0	0	0	1	m1
0	0	1	1	m3
0	0	1	0	m2
0	1	1	0	m6
0	1	1	1	m7
0	1	0	1	m5
0	1	0	0	m4
1	1	0	0	m12
1	1	0	1	m13
1	1	1	1	m15
1	1	1	0	m14
1	0	1	0	m10
1	0	1	1	m11
1	0	0	1	m9
1	0	0	0	m8

Folding of Gray Code Truth Table into K-map

ABCD	
0000	
0001	
0011	
0010	
0110	
0111	
0101	
0100	
1100	
1101	
1111	
1110	
1010	
1011	
1001	
1000	

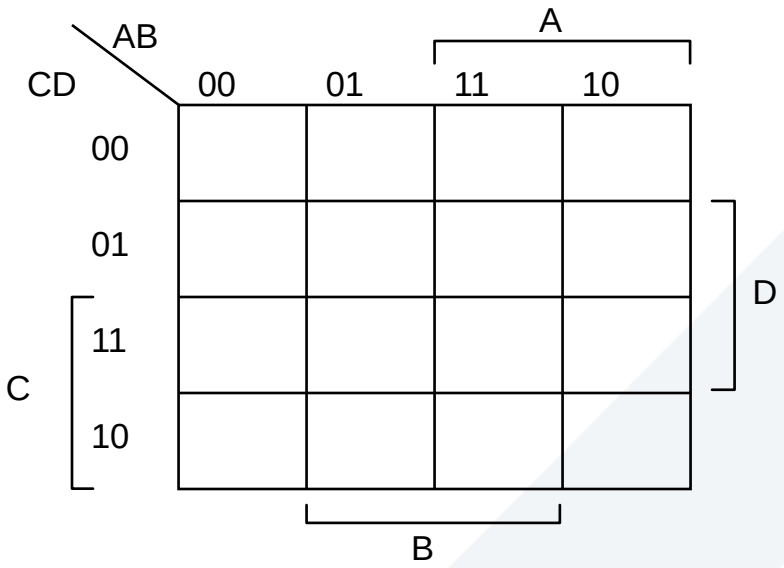




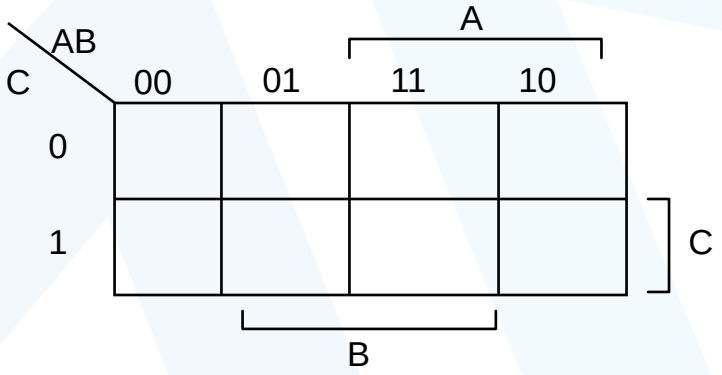
CD \ AB	00	01	11	10
00				
01				
11				
10				

K-map Minimization (cont.)

- For any cell in 2-D array, there are four direct neighbors (top, bottom, left, right)
- 2-D array can therefore show adjacencies of up to four variables.



Four variable K-map

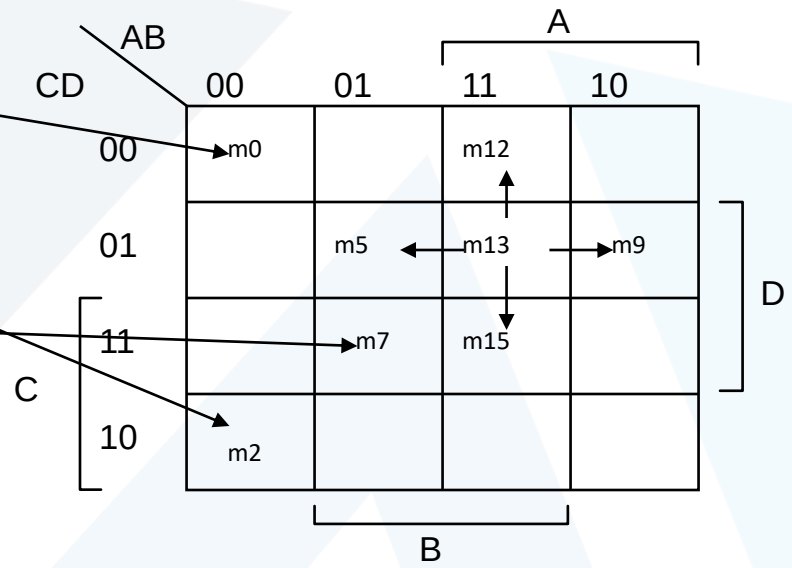


Three variable K-map

Truth Table to K-map

Number of TT rows MUST match number of K-map cells

A	B	C	D	F	
0	0	0	0		x
0	0	0	1		x
0	0	1	0		x
0	0	1	1		x
0	1	0	0		x
0	1	0	1		x
0	1	1	0		x
0	1	1	1		x
1	0	0	0		x
1	0	0	1		x
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

