DIGITAL DESIGN FUNDAMENTALS

Logical Gates

- Keep in mind that computers work on an electrical flow where a high voltage is considered a 1 and a low voltage is considered a 0. Using these highs and lows, data are represented.
- Electronic circuits must be designed to manipulate these positive and negative pulses into meaningful logic.
- Logic gates are the building blocks of digital circuits. Combinations of logic gates form circuits designed with specific tasks in mind. For example, logic gates are combined to form circuits to add binary numbers (adders), set and reset bits of memory (flip flops), multiplex multiple inputs, etc.

BINARY LOGIC AND GATES

- 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.

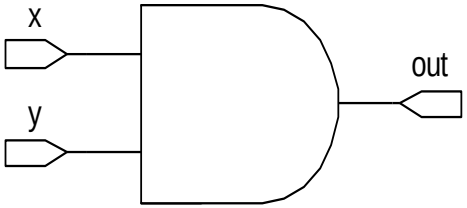
Binary Variables

- Recall that the two binary values have different names:
  - True/False
  - On/Off
  - Yes/No
  - 1/0
- We use 1 and 0 to denote the two values.
- Variable identifier examples:
  - A, B, y, z

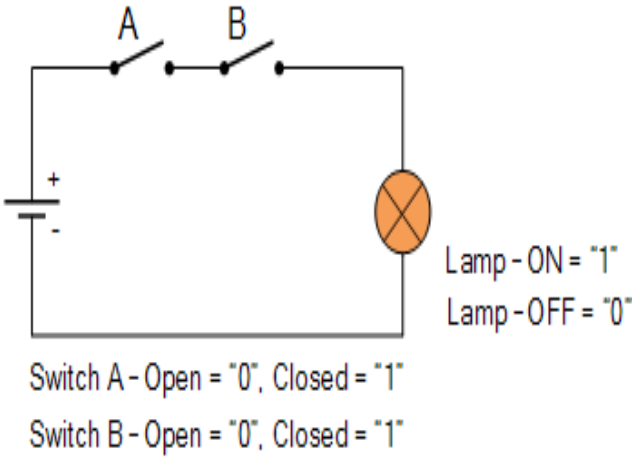
LOGICAL OPERATIONS

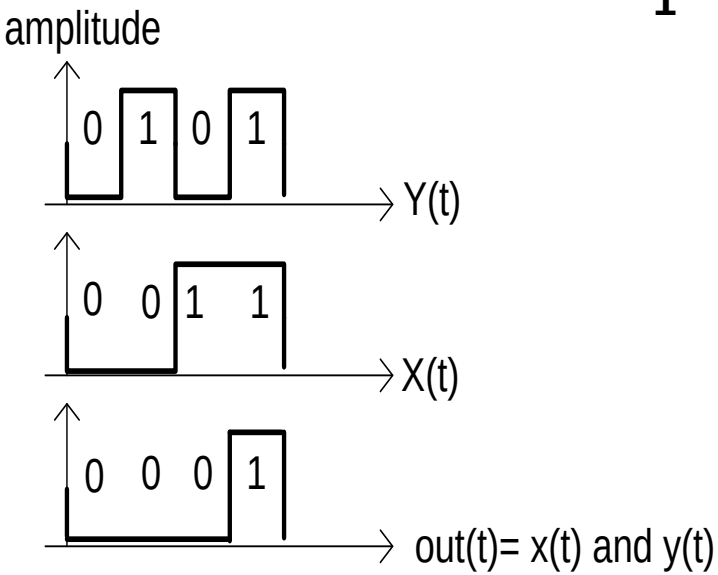
- The three basic logical operations are:
  - AND
  - OR
  - NOT
- AND is denoted by a dot (·).
- OR is denoted by a plus (+).
- NOT is denoted by an overbar (¯), a single quote mark (′) after, or (~) before the variable.

