Chapter 2
Combinational Logic Circuits

CSCM601150, 2022/2023 – 1

Instructor: Erdefi Rakun dan Tim Dosen PSD

Fasilkom UI

Outline:

- Binary Logic and Gates
- Boolean Algebra
- Standard Form
- Two-level Circuit Optimization
- Map Minipulation
- Exclusive-Or Operator and Gates

Note: Portion of this material are taken from Aaron Tan's slide and other portions of this material © 2008 by Pearson Education,Inc

2-1 Binary Logic and Gates

Digital Circuits

Advantages of digital circuits over analog circuits

- More reliable (simpler circuits, less noise-prone)
- Specified accuracy (determinable)
- Binary variables take on one of two values.
- Logical operators operate on binary values and binary variables.
- Basic logical operators are the logic functions AND, OR and NOT.
- Logic gates implement logic functions.

Boolean Algebra

- Boolean Algebra: a useful mathematical system for specifying and transforming logic functions.
- We study Boolean algebra as a foundation for designing and analyzing digital systems!

Boolean Algebra

- Boolean values:
  - True (1)
  - False (0)
- Connectives
  - Conjunction (AND)
    - A • B; A ∧ B
  - Disjunction (OR)
    - A + B; A ∨ B
  - Negation (NOT)
    - $\bar{A}$; ¬A; A'

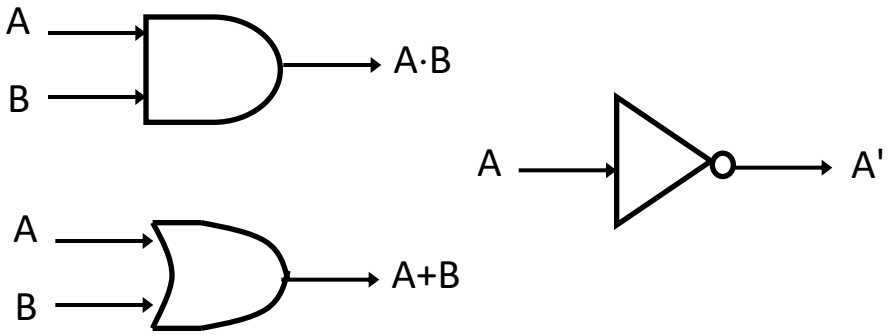
- Truth tables

A	B	A • B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

A	A'
0	1
1	0

- Logic gates



Notation Examples

- Examples:

  $\mathbf{Y = A \times B}$ is read "Y is equal to A AND B."

  $\mathbf{z = x + y}$ is read "z is equal to x OR y."

  $\mathbf{X = \overline{A}}$ is read "X is equal to NOT **A.**"

- Note: The statement:

  1 + 1 = 2 (read "one plus one equals two")

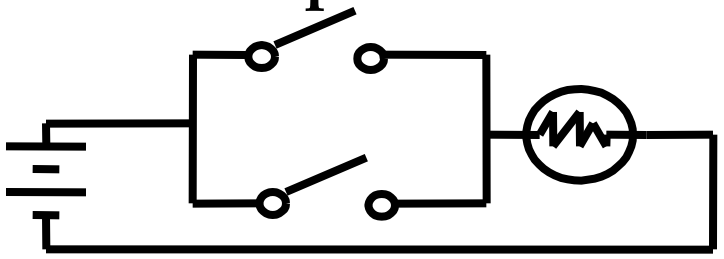
  is not the same as

  1 + 1 = 1 (read "1 or 1 equals 1").

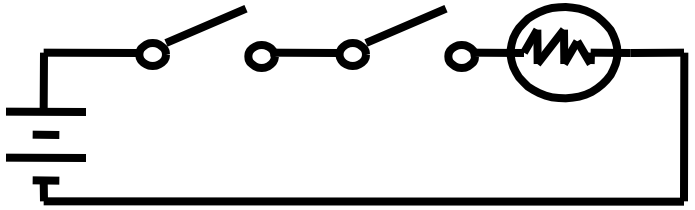
Logic Function Implementation

- Using Switches
  - For inputs:
    - logic 1 is switch closed
    - logic 0 is switch open
  - For outputs:
    - logic 1 is light on
    - logic 0 is light off.
  - NOT uses a switch such that:
    - logic 1 is switch open
    - logic 0 is switch closed

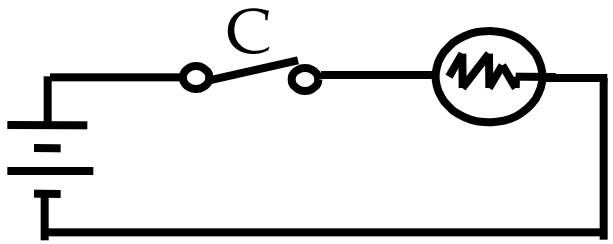
Switches in parallel => OR



Switches in series => AND

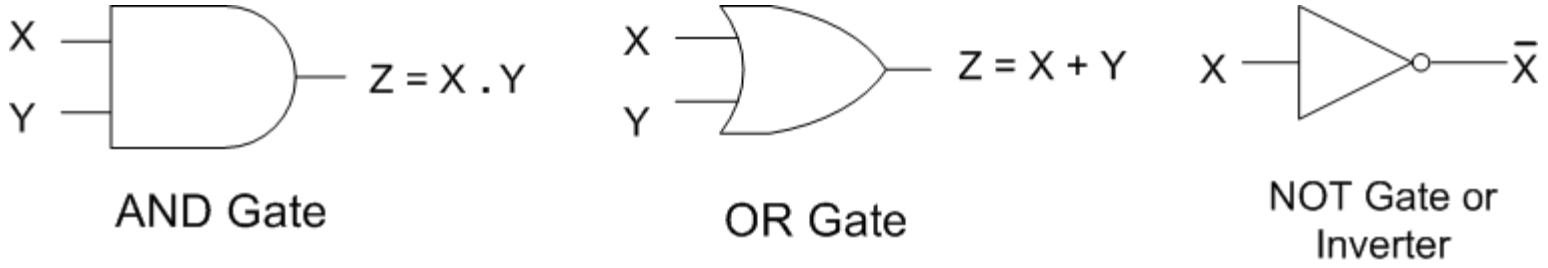


Normally-closed switch => NOT

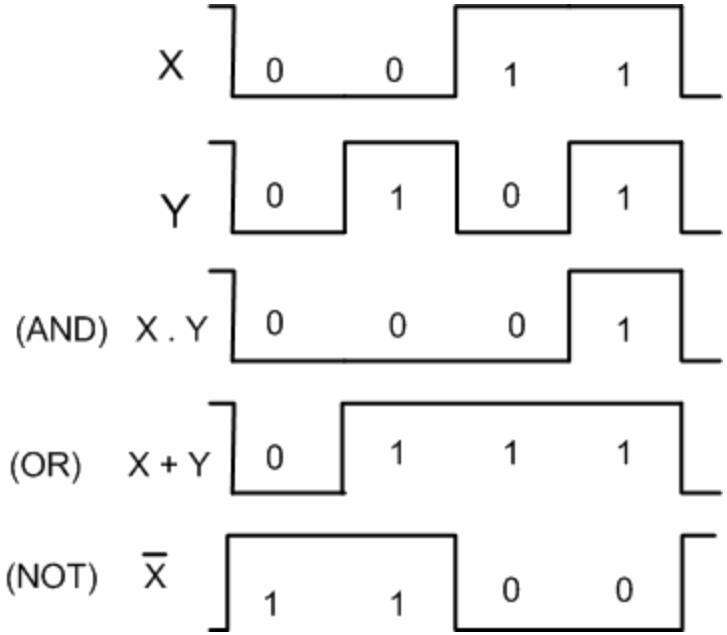


Logic Gate Symbols and Behavior

- **Logic gates have special symbols:**

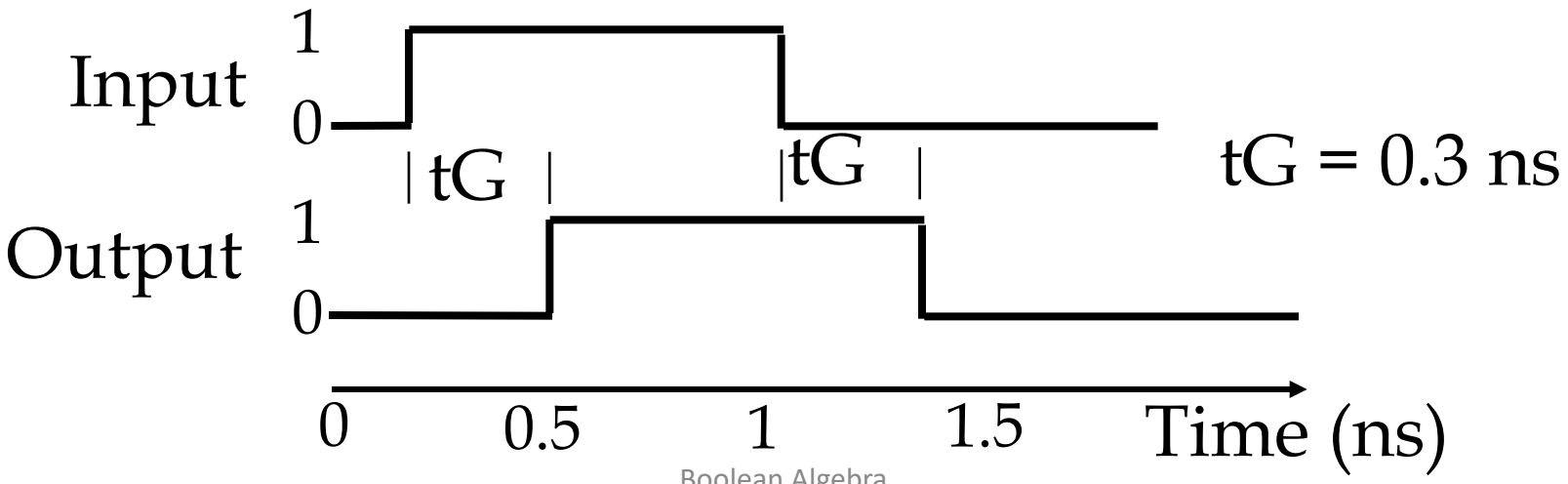


- **And waveform behavior in time as follows:**



Gate Delay

- In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously.
- The delay between an input change(s) and the resulting output change is the *gate delay* denoted by t_G:



Other Gate Types

- Why?
  - Implementation feasibility and low cost
  - Power in implementing Boolean functions
  - Convenient conceptual representation
- Gate classifications
  - Primitive gate - a gate that can be described using a single primitive operation type (AND or OR) plus an optional inversion(s).
  - Complex gate - a gate that requires more than one primitive operation type for its description
- Primitive gates will be covered first

Logic Gates

Name	Distinctive-Shape Graphics Symbol	Algebraic Equation	Truth Table
AND		$F = XY$	X Y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = X + Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT (inverter)		$F = \overline{X}$	X | F 0 | 1 1 | 0
NAND		$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = X\overline{Y} + \overline{X}Y$ $= X \oplus Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR (XNOR)		$F = XY + \overline{X}\overline{Y}$ $= X \oplus Y$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Logic Diagrams and Expressions

TABLE 2-5
Truth Table for the Function $L = D\overline{X} + A$

D	X	A	L
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Equation

$$L = A + \overline{X}D$$

Logic Diagram

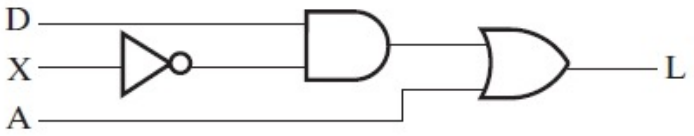


FIGURE 2-5
Logic Circuit Diagram for $L = D\overline{X} + A$

- Boolean equations, truth tables and logic diagrams describe the same function!
- Truth tables are unique; expressions and logic diagrams are not. This gives flexibility in implementing functions.

2-2 Boolean Algebra

Laws of Boolean Algebra

- Identity laws

  $A + 0 = 0 + A = A$; $A \cdot 1 = 1 \cdot A = A$

- Inverse/complement laws

  $A + A' = 1$; $A \cdot A' = 0$

- Commutative laws

  $A + B = B + A$; $A \cdot B = B \cdot A$

- Associative laws

  $A + (B + C) = (A + B) + C$; $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

- Distributive laws

  $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$; $A + (B \cdot C) = (A + B) \cdot (A + C)$

Precedence of Operators

- Precedence from highest to lowest
  - Not
  - And
  - Or
- Examples:
  - $A \cdot B + C = (A \cdot B) + C$
  - $X + Y' = X + (Y')$
  - $P + Q' \cdot R = P + ((Q') \cdot R)$
- Use parenthesis to overwrite precedence.
  Examples:
  - $A \cdot (B + C)$
  - $(P + Q)' \cdot R$

Truth Table

- Provide a listing of every possible combination of inputs and its corresponding outputs.
  - Inputs are usually listed in binary sequence.
- Example
  - Truth table with 3 inputs and 2 outputs

x	y	z	y + z	x • (y + z)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Proof Using Truth Table

- Prove: **x • (y + z) = (x • y) + (x • z)**
  - Construct truth table for LHS and RHS

x	y	z	y + z	**x • (y + z)**	x • y	x • z	**(x • y) + (x • z)**
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

  - Check that column for LHS = column for RHS

Duality

- If the AND/OR operators and identity elements 0/1 in a Boolean equation are interchanged, it remains valid

  Example:

  The dual equation of $a+(b\cdot c)=(a+b)\cdot(a+c)$ is $a\cdot(b+c)=(a\cdot b)+(a\cdot c)$

- Duality gives free theorems – "two for the price of one". You prove one theorem and the other comes for free!