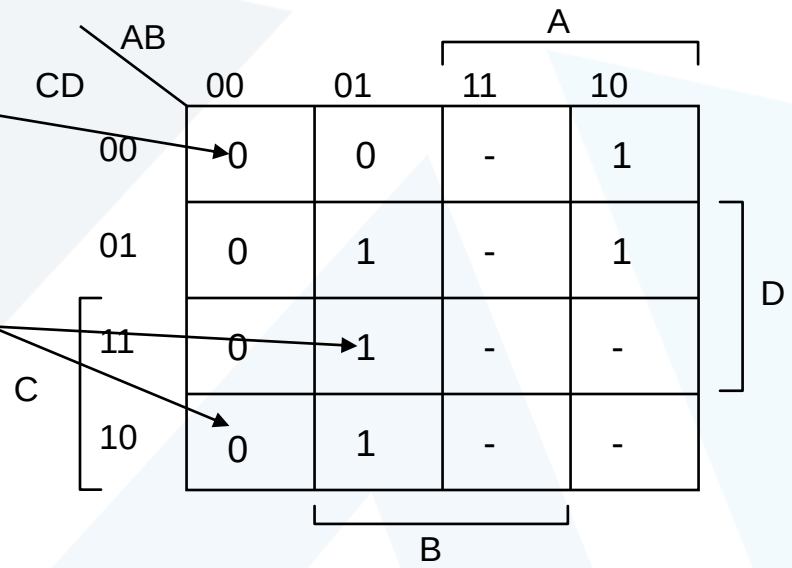


Note different ways K-map is labeled

K-map Minimization of X_3

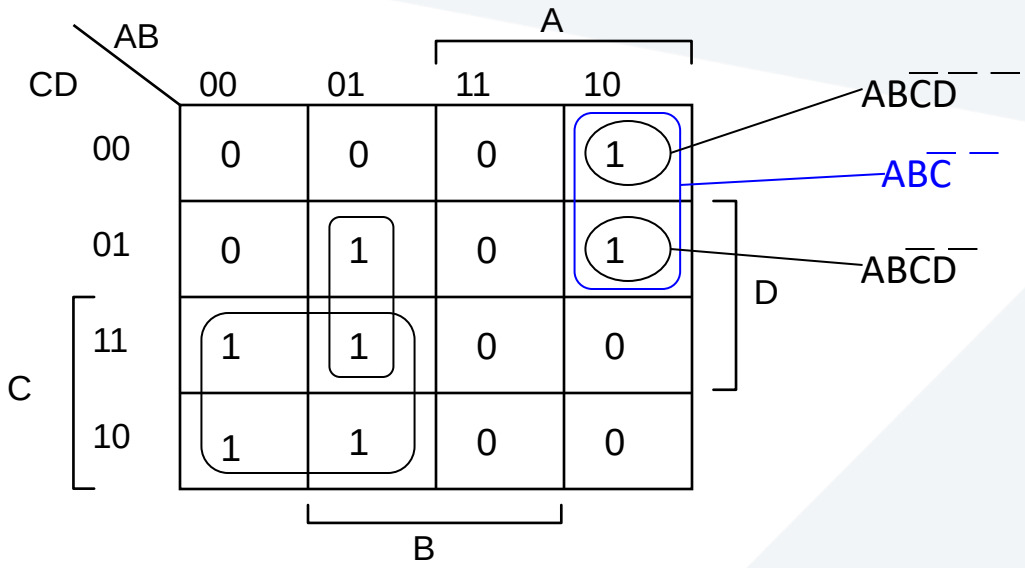
Entry of TT data into K-map

b3	b2	b1	b0	x3
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	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	-
1	1	1	1	-



Watch out for ordering of 10 and 11 rows and columns!

Grouping - Applying Adjacency



If two cells have the same value and are next to each other, the terms are adjacent.

This adjacency is shown by enclosing them.

Groups can have common cells.

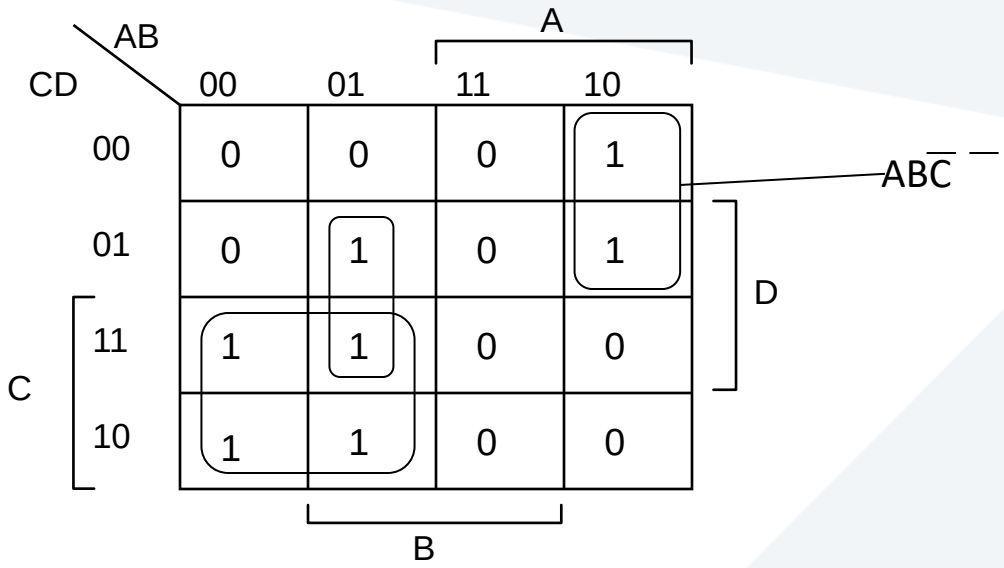
Group size is a power of 2 and groups are rectangular.

You can group 0s or 1s.

Reading the Groups

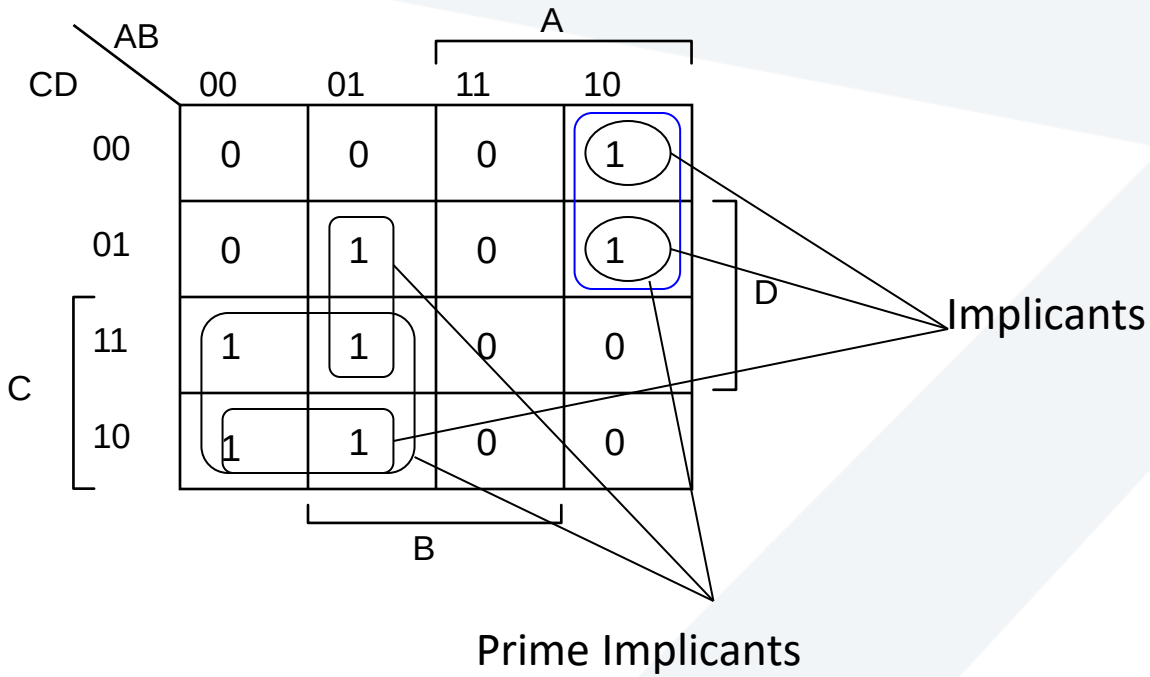
If 1s grouped, the expression is a product term, 0s - sum term.