AND



x	y	out = xy
0	0	0
0	1	0
1	0	0
1	1	1

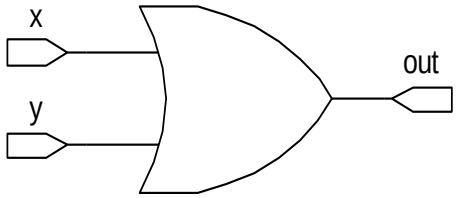


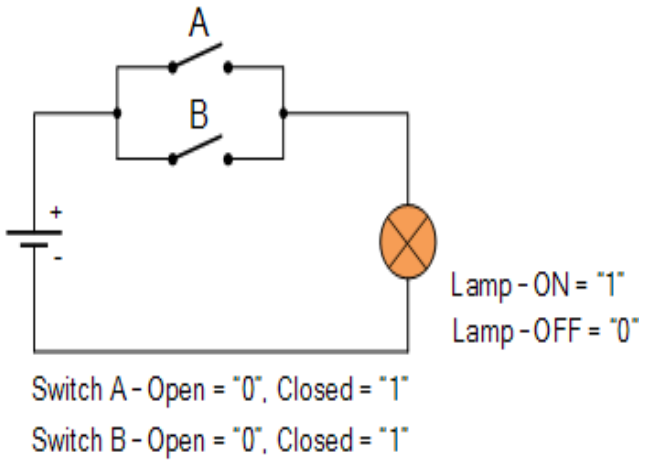


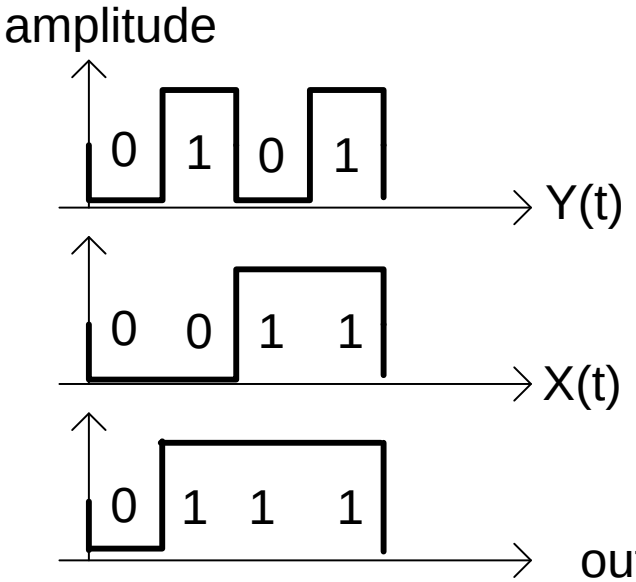
- Output is one if every input has value of 1
- More than two values can be "and-ed" together
- For example xyz = 1 only if x=1, y=1 and z=1

OR

x	y	out = x+y
0	0	0
0	1	1
1	0	1
1	1	1



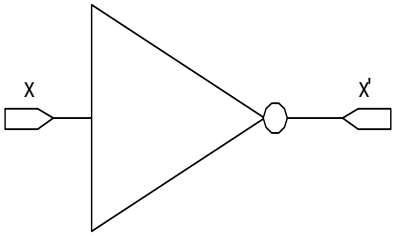


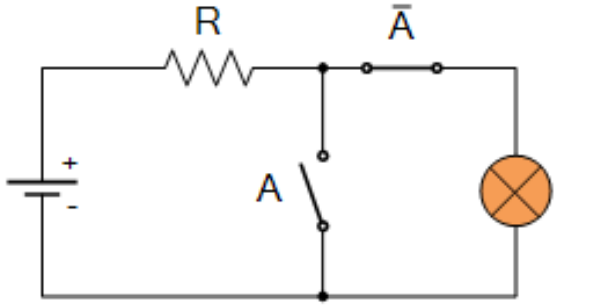


- Output is 1 if at least one input is 1.
- More than two values can be "or-ed" together.
- For example x+y+z = 1 if at least one of the three values is 1.

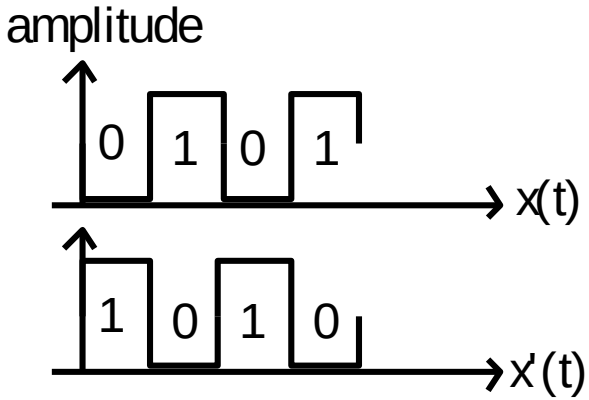
NOT

x	x'
0	1
1	0





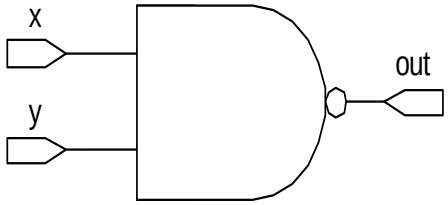
Switch A – Open = "0", Lamp – ON = "1"
Switch A – Closed = "1", Lamp – OFF = "0"