  Examples:

  If $(x+y+z)' = x'\cdot y'\cdot z'$ is valid, then its dual is also valid:
  $(x\cdot y\cdot z)' = x'+y'+z'$

  If $x+1 = 1$ is valid, then its dual is also valid:
  $x\cdot 0 = 0$

Basic Theorems (1/2)

1. Idempotency

   $X + X = X$; $X \cdot X = X$

2. Zero and One elements

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

3. Involution

   $(X')' = X$

4. Absorption

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

5. Absorption (variant)

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

Basic Theorems (2/2)

6. DeMorgan's

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

DeMorgan's Theorem can be generalized to more than two variables, example: $(A + B + \ldots + Z)' = A' \cdot B' \cdot \ldots \cdot Z'$

7. Consensus

$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$

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

Proving a Theorem

- Theorems can be proved using truth table, or by algebraic manipulation using other theorems/laws.

  Example: Prove absorption theorem $X + X \cdot Y = X$

  $X + X \cdot Y = X \cdot 1 + X \cdot Y$ (by identity)

  $= X \cdot (1+Y)$ (by distributivity)

  $= X \cdot (Y+1)$ (by commutativity)

  $= X \cdot 1$ (by one element)

  $= X$ (by identity)

  By duality, we have also proved $X \cdot (X+Y) = X$

- Our primary reason for doing proofs is to learn:
  - Careful and efficient use of the identities and theorems of Boolean algebra, and
  - How to choose the appropriate identity or theorem to apply to make forward progress, irrespective of the application.

Example 2: Boolean Algebraic Proofs

- $AB + \overline{A}C + BC = AB + \overline{A}C$ (Consensus Theorem)

Proof Steps	Justification (identity or theorem)
$AB + \overline{A}C + BC$	
$= AB + \overline{A}C + 1 \cdot BC$	?
$= AB + \overline{A}C + (A + \overline{A}) \cdot BC$	?
$=$	

Example 3: Boolean Algebraic Proofs

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

Proof Steps	Justification (identity or theorem)
$\overline{(X + Y)}Z + X\overline{Y}$	
=	

Proof of Simplification

$$x \cdot y + \overline{x} \cdot y = y$$

Boolean Functions

- Examples of Boolean functions (logic equations):

  F1(x,y,z) = x·y·z'

  F2(x,y,z) = x + y'·z

  F3(x,y,z) = x'·y'·z + x'·y·z + x·y'

  F4(x,y,z) = x·y' + x'·z

x	y	z	F1	F2	F3	F4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Complement

- Given a Boolean function F, the complement of F, denoted as F', is obtained by interchanging 1 with 0 in the function's output values.
- Example: F1 = x·y·z'
- What is F1' ?

x	y	z	F1	F1'
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	0	

Complementing Functions

- Use DeMorgan's Theorem to complement a function:
  1. Interchange AND and OR operators
  2. Complement each constant value and literal
- Example: Complement F = $\bar{x}y\bar{z} + x\bar{y}\bar{z}$

  $\bar{F} = (x + \bar{y} + z)(\bar{x} + y + z)$
- Example: Complement G = $(\bar{a} + bc)\bar{d} + e$

  $\bar{G} =$

Expression Simplification

- An application of Boolean algebra
- Simplify to contain the smallest number of literals (complemented and uncomplemented variables):

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

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

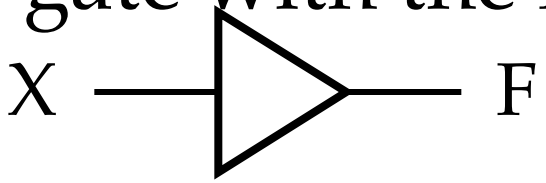
$$= AB + AB(CD) + \overline{A} C (D + \overline{D}) + \overline{A} B D$$

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

$$= B (A + D) + \overline{A} C \quad \textbf{5 literals}$$

Buffer

- A buffer is a gate with the function F = X:



- In terms of Boolean function, a buffer is the same as a connection!
- So why use it?
  - A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation.

2-3 Standard Forms

Minterms and Maxterms (1/2)

- A minterm of n variables is a product term that contains n literals from all the variables.

  Example: On 2 variables x and y, the minterms are:

  x'·y', x'·y, x·y' and x·y

- A maxterm of n variables is a sum term that contains n literals from all the variables.

  Example: On 2 variables x and y, the maxterms are:

  x'+y', x'+y, x+y' and x+y

- In general, with n variables we have 2^n minterms and 2^n maxterms.

Minterms and Maxterms (2/2)

- The minterms and maxterms on 2 variables are denoted by m0 to m3 and M0 to M3 respectively.

x	y	Minterms		Maxterms	
		Term	Notation	Term	Notation
0	0	x'·y'	m0	x+y	M0
0	1	x'·y	m1	x+y'	M1
1	0	x·y'	m2	x'+y	M2
1	1	x·y	m3	x'+y'	M3

- Each minterm is the complement of the corresponding maxterm
  - Example: m2 = x·y'

$$m2' = (x \cdot y')' = x' + (y')' = x' + y = M2$$

Standard Order

- Minterms and maxterms are designated with a subscript
- The subscript is a number, corresponding to a binary pattern
- The bits in the pattern represent the complemented or normal state of each variable listed in a standard order.
- All variables will be present in a minterm or maxterm and will be listed in the same order (usually alphabetically)
- Example: For variables a, b, c:
  - Maxterms: $(a + b + \bar{c})$, $(a + b + c)$
  - Terms: $(b + a + c)$, $a\ \bar{c}\ b$, and $(c + b + a)$ are NOT in standard order.
  - Minterms: $a\ \bar{b}\ c$, $a\ b\ c$, $\bar{a}\ \bar{b}\ c$
  - Terms: $(a + c)$, $\bar{b}\ c$, and $(\bar{a} + b)$ do not contain all variables

Purpose of the Index

- The index for the minterm or maxterm, expressed as a binary number, is used to determine whether the variable is shown in the true form or complemented form.
- For Minterms:
  - "1" means the variable is "Not Complemented" and
  - "0" means the variable is "Complemented".
- For Maxterms:
  - "0" means the variable is "Not Complemented" and
  - "1" means the variable is "Complemented".

Index Example in Three Variables

- Example: (for three variables)
- Assume the variables are called X, Y, and Z.
- The standard order is X, then Y, then Z.
- The Index 0 (base 10) = 000 (base 2) for three variables). All three variables are complemented for minterm 0 ($\overline{X}, \overline{Y}, \overline{Z}$) and no variables are complemented for Maxterm 0 (X,Y,Z).
  - Minterm 0, called m_0 is $\overline{X}\,\overline{Y}\,\overline{Z}$.
  - Maxterm 0, called M_0 is (X + Y + Z).
  - Minterm 6 ?
  - Maxterm 6 ?

Index Examples – Four Variables

Index i	Binary Pattern	Minterm m_i	Maxterm M_i
0	0000	$\bar{a}\bar{b}\bar{c}\bar{d}$	$a+b+c+d$
1	0001	$\bar{a}\bar{b}\bar{c}d$	?
3	0011	?	$a+b+\bar{c}+\bar{d}$
5	0101	$\bar{a}b\bar{c}d$	$a+\bar{b}+c+\bar{d}$
7	0111	?	$a+\bar{b}+\bar{c}+\bar{d}$
10	1010	$a\bar{b}c\bar{d}$	$\bar{a}+b+\bar{c}+d$
13	1101	$ab\bar{c}d$	?
15	1111	$abcd$	$\bar{a}+\bar{b}+\bar{c}+\bar{d}$

Minterm and Maxterm Relationship

- **Review: DeMorgan's Theorem**
  $\overline{x \cdot y} = \bar{x} + \bar{y}$ and $\overline{x + y} = \bar{x} \bullet \bar{y}$
- **Two-variable example:**
  $M_2 = \bar{x} + y$ and $m_2 = x\,\bar{y}$
  Thus M_2 is the complement of m_2 and vice-versa.
- **Since DeMorgan's Theorem holds for *n* variables, the above holds for terms of *n* variables**
- **giving:**

$$M_i = \overline{m}_i \text{ and } m_i = \overline{M}_i$$

Thus M_i is the complement of m_i.

Function Tables for Both

- Minterms of 2 variables

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

Maxterms of 2 variables

x y	M_0	M_1	M_2	M_3
0 0	0	1	1	1
0 1	1	0	1	1
1 0	1	1	0	1
1 1	1	1	1	0

- Each column in the maxterm function table is the complement of the column in the minterm function table since M_i is the complement of m_i.

Observations

- In the function tables:
  - Each minterm has one and only one 1 present in the 2^n terms (a minimum of 1s). All other entries are 0.
  - Each maxterm has one and only one 0 present in the 2^n terms All other entries are 1 (a maximum of 1s).
- We can implement any function by "**OR**ing" the minterms corresponding to "1" entries in the function table. These are called the minterms of the function.
- We can implement any function by "**AND**ing" the maxterms corresponding to "0" entries in the function table. These are called the maxterms of the function.
- This gives us two canonical forms:
  - **Sum of Minterms (SOM)**
  - **Product of Maxterms (POM)**

for stating any Boolean function.

Minterm Function Example

- Example: Find $F_1 = m_1 + m_4 + m_7$
- F1 = $\overline{x}\ \overline{y}\ z + x\ \overline{y}\ \overline{z} + x\ y\ z$

x y z	index	m1	+	m4	+	m7	= F1
0 0 0	0	0	+	0	+	0	= 0
0 0 1	1	1	+	0	+	0	= 1
0 1 0	2	0	+	0	+	0	= 0
0 1 1	3	0	+	0	+	0	= 0
1 0 0	4	0	+	1	+	0	= 1
1 0 1	5	0	+	0	+	0	= 0
1 1 0	6	0	+	0	+	0	= 0
1 1 1	7	0	+	0	+	1	= 1

Minterm Function Example

- $F(A, B, C, D, E) = m_2 + m_9 + m_{17} + m_{23}$
- $F(A, B, C, D, E) =$

Maxterm Function Example

- Example: Implement F1 in maxterms:

$F_1 = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$

$F_1 = (x + y + z) \cdot (x + \bar{y} + z) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + y + \bar{z}) \cdot (\bar{x} + \bar{y} + z)$

x y z	i	$M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 = F1$
0 0 0	0	0 · 1 · 1 · 1 · 1 = 0
0 0 1	1	1 · 1 · 1 · 1 · 1 = 1
0 1 0	2	1 · 0 · 1 · 1 · 1 = 0
0 1 1	3	1 · 1 · 0 · 1 · 1 = 0
1 0 0	4	1 · 1 · 1 · 1 · 1 = 1
1 0 1	5	1 · 1 · 1 · 0 · 1 = 0
1 1 0	6	1 · 1 · 1 · 1 · 0 = 0
1 1 1	7	1 · 1 · 1 · 1 · 1 = 1

Maxterm Function Example

- $F(A,B,C,D) = M_3 \times M_8 \times M_{11} \times M_{14}$
- F(A, B,C,D) =

Canonical Sum of Minterms

- Any Boolean function can be expressed as a Sum of Minterms.
  - For the function table, the minterms used are the terms corresponding to the 1's
  - For expressions, expand all terms first to explicitly list all minterms. Do this by "ANDing" any term missing a variable v with a term $(v + \bar{v})$.
- Example: Implement $f = x + \bar{x}\,\bar{y}$ as a sum of minterms.

  First expand terms: $f = x(y + \bar{y}) + \bar{x}\,\bar{y}$

  Then distribute terms: $f = xy + x\bar{y} + \bar{x}\,\bar{y}$

  Express as sum of minterms: $f = m_3 + m_2 + m_0$

Another SOM Example

- Example: $F = A + \overline{B}\,C$
- There are three variables, A, B, and C which we take to be the standard order.
- Expanding the terms with missing variables:

- Collect terms (removing all but one of duplicate terms):
- Express as SOM:

Shorthand SOM Form

- From the previous example, we started with:

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

- We ended up with:

  $$F = m_1 + m_4 + m_5 + m_6 + m_7$$

- This can be denoted in the formal shorthand:

  $$\mathbf{F(A, B, C) = \Sigma_m(1,4,5,6,7)}$$

- Note that we explicitly show the standard variables in order and drop the "m" designators.