Within group, note when variable values change as you go cell to cell. This determines how the term expression is formed by the following table



	Grouping 1s	Grouping 0s
Variable changes	Exclude	Exclude
Variable constant 0	Inc. comp.	Inc. true
Variable constant 1	Inc. true	Inc. comp.

Implicants and Prime Implicants



Single cells or groups that could be part of a larger group are know as implicants

A group that is as large as possible is a prime implicant

Single cells can be prime implicants is they cannot be grouped with any other cell

Implicants and Minimal Expressions

- Any set of implicants that encloses (covers) all values is “sufficient”; i.e. the associated logical expression represents the desired function.
  - All minterms or maxterms are sufficient.
- The smallest set of prime implicants that covers all values forms a minimal expression for the desired function.
  - There may be more than one minimal set.

Essential and Secondary Prime Implicants

- If a prime implicant has any cell that is not covered by any other prime implicant, it is an “essential prime implicant”
- If a prime implicant is not essential is is a “secondary prime implicant”
- A minimal set MUST include all essential prime implicants and the minimum number of secondary PIs as needed to cover all values.

K-map Minimization Method

- Technique is valid for either 1s or 0s

A) Find all prime implicants (largest groups of 1s or 0s in order of largest to smallest)

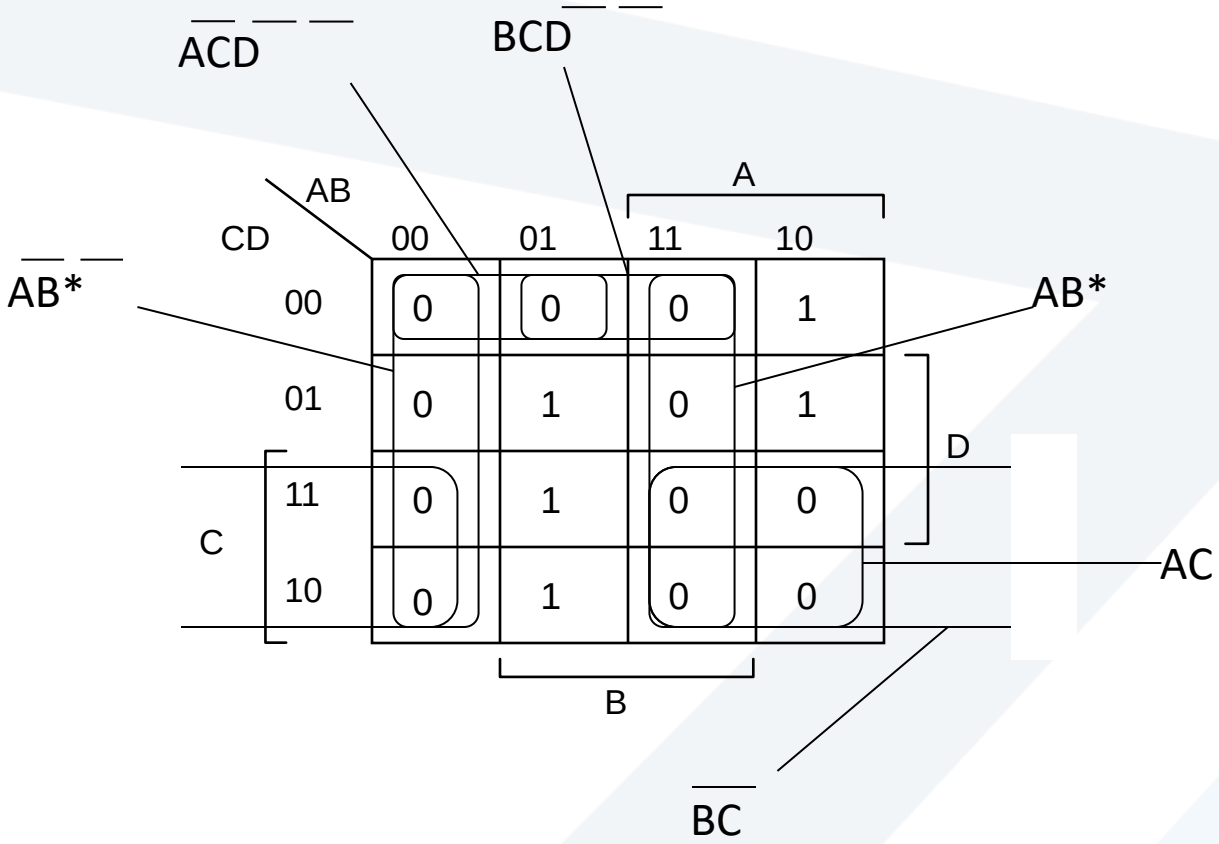
B) Identify minimal set of PIs

1) Find all essential PIs

2) Find smallest set of secondary PIs

The resulting expression is minimal.

K-map Minimization of X_3 (CONT.)



Assume output is active-0. We need output of comp. function; circle 0s

* PIs are essential; what's left is set of secondary PIs.