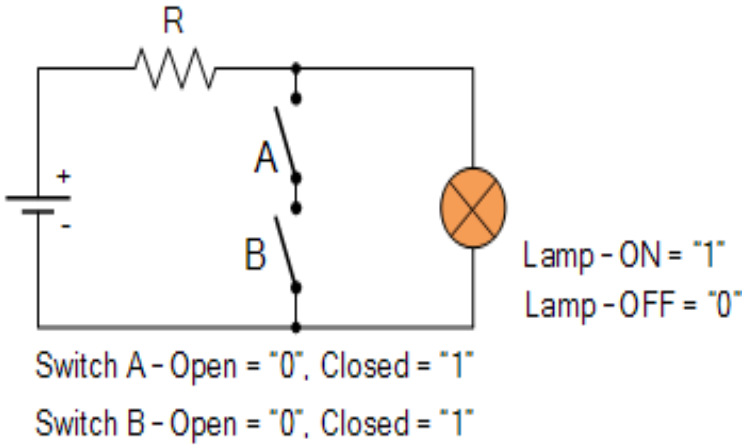


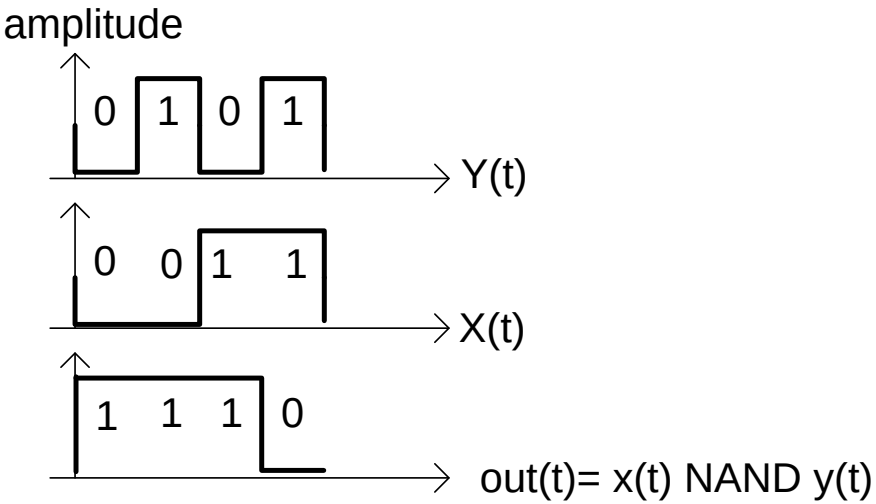
- This function operates on a single Boolean value.
- Its output is the complement of its input.
- An input of 1 produces an output of 0 and an input of 0 produces an output of 1

NAND

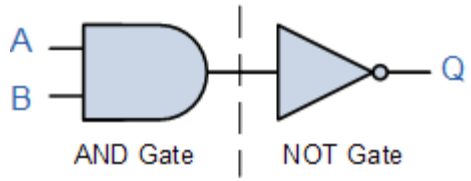
x	y	out = x NAND y
0	0	1
0	1	1
1	0	1
1	1	0





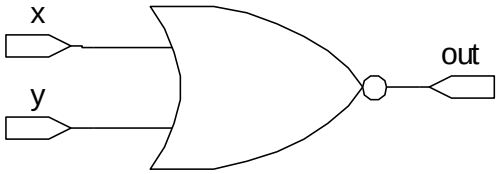


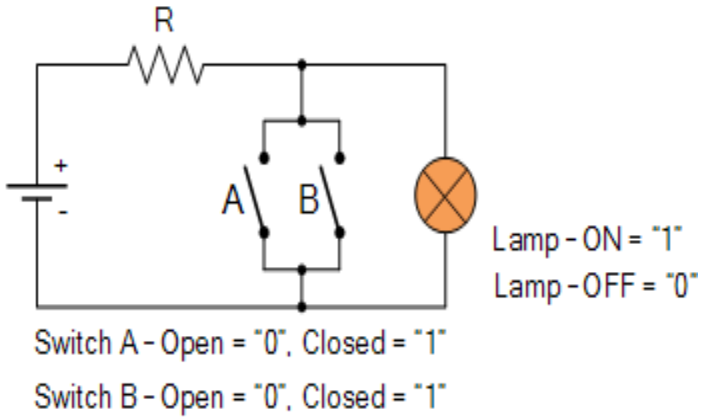
- Output value is the complemented output from an "AND" function.

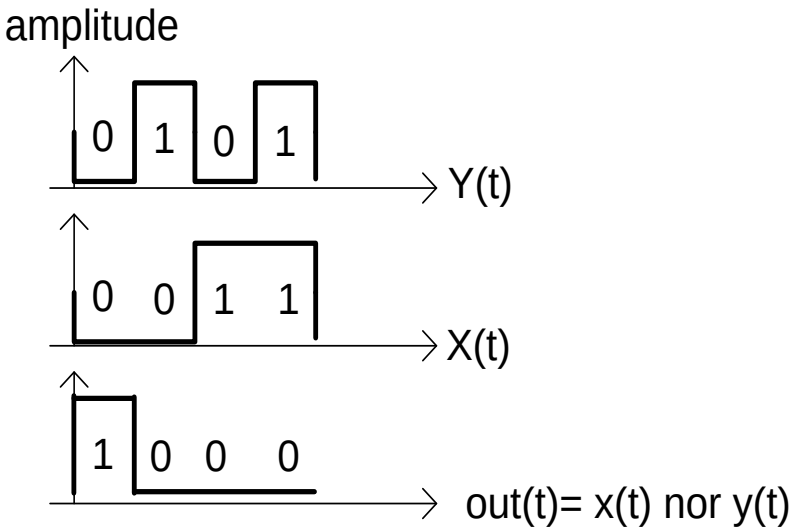


NOR

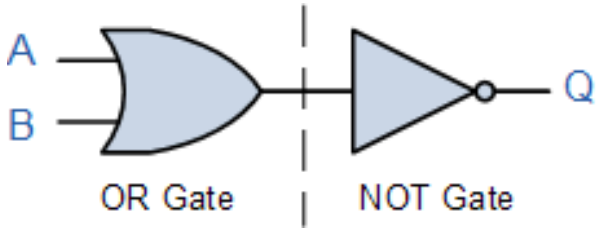
x	y	out = x NOR y
0	0	1
0	1	0
1	0	0
1	1	0