Canonical Product of Maxterms

- Any Boolean Function can be expressed as a Product of Maxterms (POM).
  - For the function table, the maxterms used are the terms corresponding to the 0's.
  - For an expression, expand all terms first to explicitly list all maxterms. Do this by first applying the second distributive law , "ORing" terms missing variable v with a term equal to $v \cdot \bar{v}$ and then applying the distributive law again.
- Example: Convert to product of maxterms:

$$f(x, y, z) = x + \bar{x}\,\bar{y}$$

Apply the distributive law:

$$x + \bar{x}\,\bar{y} = (x + \bar{x})(x + \bar{y}) = 1 \cdot (x + \bar{y}) = x + \bar{y}$$

Add missing variable z:

$$x + \bar{y} + z \cdot \bar{z} = (x + \bar{y} + z)\,(x + \bar{y} + \bar{z})$$

Express as POM: $f = M_2 \cdot M_3$

Another POM Example

- Convert to Product of Maxterms:

$$f(A,B,C) = A\overline{C} + BC + \overline{A}\,\overline{B}$$

- Use $x + yz = (x+y)\cdot(x+z)$ with $x = (A\overline{C} + BC)$, $y = \overline{A}$, and $z = \overline{B}$ to get:

$$f = (A\overline{C} + BC + \overline{A})(A\overline{C} + BC + \overline{B})$$

- Then use $x + \overline{x}\,y = x + y$ to get:

$$f = (\overline{C} + BC + \overline{A})(A\overline{C} + C + \overline{B})$$

and a second time to get:

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

- Rearrange to standard order,

$$f = (\overline{A} + B + \overline{C})(A + \overline{B} + C)$$ to give $f = M_5 \cdot M_2$

Another POM Example

Convert to Product of Maxterms: $F_1(A,B,C) = A\,C' + B\,C + A'B'$

A	B	C	F_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$$F1 = M2 \bullet M5$$
$$= (A + B' + C) \bullet (A' + B + C')$$

Function Complements

- The complement of a function expressed as a sum of minterms is constructed by selecting the minterms missing in the sum-of-minterms canonical forms.
- Alternatively, the complement of a function expressed by a Sum of Minterms form is simply the Product of Maxterms with the same indices.
- Example: Given

$$F(x, y, z) = \sum_m(1,3,5,7)$$

$$\overline{F}(x, y, z) = \sum_m(0,2,4,6)$$

$$\overline{F}(x, y, z) = \prod_M(1,3,5,7)$$

Conversion Between Forms

- To convert between sum-of-minterms and product-of-maxterms form (or vice-versa) we follow these steps:
  - Find the function complement by swapping terms in the list with terms not in the list.
  - Change from products to sums, or vice versa.
- Example:Given F as before: $\mathbf{F(x, y, z) = \Sigma_m(1,3,5,7)}$
- Form the Complement: $\mathbf{\bar{F}(x, y, z) = \sum_m(0,2,4,6)}$
- Then use the other form with the same indices – this forms the complement again, giving the other form of the original function:

$$\mathbf{F(x, y, z) = \Pi_M(0,2,4,6)}$$

Standard Forms (1/2)

- Certain types of Boolean expressions lead to circuits that are desirable from implementation viewpoint.
- Two standard forms:
  - Sum-of-Products
  - Product-of-Sums
- Literals
  - A Boolean variable on its own or in its complemented form
  - Examples: x, x', y, y'
- Product term
  - A single literal or a logical product (AND) of several literals
  - Examples: x, x·y·z', A'·B, A·B, d·g'·v·w

Standard Forms (2/2)

- Sum term
  - A single literal or a logical sum (OR) of several literals
  - Examples: x, x+y+z', A'+B, A+B, c+d+h'+j
- Sum-of-Products (SOP) expression
  - A product term or a logical sum (OR) of several product terms
  - Examples: x, x + y·z', x·y' + x'·y·z, A·B + A'·B', A + B'·C + A·C' + C·D
- Product-of-Sums (POS) expression
  - A sum term or a logical product (AND) of several sum terms
  - Examples: x, x·(y+z'), (x+y')·(x'+y+z), (A+B)·(A'+B'), (A+B+C)·D'·(B'+D+E')
- Every Boolean expression can be expressed in SOP or POS.

Do it yourself

- Put the right ticks in the following table.

Expression	*SOP?*	*POS?*
X'·Y + X·Y' + X·Y·Z		
(X+Y')·(X'+Y)·(X'+Z')		
X' + Y + Z		
X·(W' + Y·Z)		
X·Y·Z'		
W·X'·Y + V·(X·Z + W')		

Standard Sum-of-Products (SOP)

- A sum of minterms form for n variables can be written down directly from a truth table.
  - Implementation of this form is a two-level network of gates such that:
    - The first level consists of n-input AND gates, and
    - The second level is a single OR gate (with fewer than 2^n inputs).
- This form often can be simplified so that the corresponding circuit is simpler.

Standard Sum-of-Products (SOP)

- A Simplification Example:
- **$F(A, B, C) = \Sigma m(1,4,5,6,7)$**
- Writing the minterm expression:

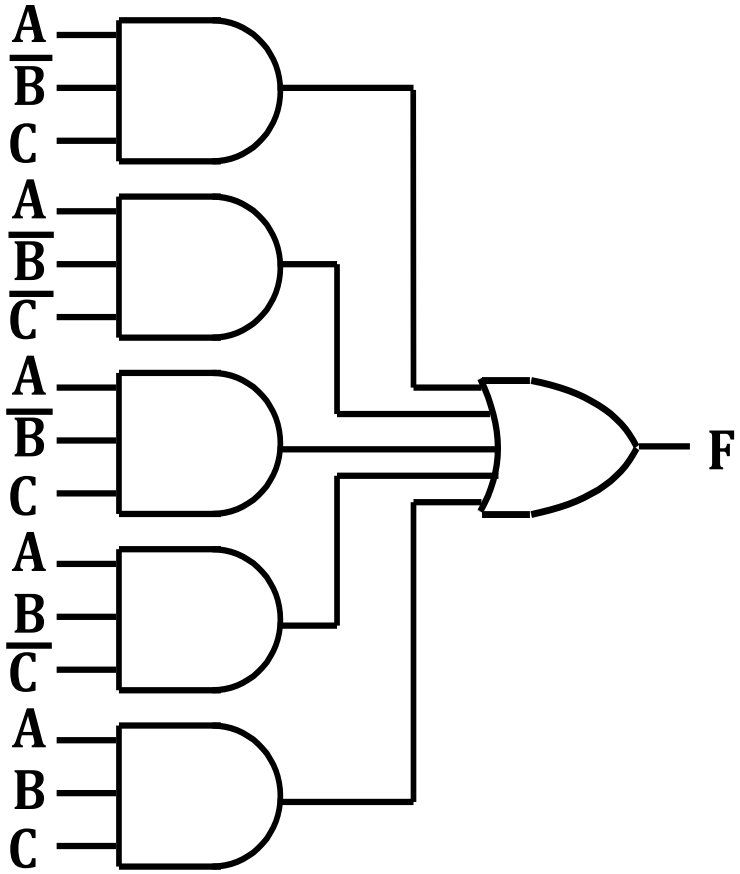
  $F = \overline{A}\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C + AB\overline{C} + ABC$
- Simplifying:

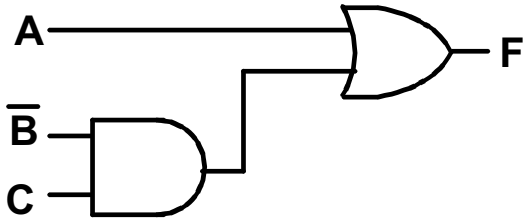
  F =

- Simplified F contains 3 literals compared to 15 in minterm F

AND/OR Two-level Implementation of SOP Expression

- The two implementations for F are shown below – it is quite apparent which is simpler!





SOP and POS Observations

- The previous examples show that:
  - Canonical Forms (Sum-of-minterms, Product-of-Maxterms), or other standard forms (SOP, POS) differ in complexity
  - Boolean algebra can be used to manipulate equations into simpler forms.
  - Simpler equations lead to simpler two-level implementations
- Questions:
  - How can we attain a "simplest" expression?
  - Is there only one minimum cost circuit?
  - The next part will deal with these issues.

2-4 Two-Level Circuit Optimization

Function Simplification

- Why simplify?
  - Simpler expression 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
    - Suitable for automation
    - Can handle many variables (but computationally intensive)

Algebraic Simplification

- Example 1: Simplify $(x+y)\cdot(x+y')\cdot(x'+z)$

$\mathbf{(x+y)\cdot(x+y')}\cdot(x'+z)$	
$= (\mathbf{x\cdot x} + \mathbf{x\cdot y'} + \mathbf{x\cdot y} + \mathbf{y\cdot y'}) \cdot (x'+z)$	(associativity)
$= (x + x\cdot(\mathbf{y'+y}) + \mathbf{0}) \cdot (x'+z)$	(idemp, assoc., complement)
$= (x + \mathbf{x\cdot 1}) \cdot (x'+z)$	(complement, identity)
$= (\mathbf{x + x}) \cdot (x'+z)$	(identity)
$= \mathbf{x \cdot (x'+z)}$	(idempotency)
$= \mathbf{x\cdot x'} + x\cdot z$	(associativity)
$= \mathbf{0 + x\cdot z}$	(complement)
$= x\cdot z$	(identity)

- Number of literals reduced from 6 to 2.

Circuit Optimization

- Goal: To obtain the simplest implementation for a given function
- Optimization is a more formal approach to simplification that is performed using a specific procedure or algorithm
- Optimization requires a cost criterion to measure the simplicity of a circuit
- Distinct cost criteria we will use:
  - Literal cost (L)
  - Gate input cost (G)
  - Gate input cost with NOTs (GN)

Literal Cost

- Literal – a variable or it complement
- Literal cost – the number of literal appearances in a Boolean expression corresponding to the logic circuit diagram
- Examples:
  - F = BD + A B′ C + A C′ D′ L = 8
  - F = BD + A B′ C + A B′ D′ + AB C′ L =
  - F = (A + B)(A + D)(B + C + D′)(B′ + C′ + D) L =
  - Which solution is best?

Gate Input Cost

- Gate input costs - the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (G - inverters not counted, GN - inverters counted)
- For SOP and POS equations, it can be found from the equation(s) by finding the sum of:
  - all literal appearances
  - the number of terms excluding single literal terms,(G) and
  - optionally, the number of distinct complemented single literals (GN).
- Example:
  - F = A B C D + A′ B′ C′ D′ G = 10, GN = 14
  - F = (A′+B)(B′+C)(C′+D)(D′+A) G = , GN =
  - Which solution is best?

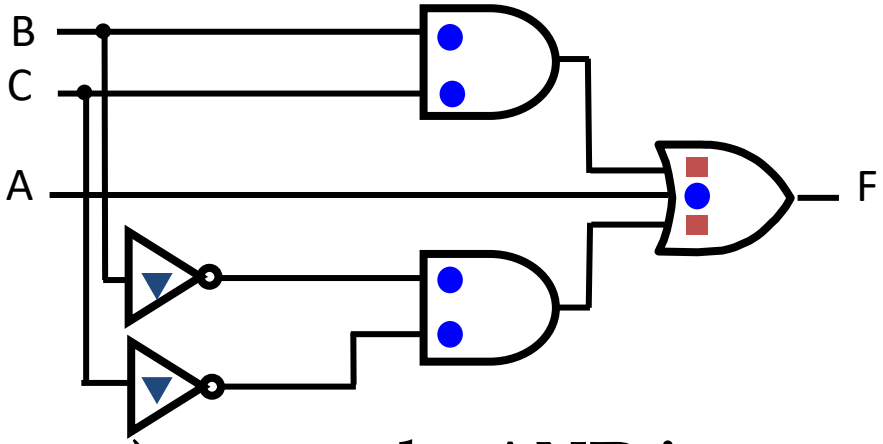
Cost Criteria (continued)

- Example 1:
- $F = A + BC + \overline{B}\,\overline{C}$

GN = G + 2 = 9

L = 5

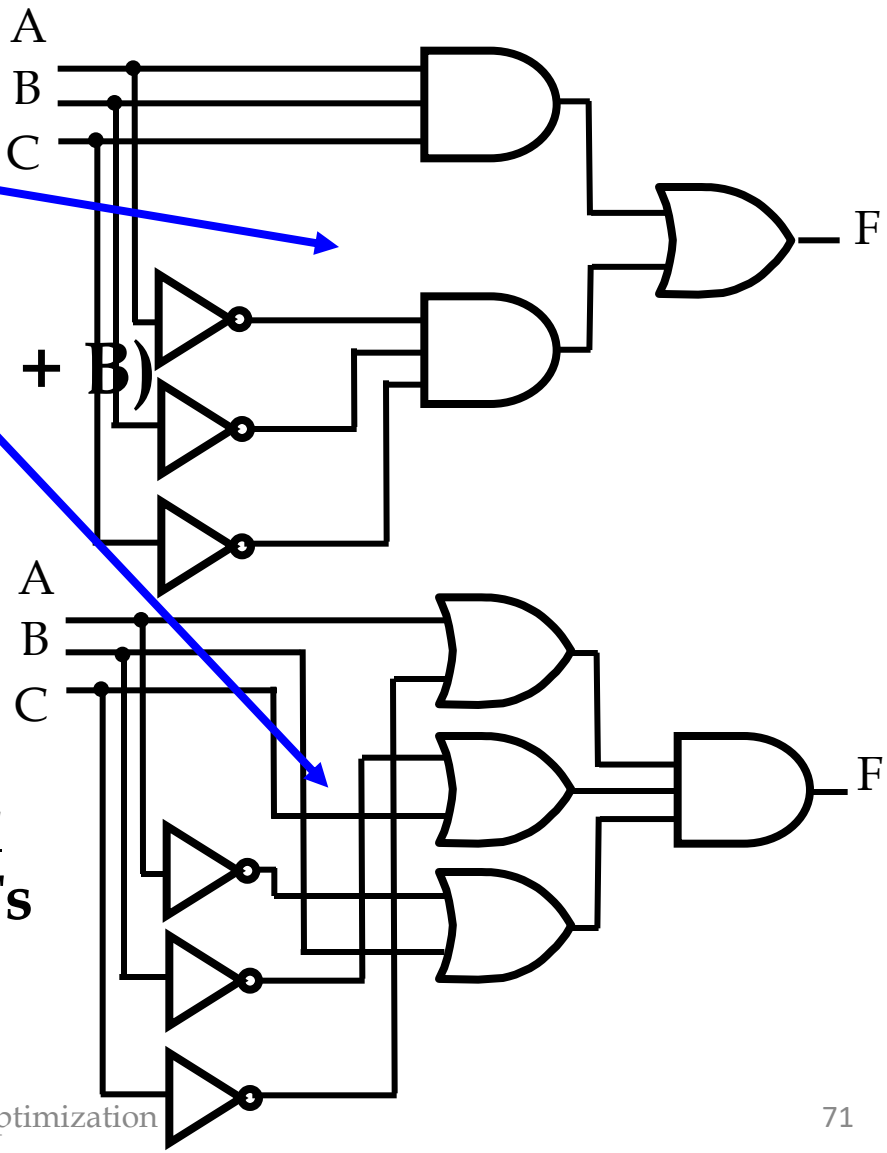
G = L + 2 = 7



- L (literal count) counts the AND inputs and the single literal OR input.
- G (gate input count) adds the remaining OR gate inputs
- GN(gate input count with NOTs) adds the inverter inputs

Cost Criteria (continued)

- Example 2:
- F = A B C + A′ B′ C′
- L = 6 G = 8 GN = 11
- F = (A + C′)(B′+ C)(A′ + B)
- L = 6 G = 9 GN = 12
- Same function and same literal cost
- But first circuit has better gate input count and better gate input count with NOTs
- Select it!