There are four minimal solutions. These are:

$$F = AB + \overline{A}\,\overline{B} + \overline{B}C + \overline{A}C\overline{D}$$

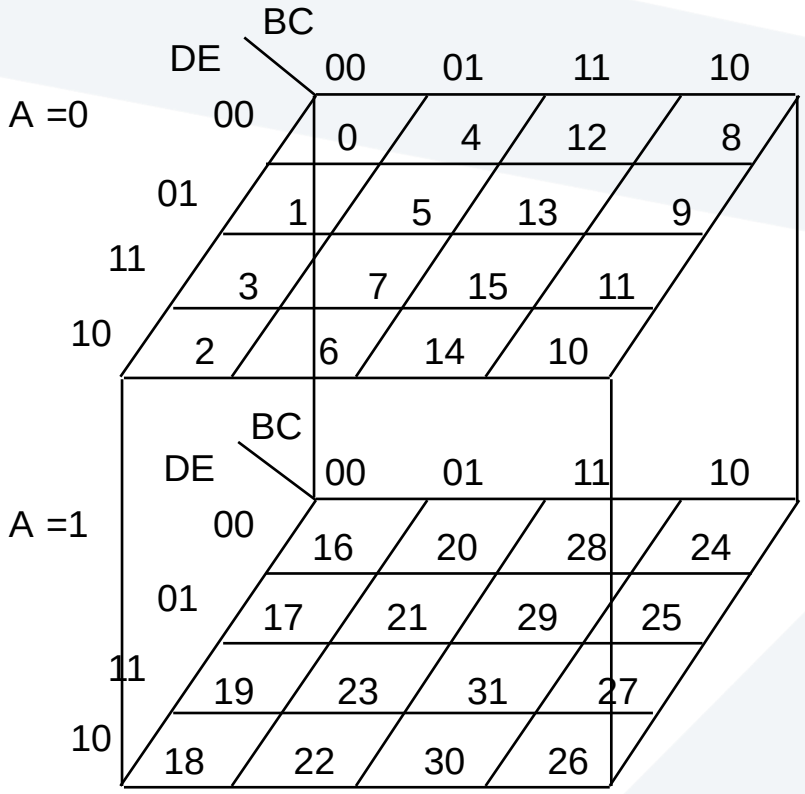
$$F = AB + \overline{A}\,\overline{B} + AC + \overline{A}C\overline{D}$$

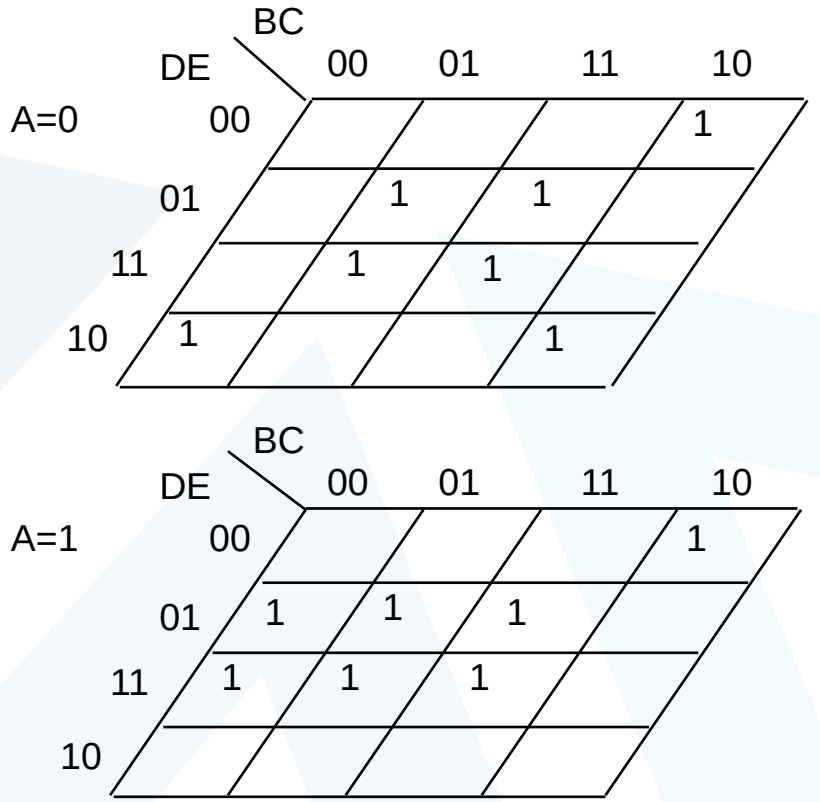
$$F = AB + \overline{A}\,\overline{B} + \overline{B}C + B\overline{C}\,\overline{D}$$