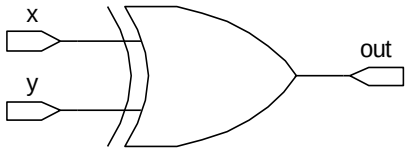


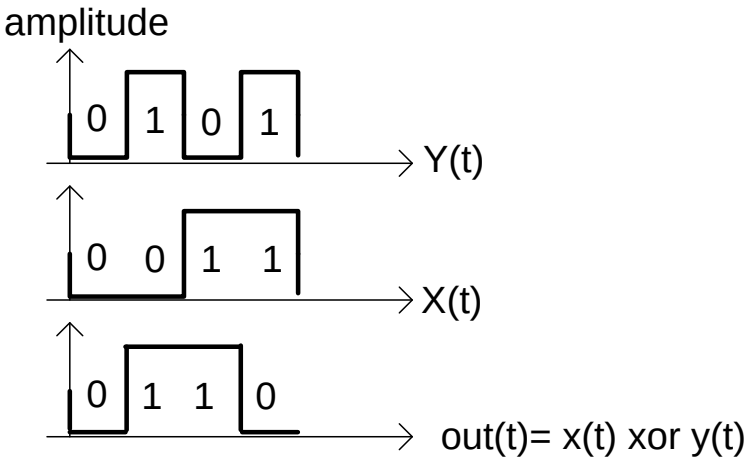
- Output value is the complemented output from an "OR" function.



XOR (EXCLUSIVE OR)

x	y	out = $x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

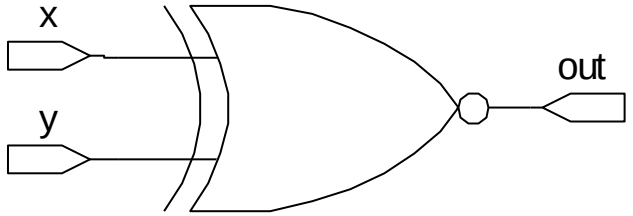


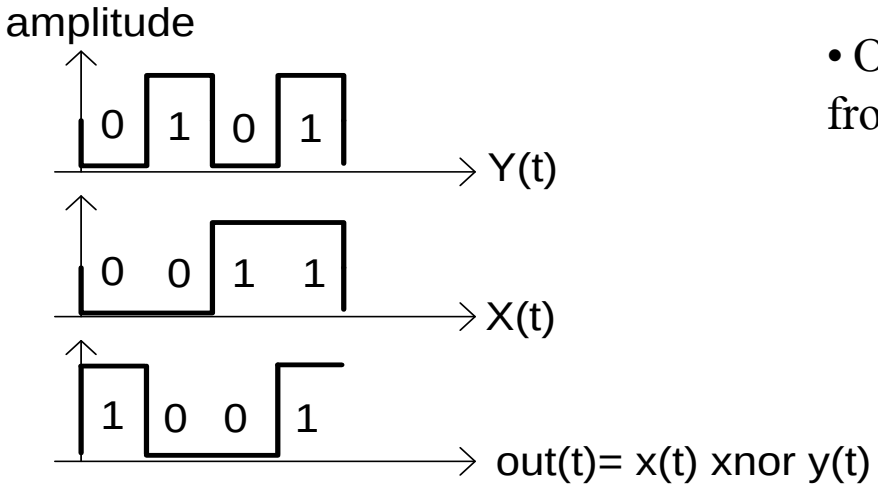


- The number of inputs that are 1 matter.
- More than two values can be "xor-ed" together.
- General rule: the output is equal to 1 if an odd number of input values are 1 and 0 if an even number of input values are 1.

XNOR

x	y	out =x xnor y
0	0	1
0	1	0
1	0	0
1	1	1





• Output value is the complemented output from an “XOR” function.

SUMMARY ON GATES

Inputs		Truth Table Outputs for each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Logic Function	Boolean Notation
AND	A.B
OR	A+B
NOT	A’
NAND	(A.B)’
NOR	(A+B)’
EX-OR	(A’.B) + (A.B’) or A⊕B
EX-NOR	(A.B) + (A’B’) or (A ⊕ B)’

TRUTH TABLES

- Used to evaluate any logic function
- Consider $F(X, Y, Z) = XY + \overline{Y}Z$

X	Y	Z	XY	$\overline{Y}$	$\overline{Y}Z$	$F = XY + \overline{Y}Z$
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	0
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