Boolean Function Optimization

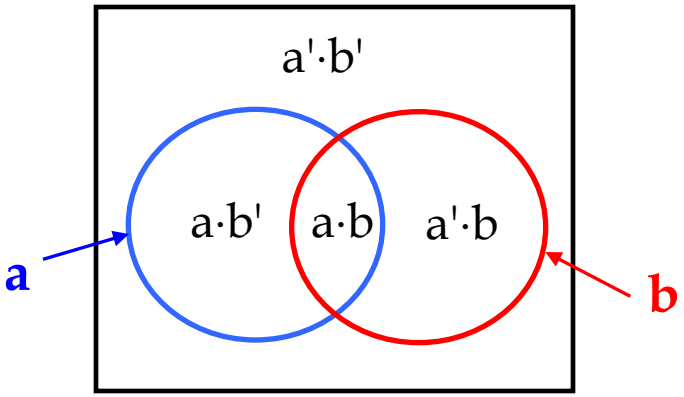
- Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) reduces circuit cost.
- We choose gate input cost.
- Boolean Algebra and graphical techniques are tools to minimize cost criteria values.
- Some important questions:
  - When do we stop trying to reduce the cost?
  - Do we know when we have a minimum cost?
- Treat optimum or near-optimum cost functions for two-level (SOP and POS) circuits first.
- Introduce a graphical technique using Karnaugh maps (K-maps, for short)

Introduction to K-Maps

- Systematic method to obtain simplified (minimal) 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.

Venn Diagram

- Example: 2 variables a and b represented by 2 circular regions. There are 4 minterms, each occupying their respective space.



- A set of minterms represents a certain Boolean function. Examples:

{ a·b, a·b' }	→ a·b + a·b' = a·(b+b') = a
{ a'·b, a·b }	→ a'·b + a·b = (a'+a)·b = b
{ a·b }	→ a·b
{ a·b, a·b', a'·b }	→ a·b + a·b' + a'·b = a + b
{ }	→ 0
{ a'·b', a·b, a·b', a'·b }	→ 1

Karnaugh Maps (K-map)

- A K-map is a collection of squares
  - Each square represents a minterm
  - The collection of squares is a graphical representation of a Boolean function
  - Adjacent squares differ in the value of one variable
  - Alternative algebraic expressions for the same function are derived by recognizing patterns of squares
- The K-map can be viewed as
  - A reorganized version of the truth table
  - A topologically-warped Venn diagram as used to visualize sets in algebra of sets

Some Uses of K-Maps

- Provide a means for:
  - Finding optimum or near optimum
    - SOP and POS standard forms, and
    - two-level AND/OR and OR/AND circuit implementations

    for functions with small numbers of variables
  - Visualizing concepts related to manipulating Boolean expressions, and
  - Demonstrating concepts used by computer-aided design programs to simplify large circuits

Two Variable Maps

- A 2-variable Karnaugh Map:
  - Note that minterm m0 and minterm m1 are "adjacent" and differ in the value of the variable y
  - Similarly, minterm m0 and minterm m2 differ in the x variable.
  - Also, m1 and m3 differ in the x variable as well.
  - Finally, m2 and m3 differ in the value of the variable y

	y = 0	y = 1
x = 0	$m_0 = \bar{x}\,\bar{y}$	$m_1 = \bar{x}\,y$
x = 1	$m_2 = x\,\bar{y}$	$m_3 = x\,y$

K-Map and Truth Tables

- The K-Map is just a different form of the truth table.
- Example – Two variable function:
  - We choose a,b,c and d from the set {0,1} to implement a particular function, F(x,y).

Function Table

Input Values (x,y)	Function Value F(x,y)
0 0	a
0 1	b
1 0	c
1 1	d

K-Map

	y = 0	y = 1
x = 0	a	b
x = 1	c	d

K-Map Function Representation

- Example: F(x,y) = x

F = x	y = 0	y = 1
x = 0	0	0
x = 1	1	1

- For function F(x,y), the two adjacent cells containing 1's can be combined using the Minimization Theorem:

$$F(x, y) = x\,\overline{y} + x\,y = x$$

K-Map Function Representation

- Example: G(x,y) = x + y

G = x+y	y = 0	y = 1
x = 0	0	1
x = 1	1	1

- For G(x,y), two pairs of adjacent cells containing 1's can be combined using the Minimization Theorem:

$$G(x,y) = \left(x\,\overline{y} + x\,y\right) + \left(xy + \overline{x}\,y\right) = x + y$$

Duplicate x y

Three Variable Maps

- A three-variable K-map:

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

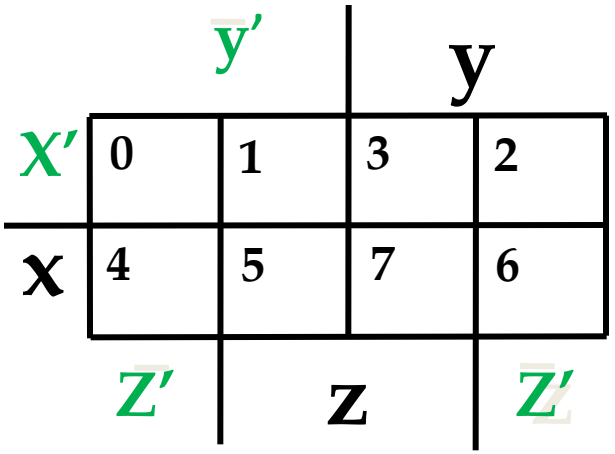
- Where each minterm corresponds to the product terms:

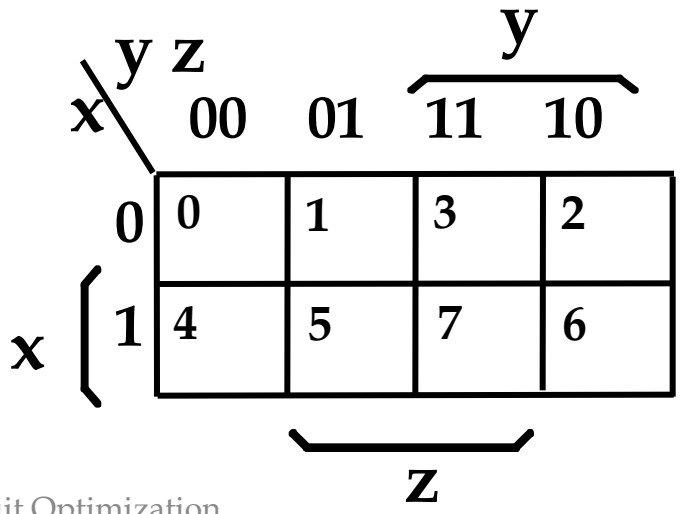
	yz=00	yz=01	yz=11	yz=10
x=0	$\bar{x}\,\bar{y}\,\bar{z}$	$\bar{x}\,\bar{y}\,z$	$\bar{x}\,y\,z$	$\bar{x}\,y\,\bar{z}$
x=1	$x\,\bar{y}\,\bar{z}$	$x\,\bar{y}\,z$	$x\,y\,z$	$x\,y\,\bar{z}$

- Note that if the binary value for an index differs in one bit position, the minterms are adjacent on the K-Map

Alternative Map Labeling

- Map use largely involves:
  - Entering values into the map, and
  - Reading off product terms from the map.
- Alternate labelings are useful:

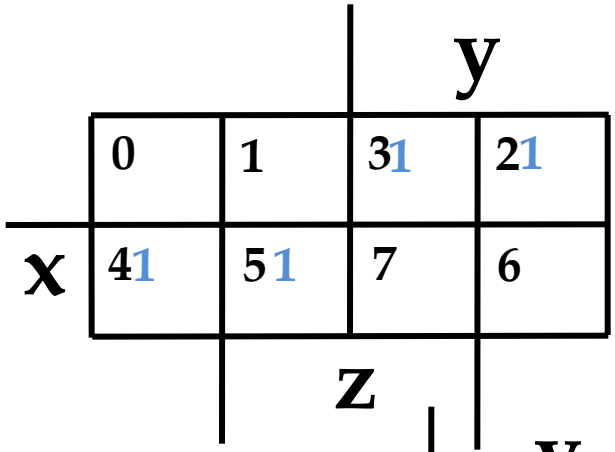




Example Functions

- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of $\overline{F}$ blank
- Example:

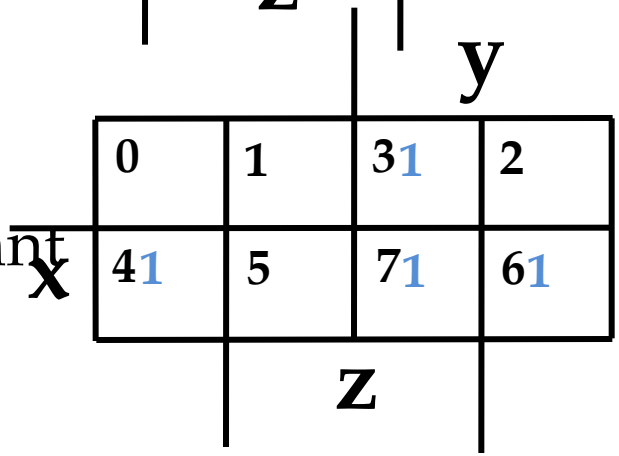
$$F(x, y, z) = \Sigma_m(2,3,4,5)$$



- Example:

$$G(a, b, c) = \Sigma_m(3,4,6,7)$$

- Learn the locations of the 8 indices based on the variable order shown (x, most significant and z, least significant) on the map boundaries



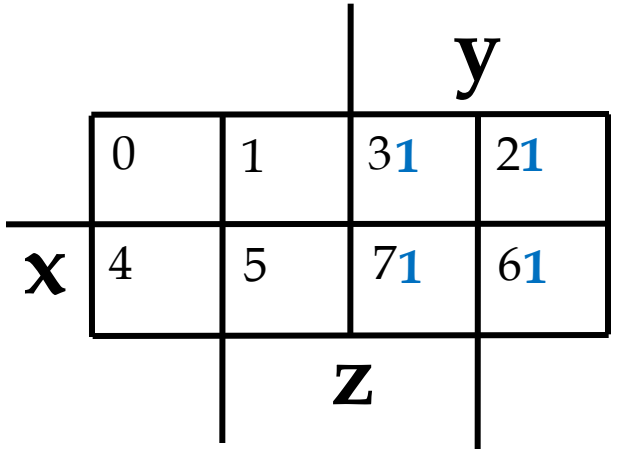
Combining Squares

- By combining squares, we reduce number of literals in a product term, reducing the literal cost, thereby reducing the other two cost criteria
- On a 3-variable K-Map:
  - One square represents a minterm with three variables
  - Two adjacent squares represent a product term with two variables
  - Four "adjacent" terms represent a product term with one variable
  - Eight "adjacent" terms is the function of all ones (no variables) = 1.

Example: Combining Squares

- Example: Let

$$F = \Sigma m(2,3,6,7)$$



- Applying the Minimization Theorem three times:

$$F(x,y,z) = \bar{x}yz + xyz + \bar{x}y\bar{z} + xy\bar{z}$$
$$= yz + y\bar{z}$$
$$= y$$

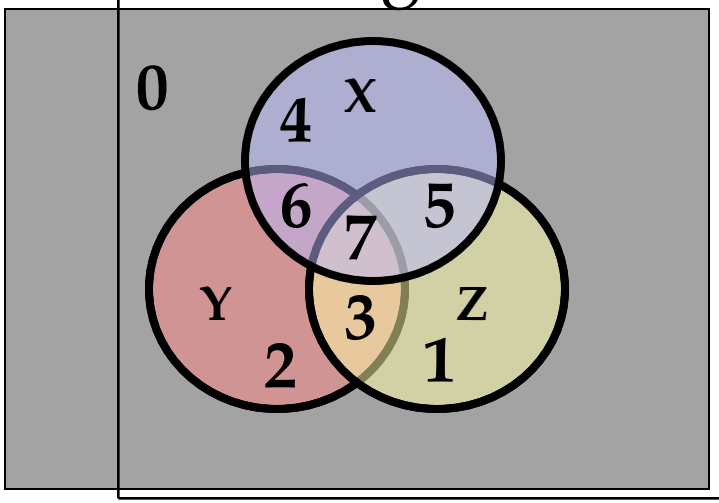
- Thus the four terms that form a 2 × 2 square correspond to the term "y".