$$F = AB + \overline{A}\,\overline{B} + AC + B\overline{C}\,\overline{D}$$

Gate Logic: Two-Level Simplification

5-Variable K-maps

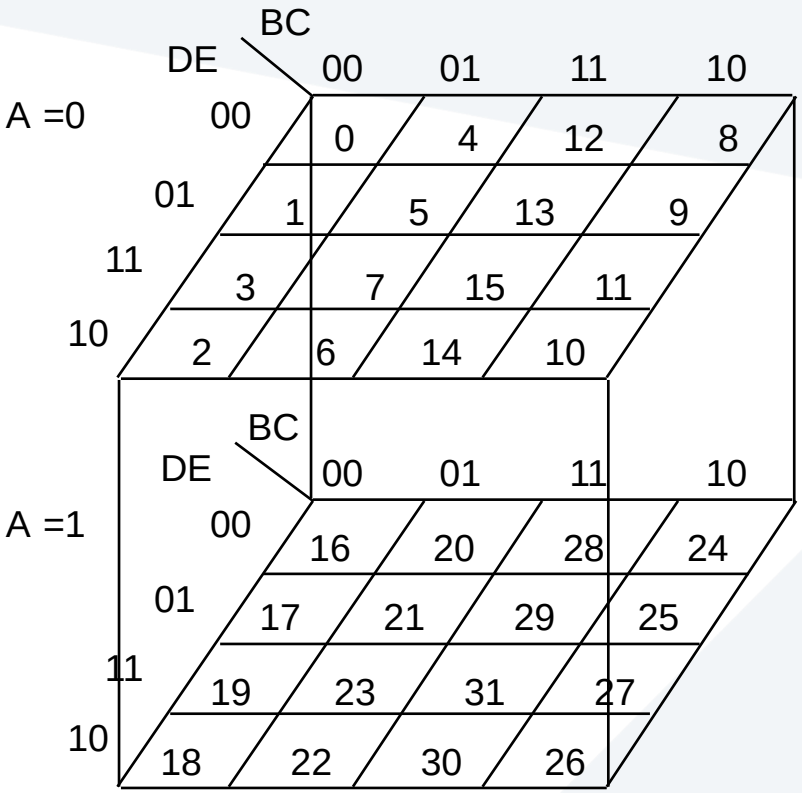


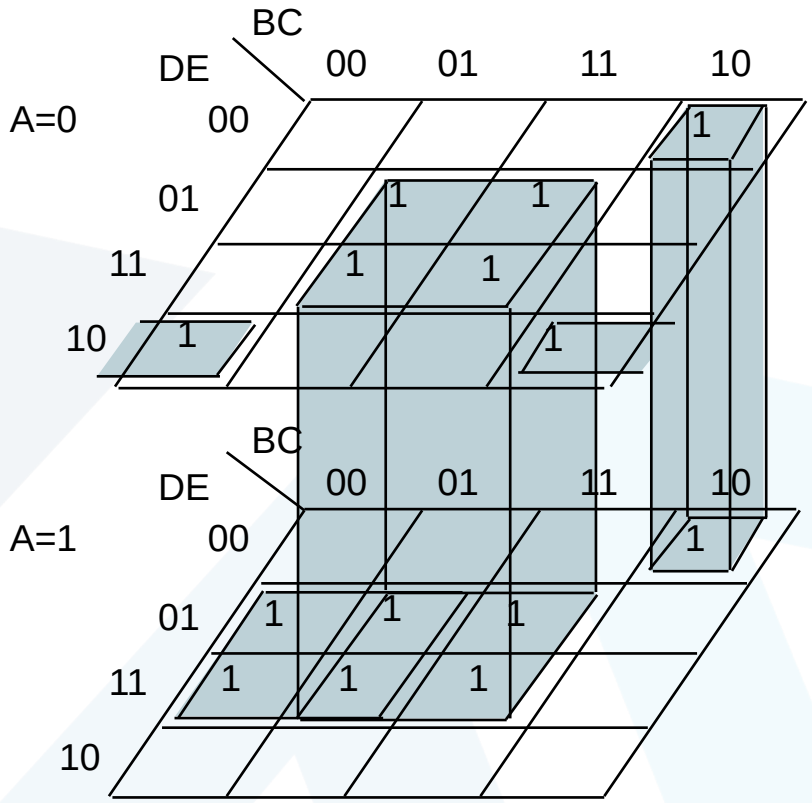


f(A,B,C,D,E) = Σm(2,5,7,8,10, 13,15,17,19,21,23,24,29 31)

Gate Logic: Two-Level Simplification

5-Variable K-maps

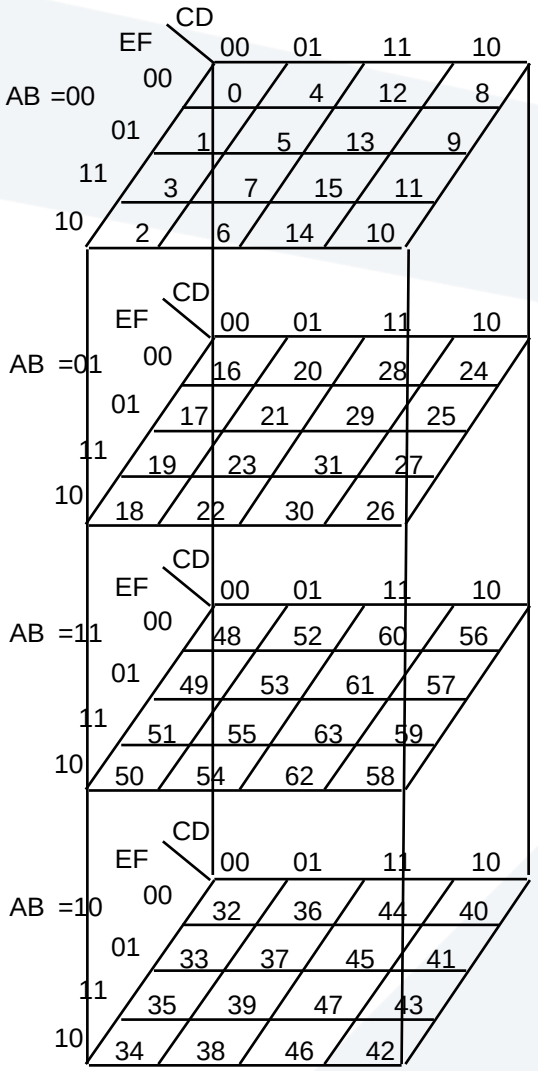




$f(A,B,C,D,E) = \Sigma m(2,5,7,8,10, 13,15,17,19,21,23,24,29\ 31)$

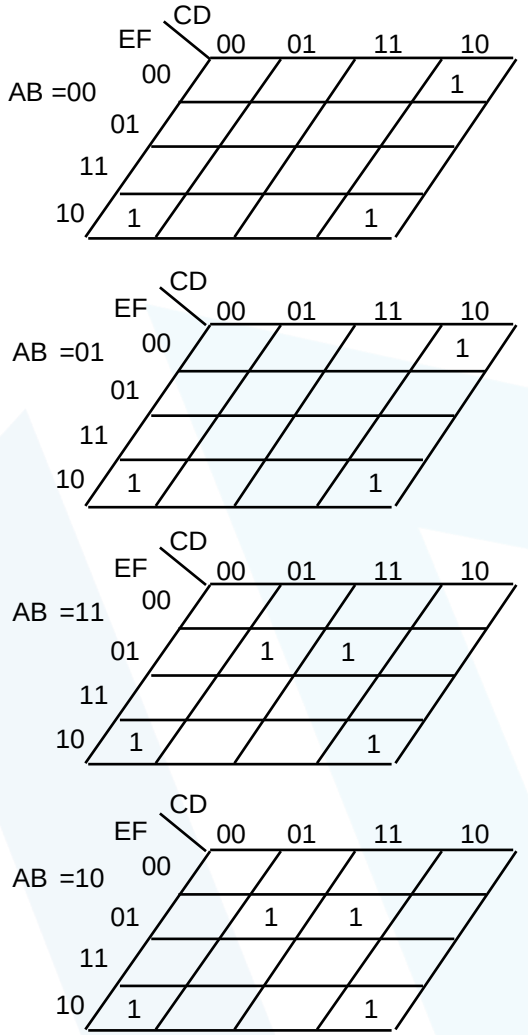
$= C\ E + A\ B'\ E + B\ C'\ D'\ E' + A'\ C'\ D\ E'$