LOGIC DIAGRAMS AND EXPRESSIONS

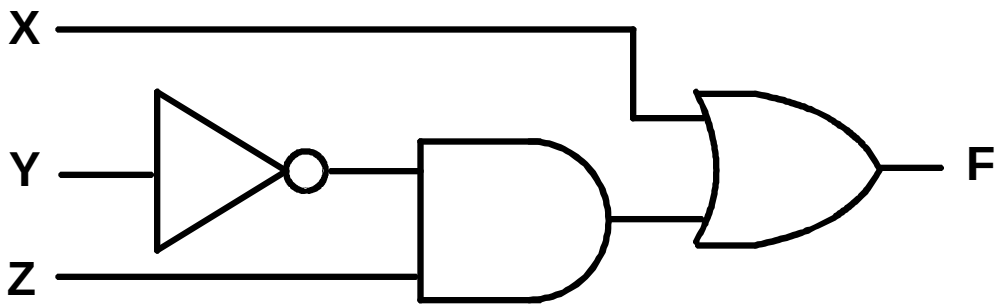
Truth Table

X Y Z	F = X + $\overline{Y}$ ×Z
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

Logic Equation

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

Logic Diagram



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

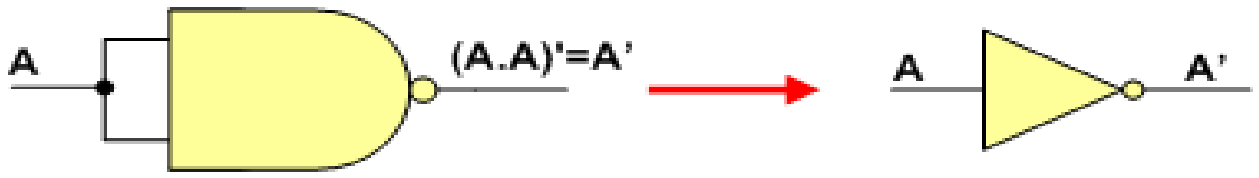
WHY UNIVERSAL GATES

- NAND and NOR both are called as universal gate. You can built any circuit or design using either NAND or NOR, this will be so flexible to fabricate any circuit in a single chip
- A universal gate is a gate which can implement any Boolean function without need to use any other gate type.
- In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.
- In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not the other way around!!
- Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter not the other way around!!

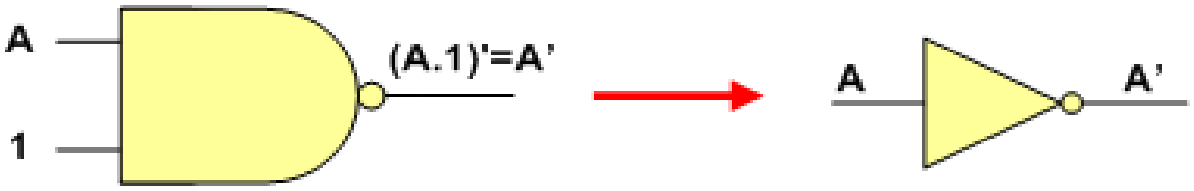
REALIZATION USING NAND

- **NOT GATE**

  All NAND input pins connect to the input signal A gives an output A'.



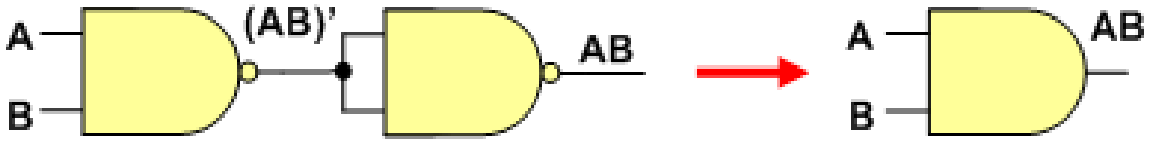
One NAND input pin is connected to the input signal A while all other input pins are connected to logic 1. The output will be A'.



REALIZATION USING NAND

- **AND GATE**

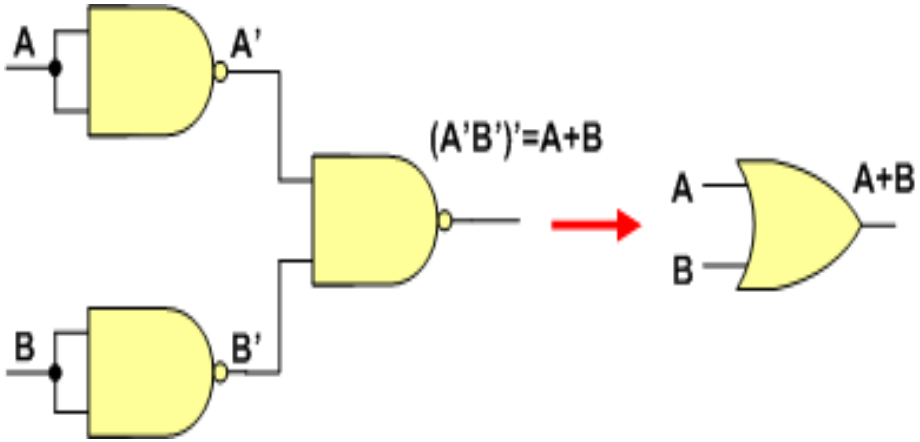
  An AND gate can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).