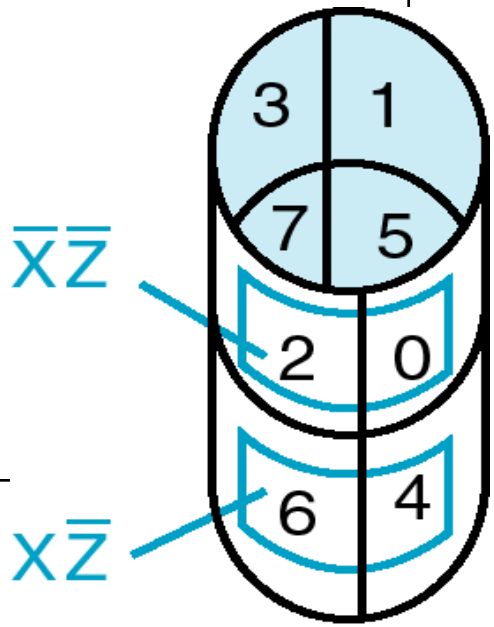
Three-Variable Maps

- Reduced literal product terms for SOP standard forms correspond to rectangles on K-maps containing cell counts that are powers of 2.
- Rectangles of 2 cells represent 2 adjacent minterms; of 4 cells represent 4 minterms that form a "pairwise adjacent" ring.
- Rectangles can contain non-adjacent cells as illustrated by the "pairwise adjacent" ring above.

Three-Variable Maps

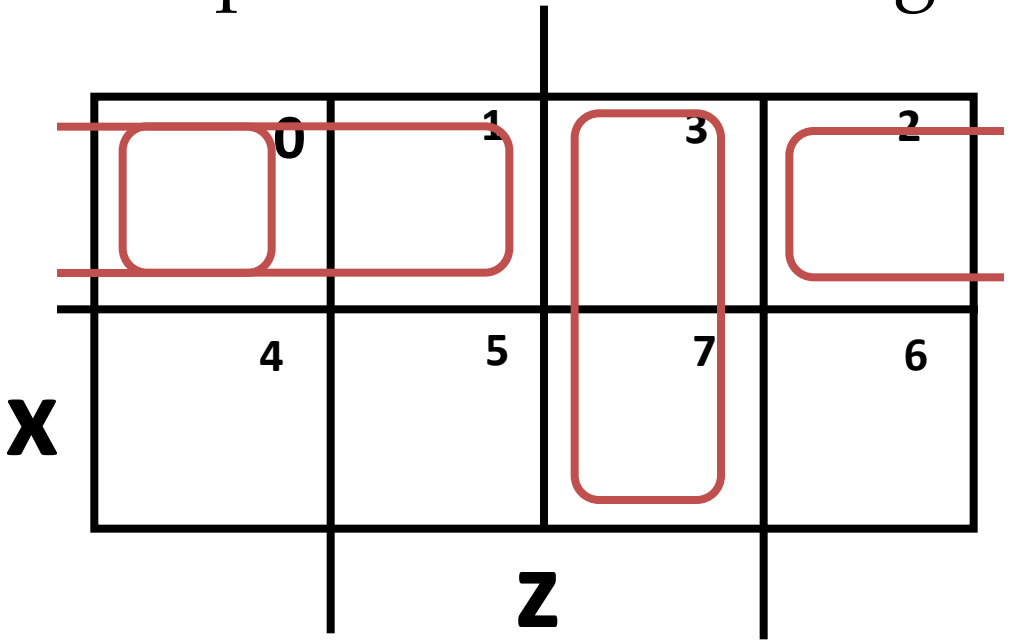
- Topological warps of 3-variable K-maps that show *all* adjacencies:
  - Venn Diagram
  - Cylinder





Three-Variable Maps

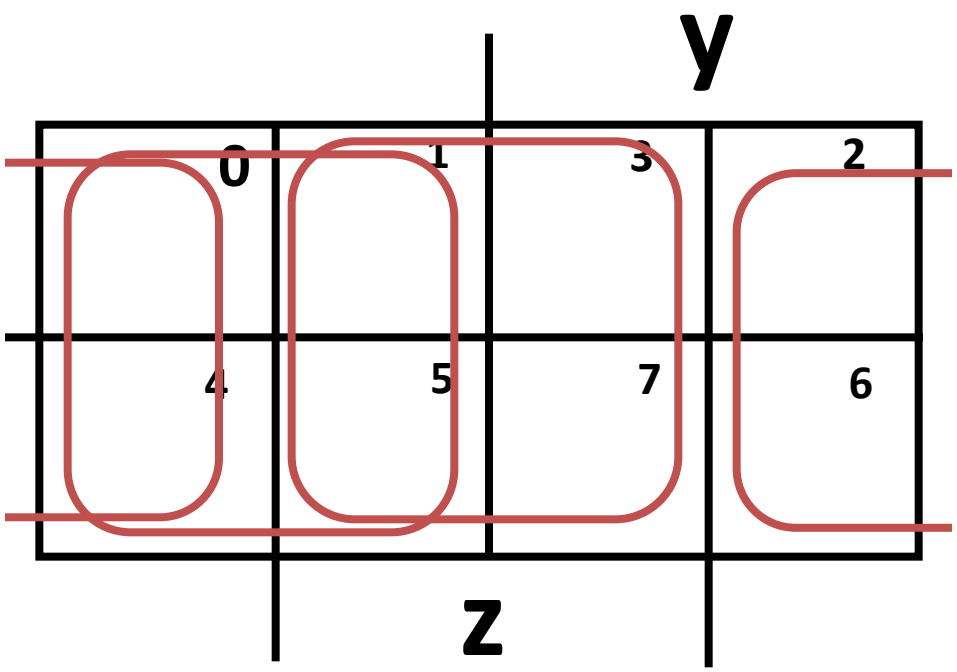
- Example Shapes of 2-cell Rectangles:



- Read off the product terms for the rectangles shown

Three-Variable Maps

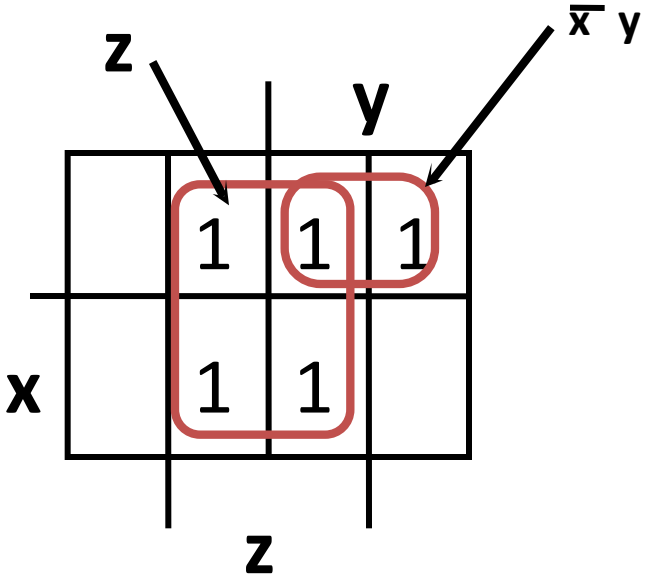
- Example Shapes of 4-cell Rectangles:



- Read off the product terms for the rectangles shown

Three Variable Maps

- K-Maps can be used to simplify Boolean functions by systematic methods. Terms are selected to cover the "1s" in the map.
- Example: Simplify $\mathbf{F(x, y, z) = \Sigma_m(1,2,3,5,7)}$



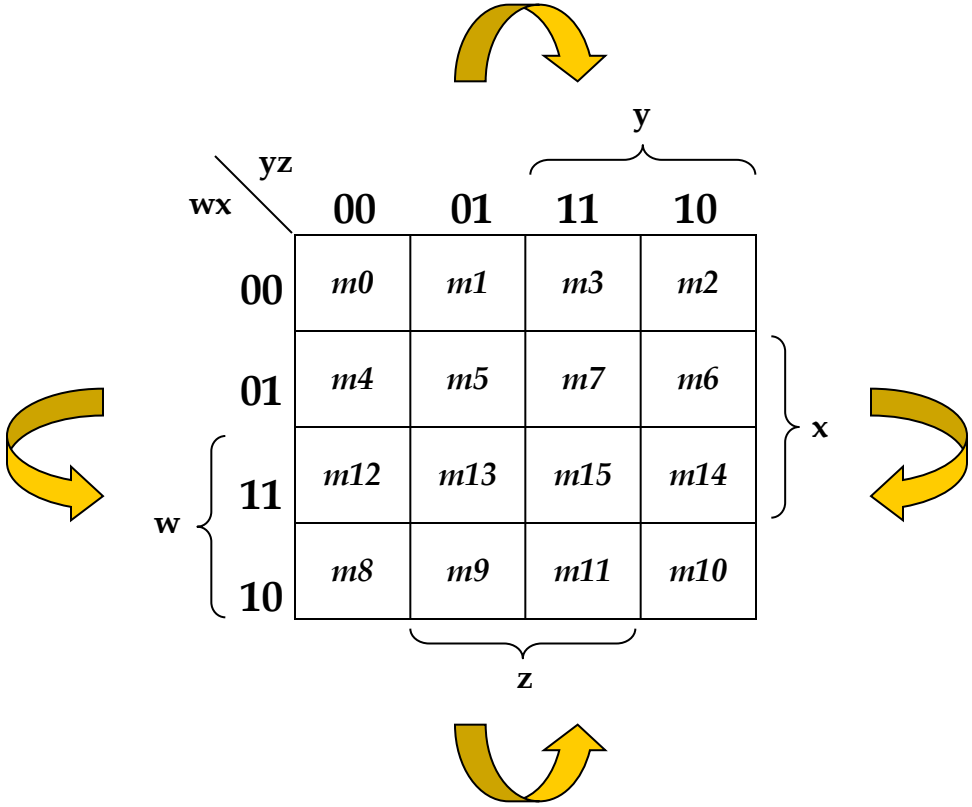
$F(x, y, z) = z + \bar{x}y$

Three-Variable Map Simplification

- Use a K-map to find an optimum SOP equation for $\mathbf{F(X, Y, Z) = \Sigma_m(0,1,2,4,6,7)}$

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.

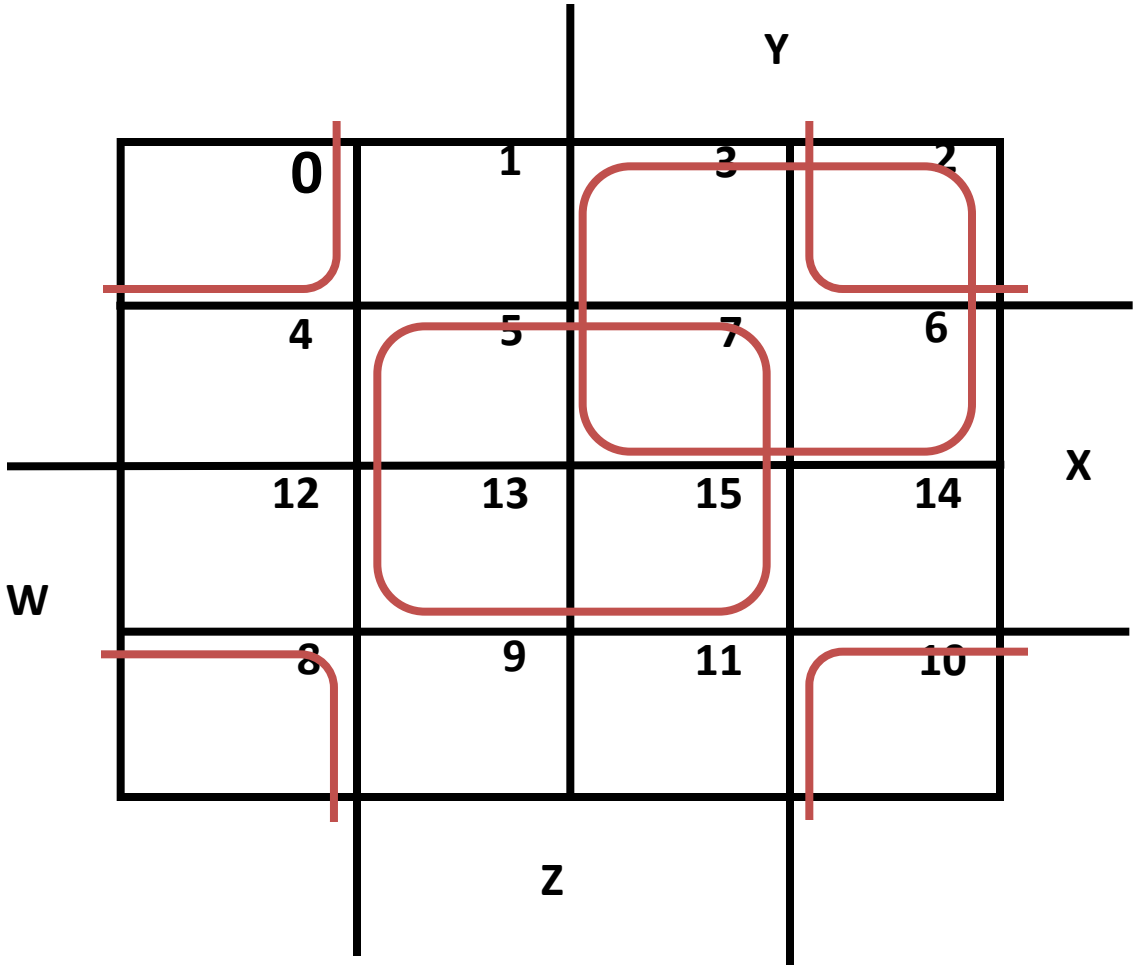


Four Variable Terms

- Four variable maps can have rectangles corresponding to:
  - A single 1 = 4 variables, (i.e. Minterm)
  - Two 1s = 3 variables,
  - Four 1s = 2 variables
  - Eight 1s = 1 variable,
  - Sixteen 1s = zero variables (i.e. Constant "1")