Gate Logic: Two Level Simplification

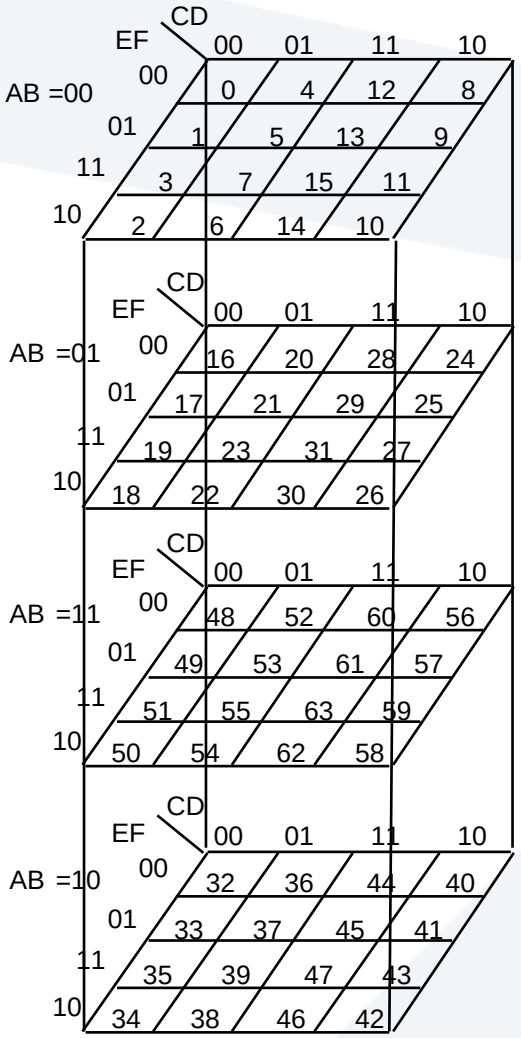


6- Variable K-Maps

$$f(A,B,C,D,E,F) = \Sigma m(2,8,10,18,24,26,34,37,42,45,50,53,58,61)$$



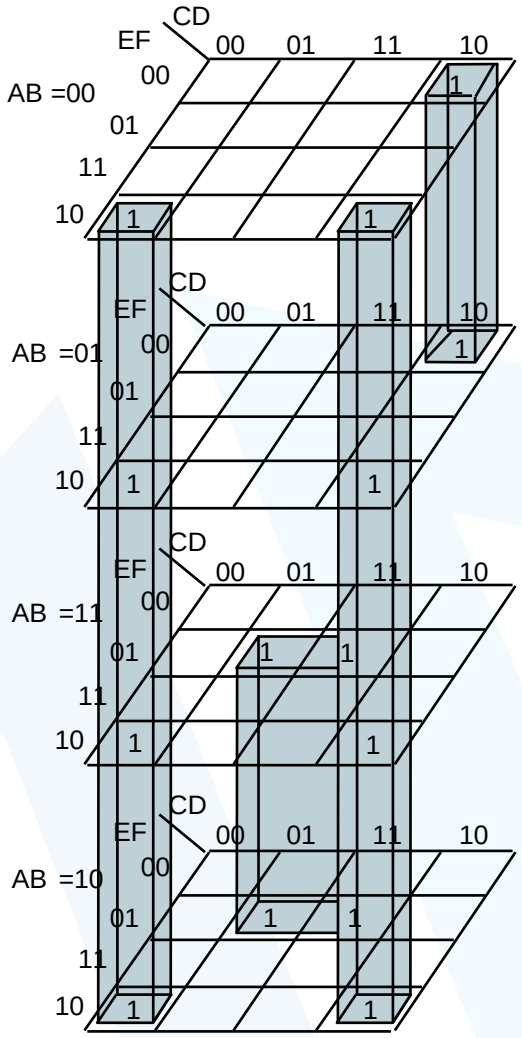
Gate Logic: Two Level Simplification



6- Variable K-Maps

$f(A,B,C,D,E,F) = \Sigma m(2,8,10,18,24,26,34,37,42,45,50,53,58,61)$

= D' E F' + A D E' F + A' C D' F'



Quine-McCluskey (Tabular) Minimization

- Two step process utilizing tabular listings to
  - Identify prime implicants (implicant tables)
  - Identify minimal cover (cover tables)
- All work is done in tabular form
  - Number of variables is not a limitation
  - Basis for many computer implementations
- Proper organization and term identification are key factors for correct results

Quine-McCluskey Minimization (cont.)

- Terms are initially listed one per line in groups
  - Each group contains terms with the same number of true and complemented variables
  - Terms are listed in numerical order within group
- Terms and implicants are identified using one of three common notations
  - full variable form
  - cellular form
  - 10- form

Notation Forms

- Full variable form - variables and complements in algebraic form
  - hard to identify when adjacency applies
  - very easy to make mistakes
- Cellular form - terms are identified by their decimal index value
  - Easy to tell when adjacency applies; indexes must differ by power of two (one bit)
  - Implicants identified by term nos. separated by comma; differing bit pos. in () following terms

Notation Forms (cont.)

- 10- form - terms are identified by their binary index value
  - Easier to translate to/from full variable form
  - Easy to identify when adjacency applies, one bit is different
  - - shows variable(s) dropped when adjacency is used
- Different forms may be mixed during the minimization

Example of Different Notations

$F(A, B, C, D) = \Sigma\, m(4,5,6,8,10,13)$

	Full variable	Cellular	10-	
1	$A\bar{B}CD$	4		0100
	$A\bar{B}\bar{C}\bar{D}$	8		1000
2	$A\bar{B}CD$	5		0101
	$A\bar{B}C\bar{D}$	6		0110
	$AB\bar{C}D$	10		1010
3	$AB\bar{C}\bar{D}$	11		1011

Implication Table

Quine-McCluskey Method

Tabular method to systematically find all prime implicants

$$f(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) + \Sigma d(0,7,15)$$

Stage 1: Find all prime implicants

Step 1: Fill Column 1 with active-set and DC-set minterm indices. Group by number of true variables.

Implication Table	
Column I	
0000	
0100	
1000	
0101	
0110	
1001	
1010	
0111	
1101	
1111	

Minimization – First Pass

Quine-McCluskey Method

Tabular method to systematically find all prime implicants

$$f(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) + \Sigma d(0,7,15)$$

Stage 1: Find all prime implicants

Step 2: Apply Adjacency - Compare elements of group w/N 1's against those with N+1 1's. Differ by one bit implies adjacent. Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00
0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.
Repeat until nothing left.