REALIZATION USING NAND

- **OR GATE**

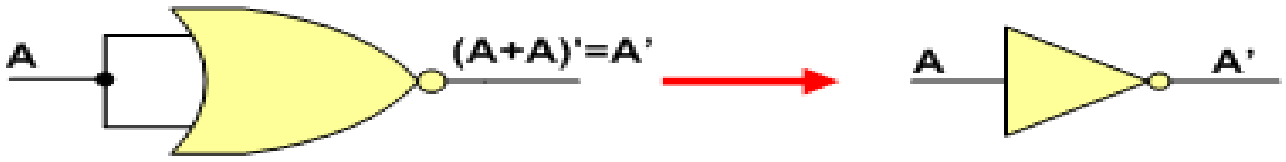
  An AND gate can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).



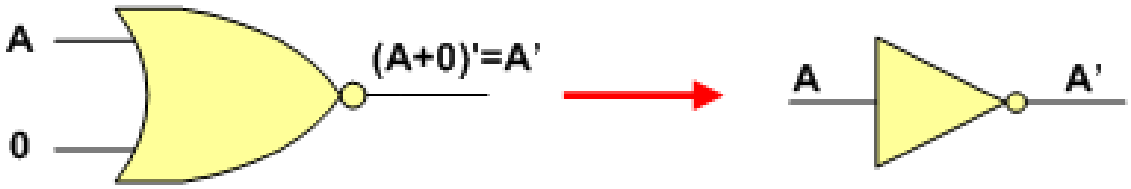
REALIZATION USING NOR

- **NOT GATE**

  All NOR input pins connect to the input signal A gives an output A'.



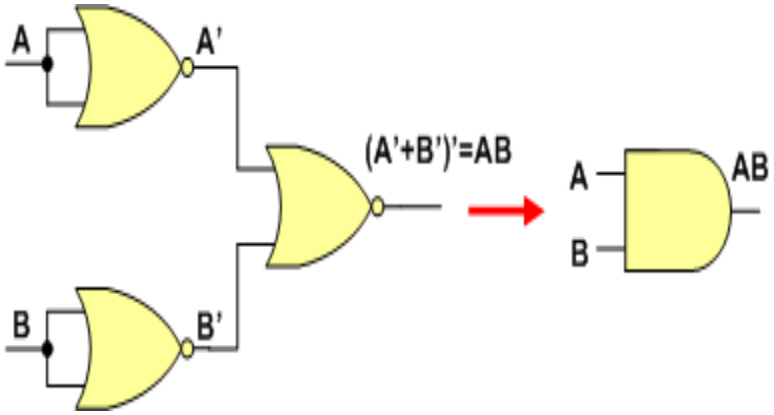
  One NOR input pin is connected to the input signal A while all other input pins are connected to logic 0. The output will be A'.



REALIZATION USING NOR

- **AND GATE**

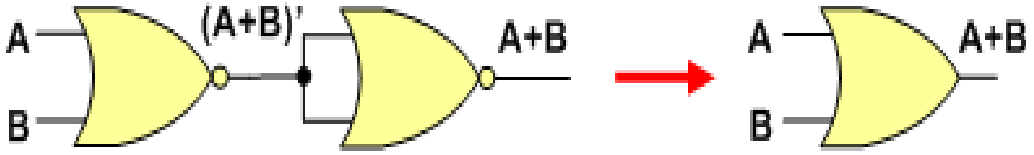
  An AND gate can be replaced by NOR gates as shown in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters)



REALIZATION USING NOR

- **OR GATE**

  An OR gate can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter)



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!

1854: Logical algebra was published by George Boole → known today as "Boolean Algebra"

  – It's a convenient way and systematic way of expressing and analyzing the operation of logic circuits.

1938: Claude Shannon was the first to apply Boole's work to the analysis and design of logic circuits.

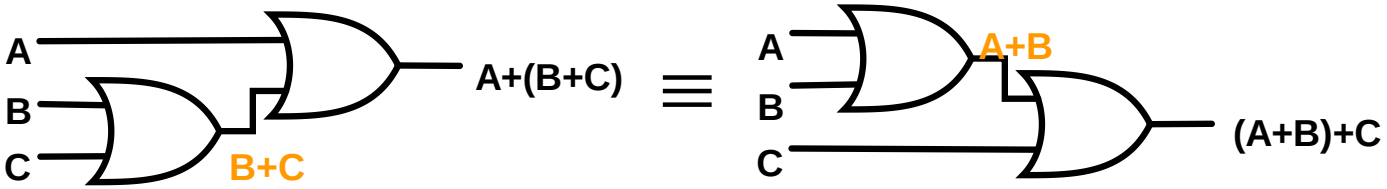
COMMUTATIVE LAWS

- The *commutative law of addition* for two variables is written as: $A+B = B+A$

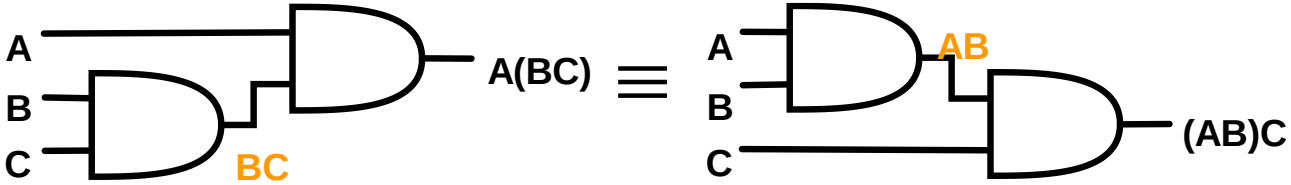
- The *commutative law of multiplication* for two variables is written as: $AB = BA$