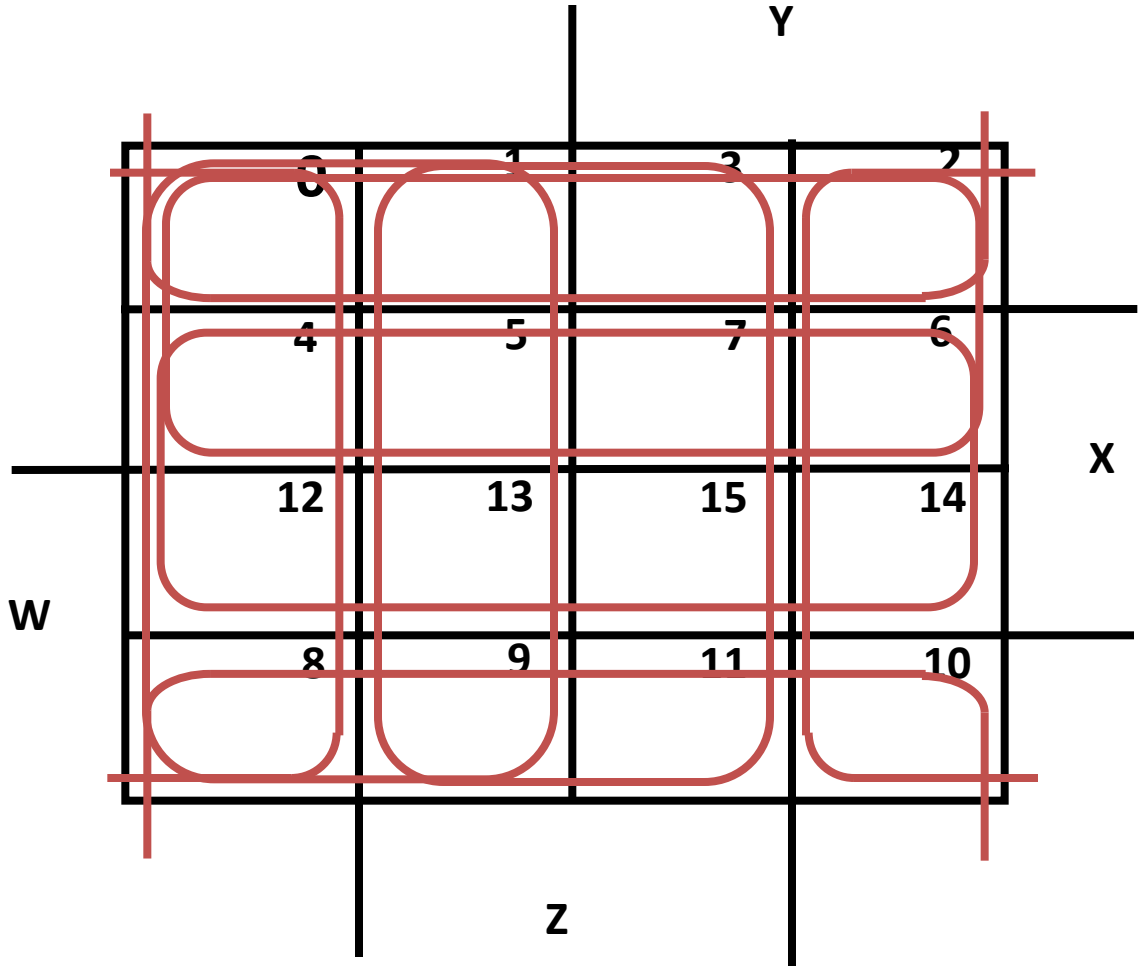
Four-Variable Maps

- Example Shapes of Rectangles:



Four-Variable Maps

- Example Shapes of Rectangles:



Four-Variable Map Simplification

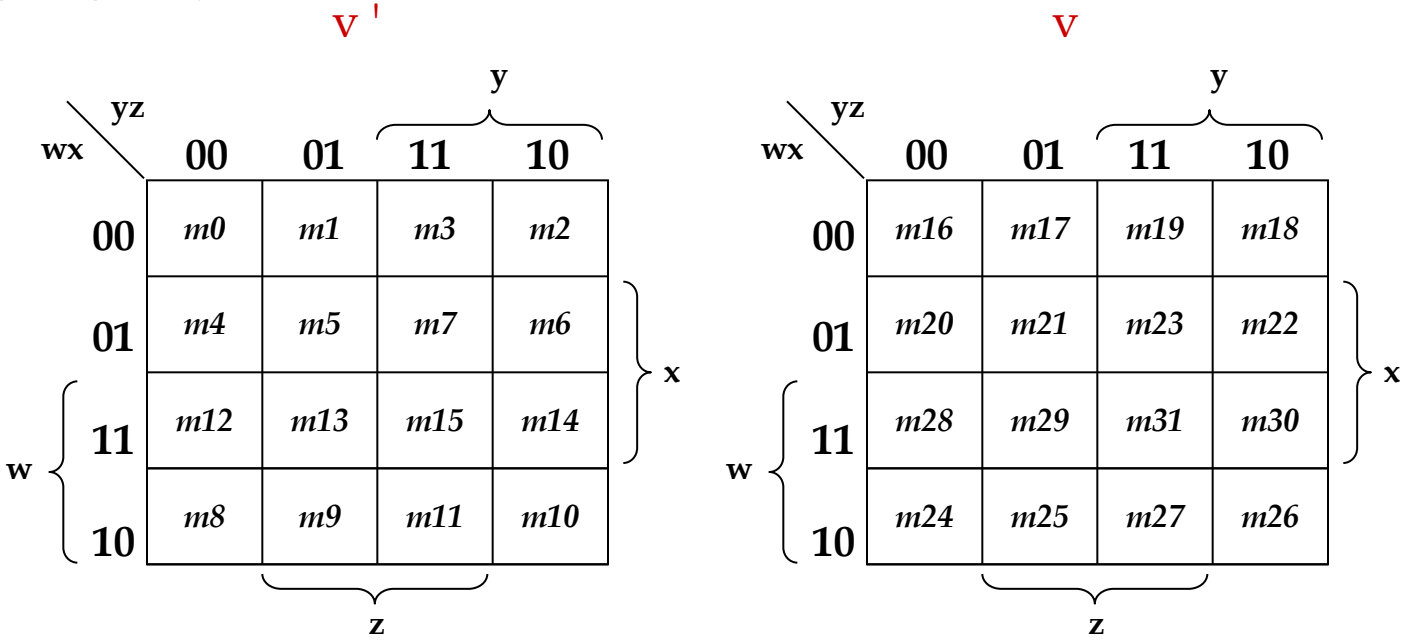
$$F(W, X, Y, Z) = \sum_m(0, 2,4,5,6,7,8,10,13,15)$$

Four-Variable Map Simplification

$$F(W, X, Y, Z) = \Sigma_m(3,4,5,7,9,13,14,15)$$

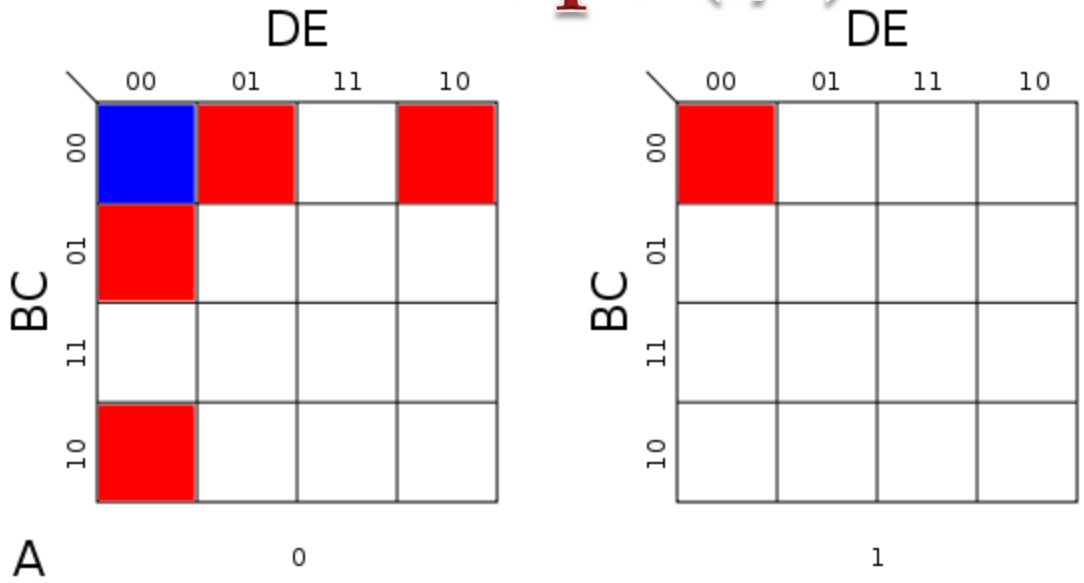
5-Variable K-Maps (1/2)

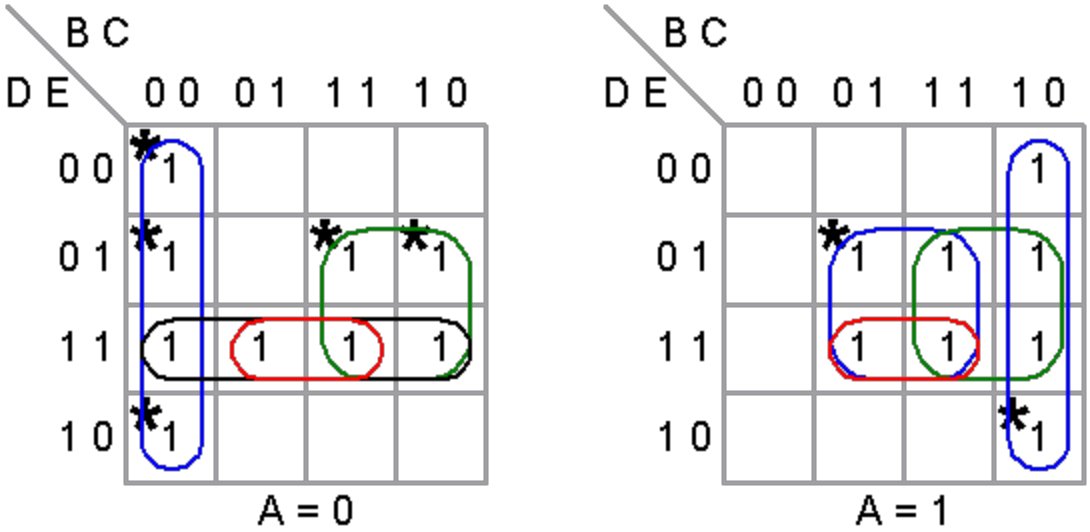
- Organised as two 4-variable K-maps. One for v' and the other for v.



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-Variable K-Maps (2/2)

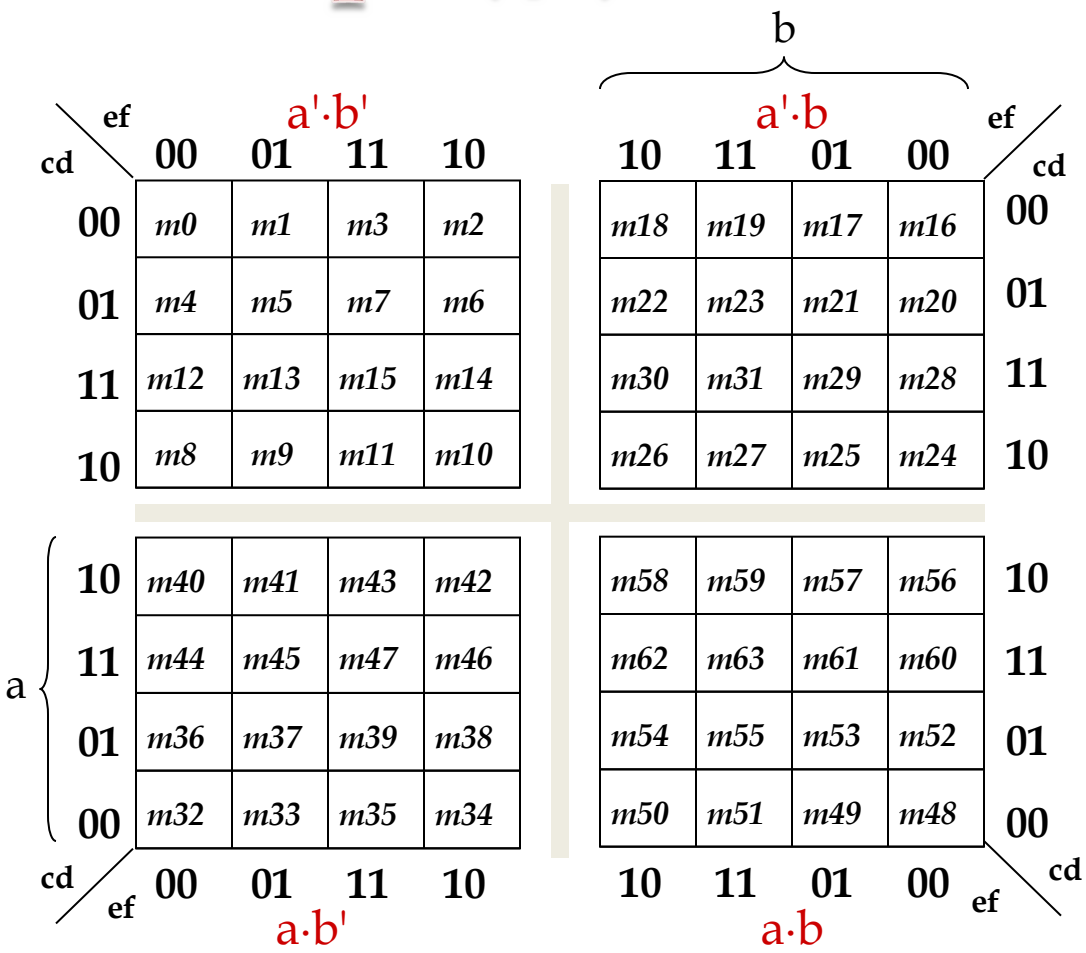




Larger K-Maps (1/2)

- 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.