Implication Table

Column I	Column II
0000 ☐	0-00
	-000
0100 ☐	
1000 ☐	010-
	01-0
0101 ☐	100-
0110 ☐	10-0
1001 ☐	
1010 ☐	01-1
	-101
0111 ☐	011-
1101 ☐	1-01
1111 ☐	-111
	11-1

Minimization - Second Pass

Step 2: Apply Adjacency Theorem—
Compare elements of group w/ N 1's against those with N+1 1's. Differ by one bit implies adjacent. Eliminate variable and place in next column. - must be align

E.g., 000- vs. 011- yields 01--
01-0 vs. 01-1 yields 01--

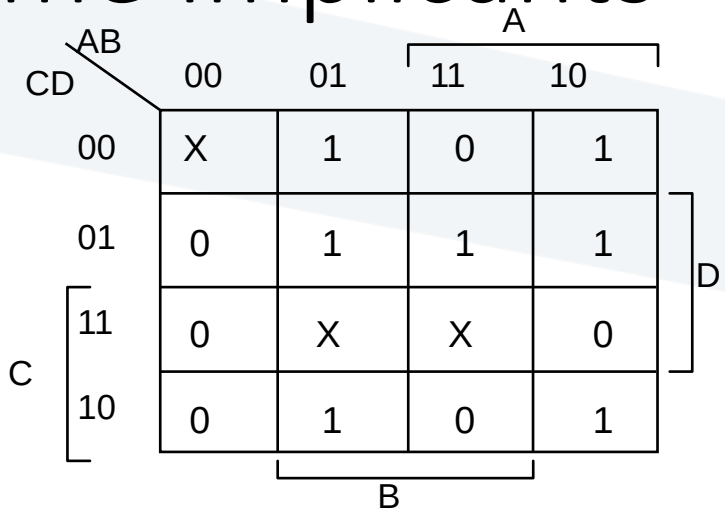
When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Repeat until no further combinations can be made.

Implication Table

Column I	Column II	Column III
0000 □	0-00 *	01-- *
	-000 *	
0100 □		-1-1 *
1000 □	010- □	
	01-0 □	
0101 □	100- *	
0110 □	10-0 *	
1001 □		
1010 □	01-1 □	
	-101 □	
0111 □	011- □	
1101 □	1-01 *	
1111 □	-111 □	
	11-1 □	

Prime Implicants



Prime Implicants:

0-00 = A' C' D'

100- = A B' C'

1-01 = A C' D

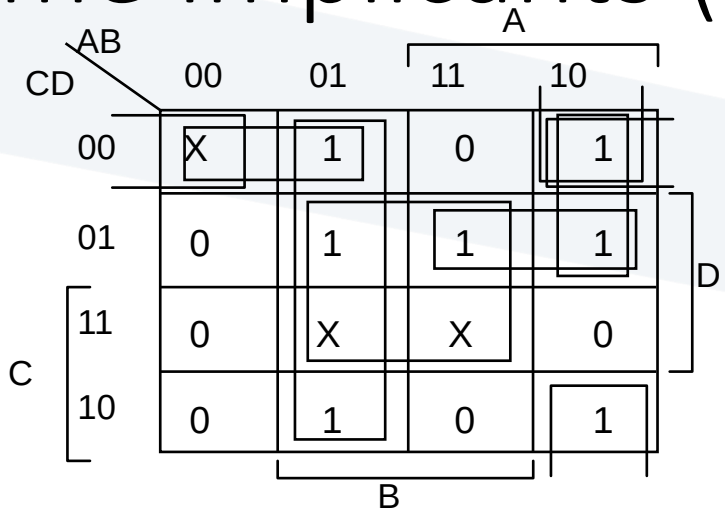
-1-1 = B D

-000 = B' C' D'

10-0 = A B' D'

01-- = A' B

Prime Implicants (cont.)



Prime Implicants:

0-00 = A' C' D'

-000 = B' C' D'

100- = A B' C'

10-0 = A B' D'

1-01 = A C' D

01-- = A' B

-1-1 = B D

Stage 2: find smallest set of prime implicants that cover the active-set

recall that essential prime implicants must be in final expression

Coverage Table

Coverage Chart

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(1000)				X			
8,9(100-)				X	X		
8,10(10-0)				X		X	
9,13(1-01)	X				X		X
4,5,6,7(01--)		X	X				
5,7,13,15(-1-1)		X					X

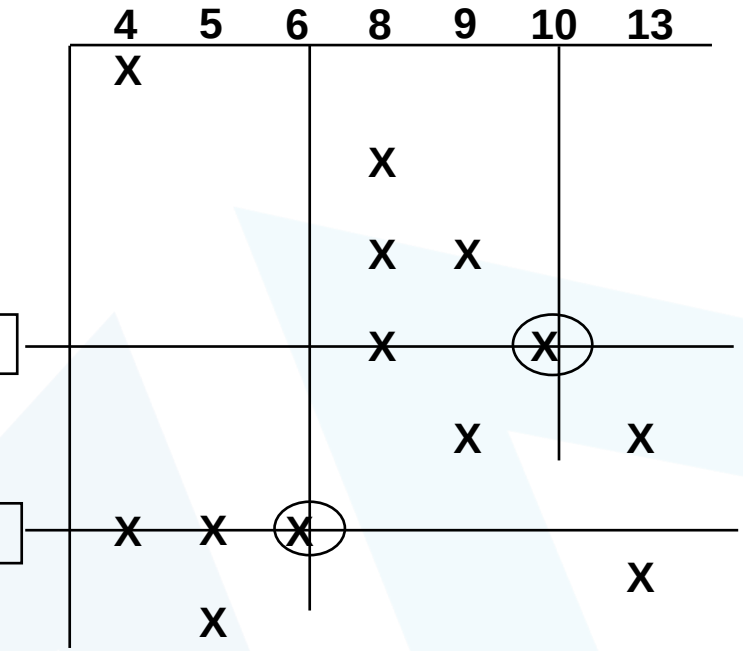
rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element is covered by the prime implicant

Note: Don't include DCs in coverage chart; they don't have covered by the final logic expression

Coverage Table (cont.)