ASSOCIATIVE LAWS

- The *associative law of addition* for 3 variables is written as: $A+(B+C) = (A+B)+C$

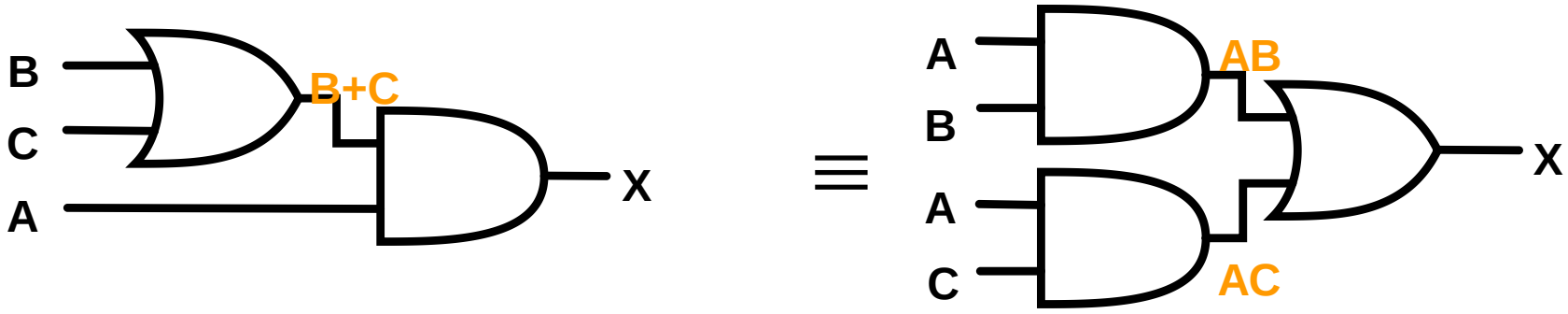


- The *associative law of multiplication* for 3 variables is written as: $A(BC) = (AB)C$



DISTRIBUTIVE LAWS

- The *distributive law* is written for 3 variables as follows:
  $A(B+C) = AB + AC$



X=A(B+C) **X=AB+AC**

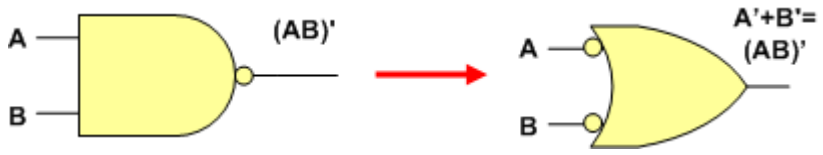
RULES OF BOOLEAN ALGEBRA

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \bullet 0 = 0$
4. $A \bullet 1 = A$
5. $A + A = A$
6. $A + \overline{A} = 1$
7. $A \bullet A = A$
8. $A \bullet \overline{A} = 0$
9. $\overline{\overline{A}} = A$
10. $A + AB = A$
11. $A + \overline{A}B = A + B$
12. $(A + B)(A + C) = A + BC$

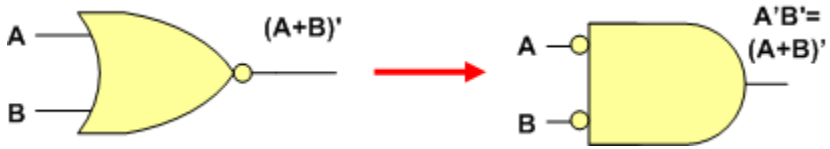
A, B, and C can represent a single variable or a combination of variables.

DEMORGAN'S THEOREMS

- DeMorgan's theorems provide mathematical verification of:
  - the equivalency of the NAND and negative-OR gates

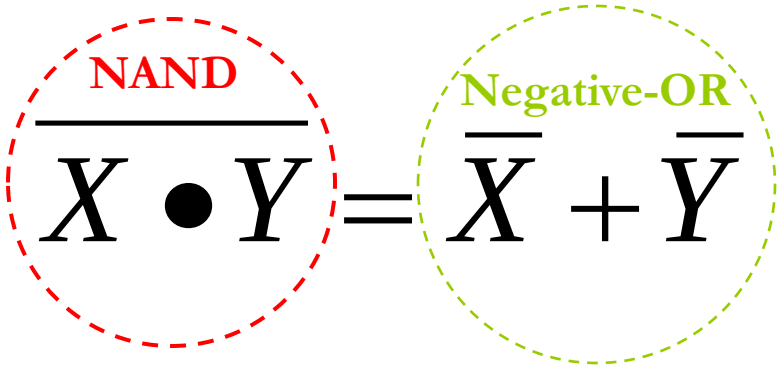


  - the equivalency of the NOR and negative-AND gates.