Larger K-Maps (2/2)



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

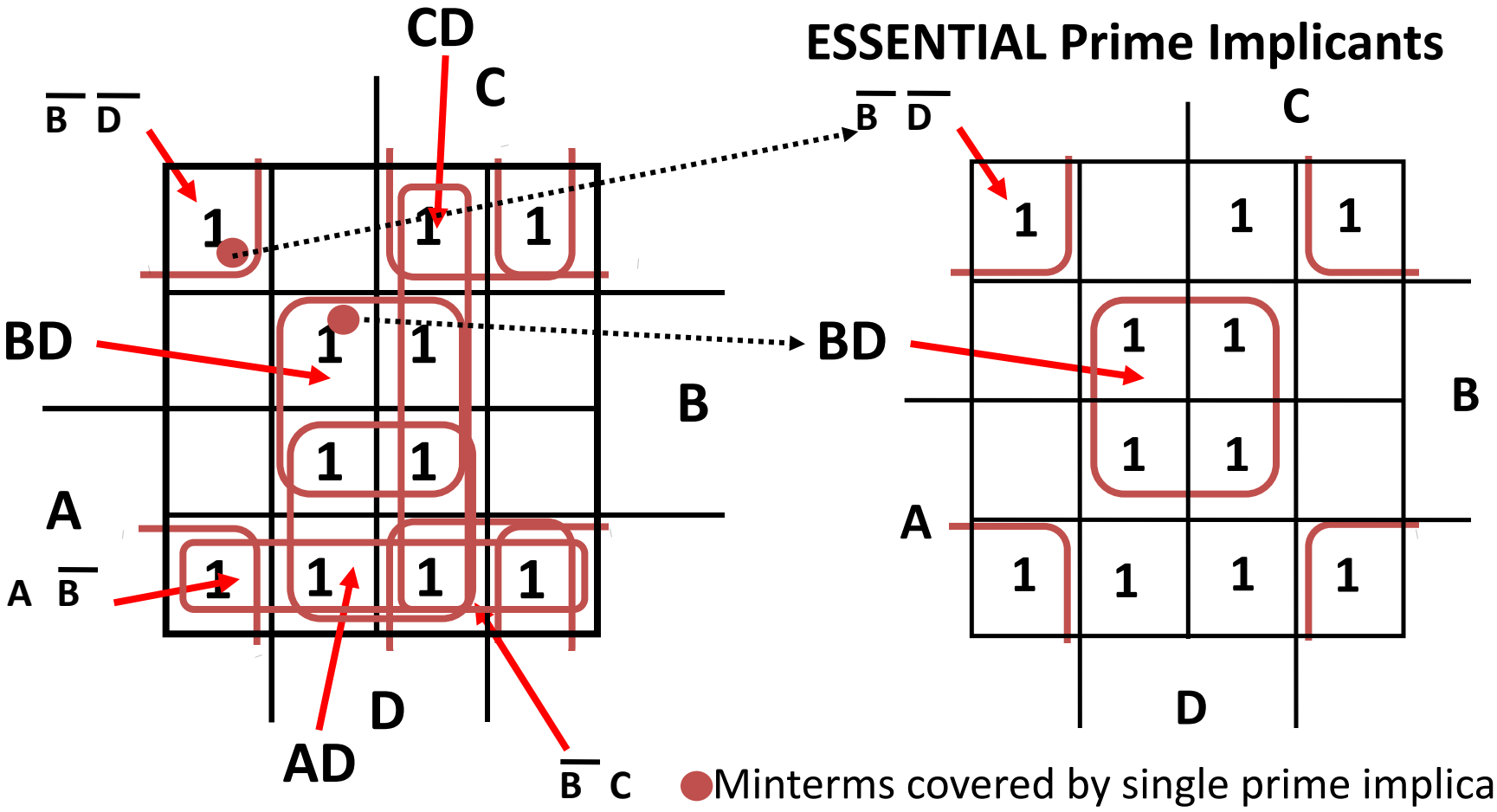
2-5 Map Manipulation

Systematic Simplification

- A *Prime Implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2.
- A prime implicant is called an *Essential Prime Implicant* if it is the only prime implicant that covers (includes) one or more minterms.
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map.
- A set of prime implicants "*covers all minterms*" if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm.

Example of Prime Implicants

- Find ALL Prime Implicants



Prime Implicant Practice

- Find all prime implicants for:

$$F(A, B, C, D) = \Sigma_m(0,2,3,8,9,10,11,12,13,14,15)$$

Another Example

- Find all prime implicants for:

$$G(A,B,C,D) = \Sigma_m(0,2,3,4,7,12,13,14,15)$$

  - Hint: There are seven prime implicants!

Don't Cares in K-Maps

- Sometimes a function table or map contains entries for which it is known:
  - the input values for the minterm will never occur, or
  - The output value for the minterm is not used
- In these cases, the output value need not be defined
- Instead, the output value is defined as a "don't care"
- By placing "don't cares" (an "x" entry) in the function table or map, the cost of the logic circuit may be lowered.
- Example 1: A logic function having the binary codes for the BCD digits as its inputs. Only the codes for 0 through 9 are used. The six codes, 1010 through 1111 never occur, so the output values for these codes are "x" to represent "don't cares."

Don't Cares in K-Maps

- Example 2: A circuit that represents a very common situation that occurs in computer design has two distinct sets of input variables:
  - A, B, and C which take on all possible combinations, and
  - Y which takes on values 0 or 1.

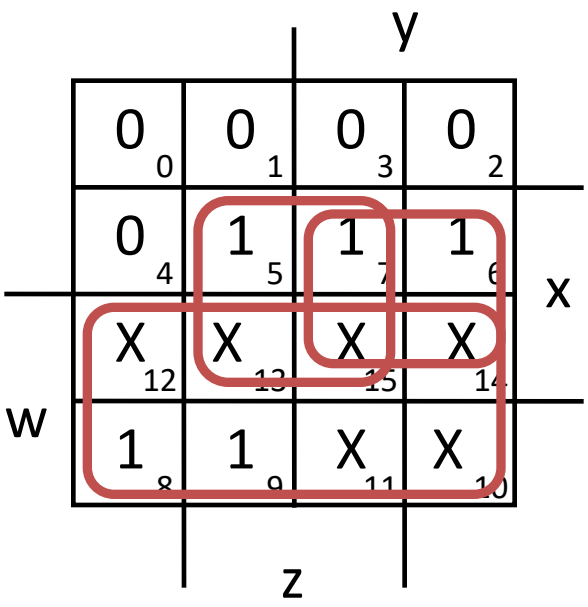
  and a single output Z. The circuit that receives the output Z observes it only for combinations of A, B, and C such A = 1 and B = 1 or C = 0, otherwise ignoring it. Thus, Z is specified only for those combinations, and for all other combinations of A, B, and C, Z is a don't care. Specifically, Z must be specified for $\overline{AB + \overline{C}} = 1$, and is a don't care for :

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

- Ultimately, each don't care "x" entry may take on either a 0 or 1 value in resulting solutions
- For example, an "x" may take on value "0" in an SOP solution and value "1" in a POS solution, or vice-versa.
- Any minterm with value "x" need not be covered by a prime implicant.

Example: BCD "5 or More"

- The map below gives a function F1(w,x,y,z) which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:



$$F1(w,x,y,z) = w + x\,z + x\,y \quad G = 7$$

» This is much lower in cost than F2 where the "don't cares" were treated as "0s."

$$F_2(w, x, y, z) = \overline{w}\,x\,z + \overline{w}\,x\,y + w\,\overline{x}\,\overline{y} \quad G = 12$$

» For this particular function, cost G for the POS solution for $F_1(w,x,y,z)$ is not changed by using the don't cares.

Product of Sums Example

- Find the optimum POS solution:

$$F(A,B,C,D) = \Sigma_m(3,9,11,12,13,14,15) + \Sigma d\,(1,4,6)$$

  - Hint: Use $\overline{F}$ and complement it to get the result.

Optimization Algorithm

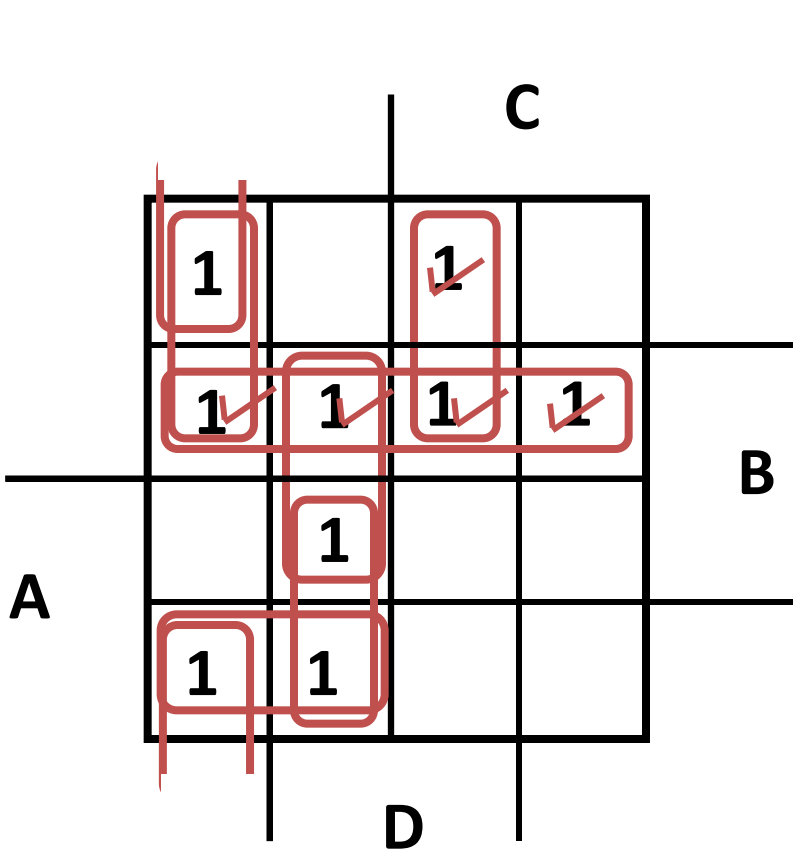
- Find all prime implicants.
- Include all essential prime implicants in the solution
- Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered:
  - Obtaining an optimum solution: See Reading Supplement - More on Optimization
  - Obtaining a good simplified solution: Use the Selection Rule

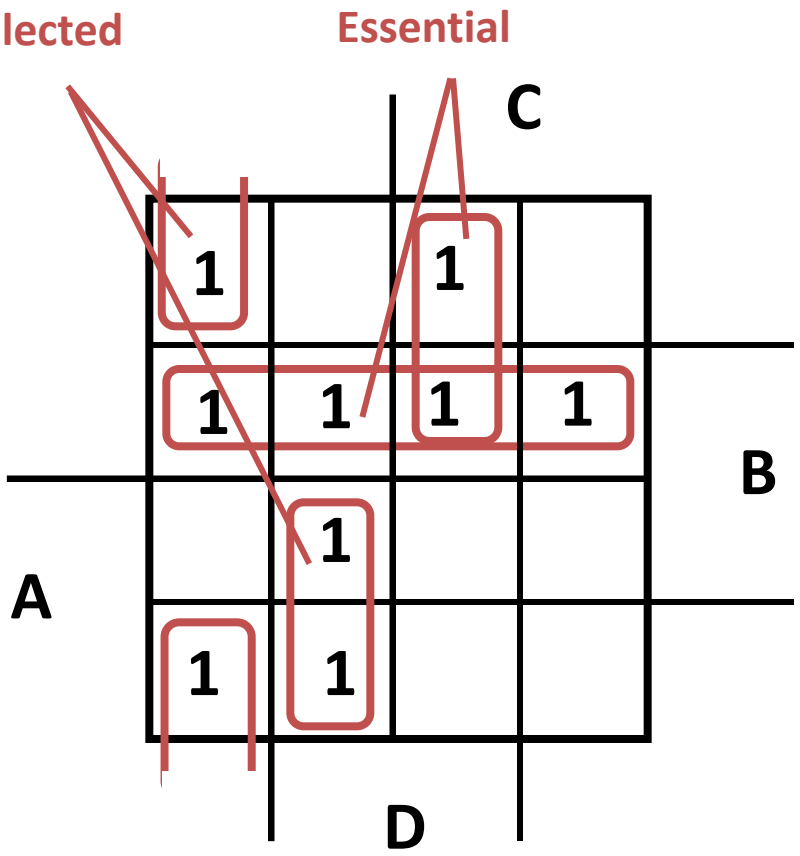
Prime Implicant Selection Rule

- Minimize the overlap among prime implicants as much as possible. In particular, in the final solution, make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected.

Selection Rule Example

- Simplify F(A, B, C, D) given on the K-map.