Coverage Chart

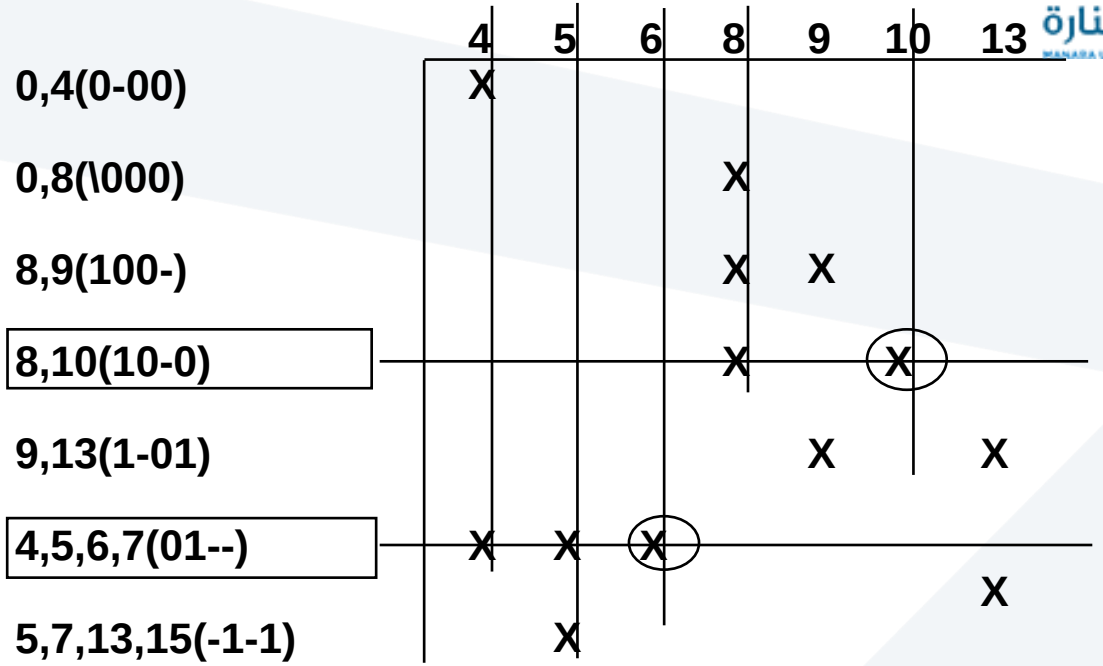
	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(\000)				X			
8,9(100-)				X	X		
8,10(10-0)				X		X	
9,13(1-01)					X		X
4,5,6,7(01--)	X	X	X				
5,7,13,15(-1-1)		X					X



rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element is covered by the prime implicant

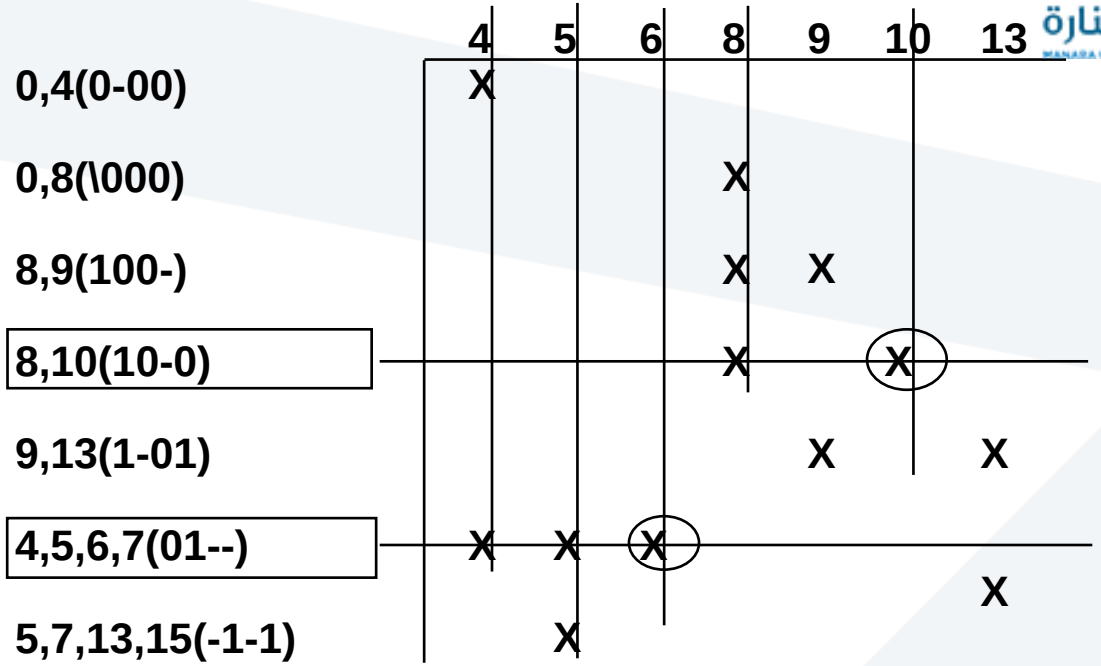
If column has a single X, than the implicant associated with the row is essential. It must appear in minimum cover

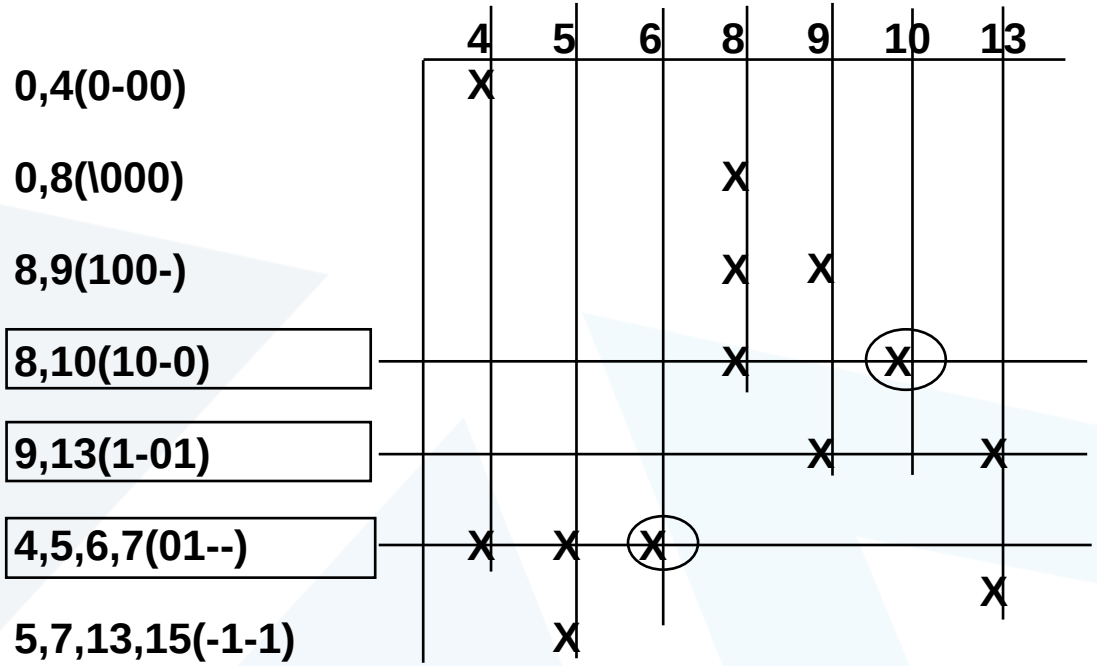
Coverage Table (cont.)



Eliminate all columns covered by essential primes

Coverage Table (cont.)





Eliminate all columns covered by essential primes

Find minimum set of rows that cover the remaining columns

$$f = A\,B'\,D' + A\,C'\,D + A'\,B$$