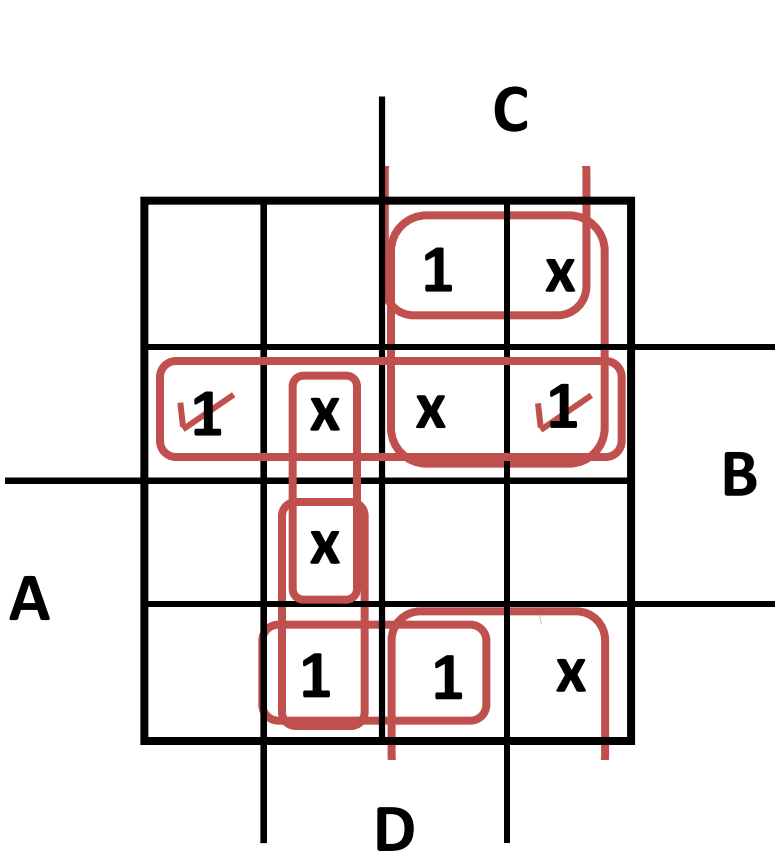


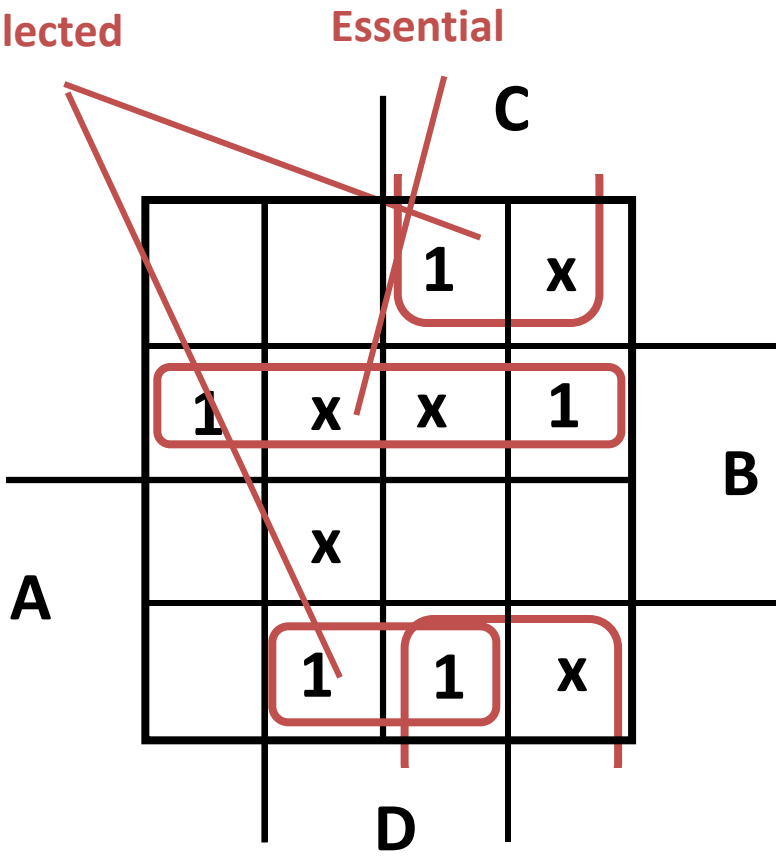


✓ Minterms covered by essential prime implicants

Selection Rule Example with Don't Cares

- Simplify F(A, B, C, D) given on the K-map.





✓ Minterms covered by essential prime implicants

Practical Optimization

- Problem: Automated optimization algorithms:
  - require minterms as starting point,
  - require determination of all prime implicants, and/or
  - require a selection process with a potentially very large number of candidate solutions to be found.
- Solution: Suboptimum algorithms not requiring any of the above in the general case

2-6 Exclusive-Or Operator and Gates

Exclusive OR/ Exclusive NOR

- The *eXclusive OR* (*XOR*) function is an important Boolean function used extensively in logic circuits.
- The XOR function may be;
  - implemented directly as an electronic circuit (truly a gate) or
  - implemented by interconnecting other gate types (used as a convenient representation)
- The *eXclusive NOR* function is the complement of the XOR function
- By our definition, XOR and XNOR gates are complex gates.

Exclusive OR/ Exclusive NOR

- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers
- Definitions
  - The XOR function is: $X \oplus Y = X\overline{Y} + \overline{X}Y$
  - The eXclusive NOR (XNOR) function, otherwise known as *equivalence* is: $\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}$
- Strictly speaking, XOR and XNOR gates do not exist for more that two inputs. Instead, they are replaced by odd and even functions.

Truth Tables for XOR/XNOR

- Operator Rules: XOR XNOR

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

X	Y	$\overline{(X \oplus Y)}$ or $X \equiv Y$
0	0	1
0	1	0
1	0	0
1	1	1

- The XOR function means:

  X OR Y, but NOT BOTH

- Why is the XNOR function also known as the *equivalence* function, denoted by the operator ≡?

XOR/XNOR (Continued)

- The XOR function can be extended to 3 or more variables. For more than 2 variables, it is called an *odd function* or *modulo 2 sum* (*Mod 2 sum*), not an XOR:

$$X \oplus Y \oplus Z = \overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + X\,Y\,Z$$

- The complement of the odd function is the even function.
- The XOR identities:

$$X \oplus 0 = X \qquad X \oplus 1 = \overline{X}$$

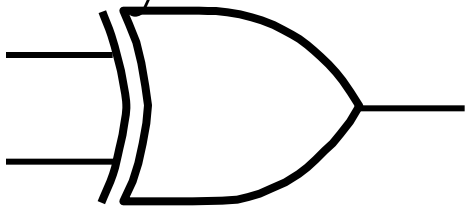
$$X \oplus X = 0 \qquad X \oplus \overline{X} = 1$$

$$X \oplus Y = Y \oplus X$$

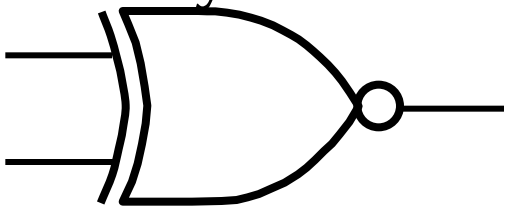
$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$$

Symbols For XOR and XNOR

- XOR symbol:



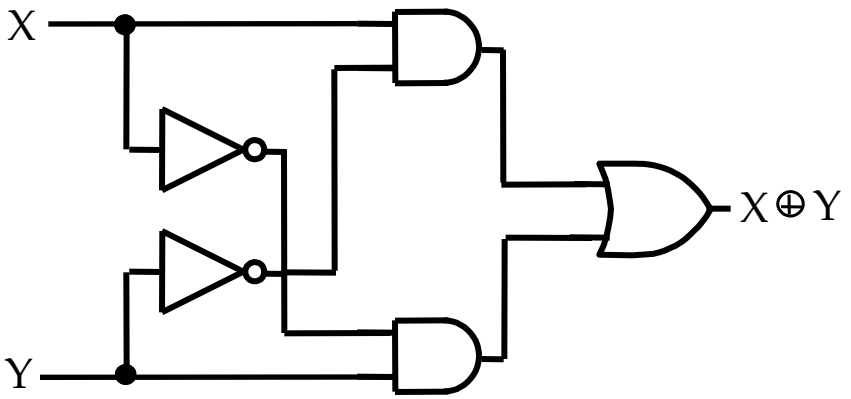
- XNOR symbol:



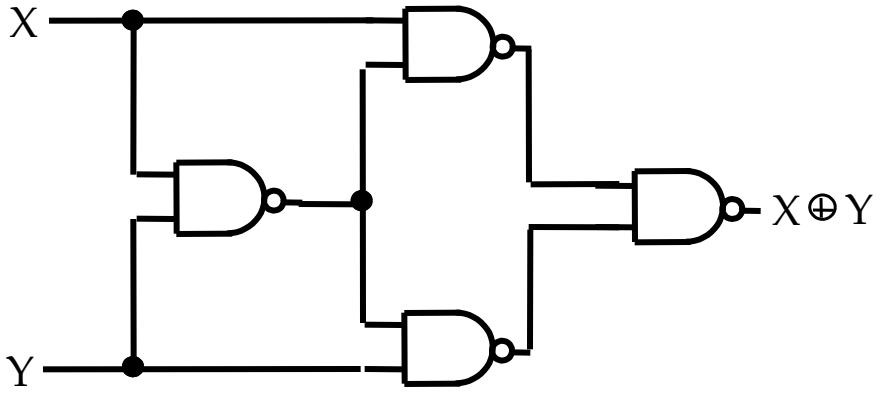
- Shaped symbols exist only for two inputs

XOR Implementations

- The simple SOP implementation uses the following structure:



- A NAND only implementation is:

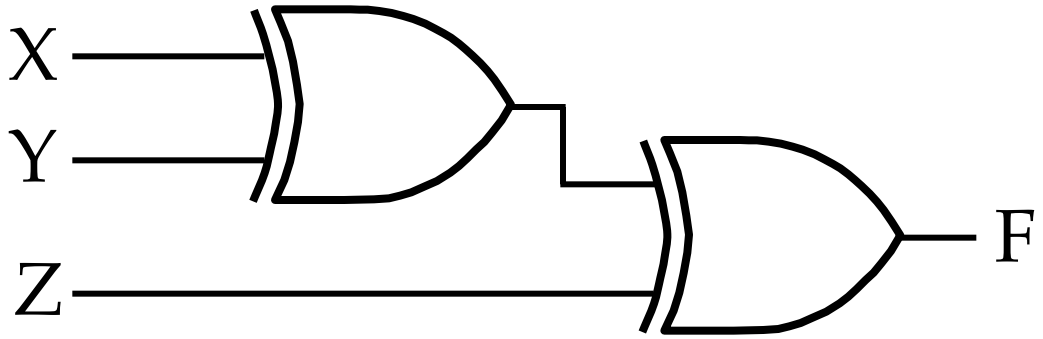


Odd and Even Functions

- The odd and even functions on a K-map form "checkerboard" patterns.
- The 1s of an odd function correspond to minterms having an index with an odd number of 1s.
- The 1s of an even function correspond to minterms having an index with an even number of 1s.
- Implementation of odd and even functions for greater than four variables as a two-level circuit is difficult, so we use "trees" made up of :
  - 2-input XOR or XNORs
  - 3- or 4-input odd or even functions

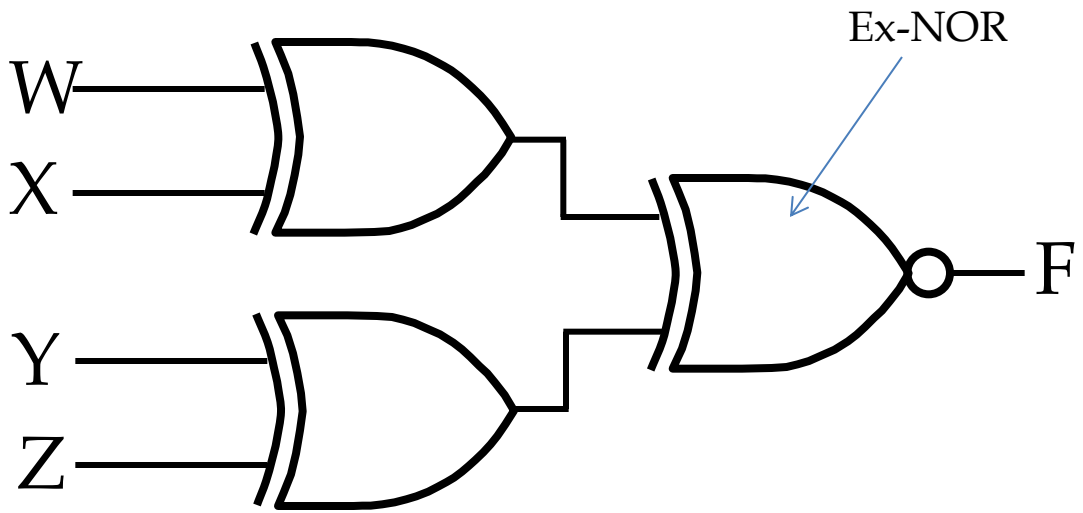
Example: Odd Function Implementation

- Design a 3-input odd function $F = X \oplus Y \oplus Z$ with 2-input XOR gates
- Factoring, $F = (X \oplus Y) \oplus Z$
- The circuit:



Example: Even Function Implementation

- Design a 4-input even function $F = W \oplus X \oplus Y \oplus Z$ with 2-input XOR and XNOR gates
- Factoring, $F = (W \oplus X) \oplus (Y \oplus Z)$
- The circuit:



Parity Generators and Checkers

- In Chapter 1, a parity bit added to n-bit code to produce an n + 1 bit code:
  - Add odd parity bit to generate code words with even parity
  - Add even parity bit to generate code words with odd parity
  - Use odd parity circuit to check code words with even parity
  - Use even parity circuit to check code words with odd parity
- Example: n = 3. Generate even parity code words of length four with odd parity generator:
- Check even parity code words of length four with odd parity checker:
- Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0. If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error.