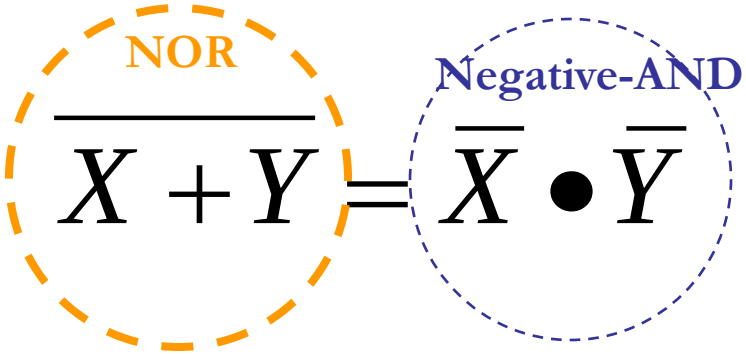


DEMORGAN'S THEOREMS

- The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.



- The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables.



DeMorgan's Theorems (Exercises)

- Apply DeMorgan's theorems to the expressions:

$$\overline{X \bullet Y \bullet Z}$$

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

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

$$\overline{\overline{W} \bullet \overline{X} \bullet \overline{Y} \bullet \overline{Z}}$$

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

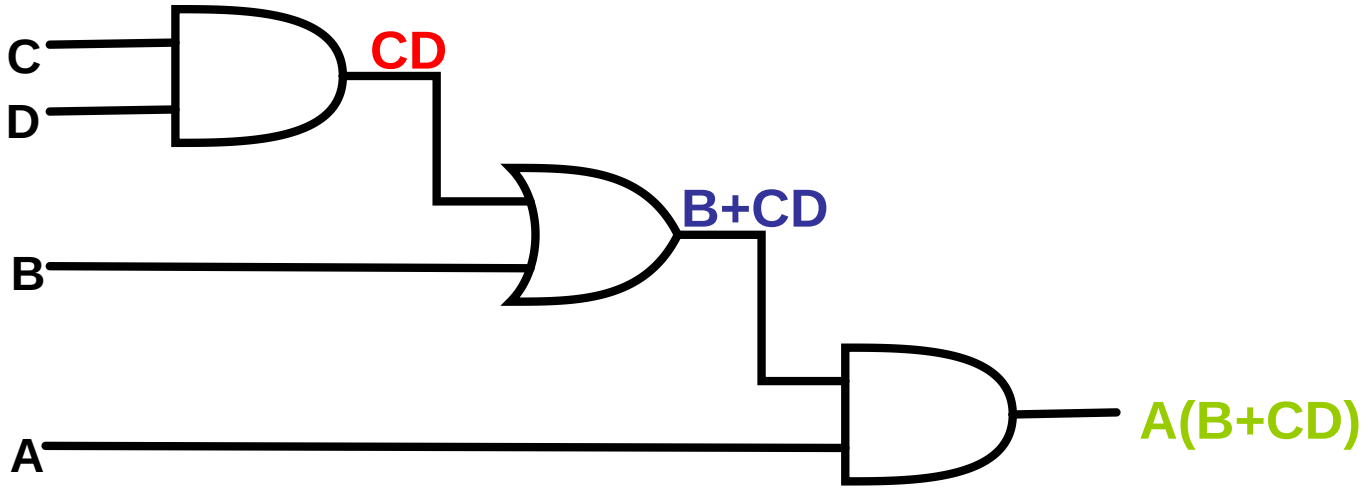
$$\overline{ABC + DEF}$$

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

$$\overline{A + B\overline{C} + D(\overline{E + \overline{F}})}$$

BOOLEAN EXPRESSION FOR A LOGIC CIRCUIT

- To derive the Boolean expression for a given logic circuit, begin at the left-most inputs and work toward the final output, writing the expression for each gate.



BOOLEAN ALGEBRA

- An algebraic structure defined by a set $B = \{0, 1\}$, together with two binary operators (+ and ·) and a unary operator ($\overline{\ }$)

1.	$X + 0 = X$	2.	$X \cdot 1 = X$	Identity element
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$	
5.	$X + X = X$	6.	$X \cdot X = X$	Idempotence
7.	$X + \overline{X} = 1$	8.	$X \cdot \overline{X} = 0$	Complement
9.	$\overline{\overline{X}} = X$			Involution
10.	$X + Y = Y + X$	11.	$XY = YX$	Commutative
12.	$(X + Y) + Z = X + (Y + Z)$	13.	$(XY)\,Z = X(Y\,Z)$	Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y)\,(X + Z)$	Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan's

SOME PROPERTIES OF BOOLEAN ALGEBRA

- Boolean Algebra is defined in general by a set *B* that can have more than two values
- A two-valued Boolean algebra is also know as Switching Algebra. The Boolean set *B* is restricted to 0 and 1. Switching circuits can be represented by this algebra.
- The dual of an algebraic expression is obtained by interchanging + and · and interchanging 0's and 1's.
- The identities appear in dual pairs. When there is only one identity on a line the identity is self-dual, i. e., the dual expression = the original expression.
- Sometimes, the dot symbol '•' (AND operator) is not written when the meaning is clear

DUAL OF A BOOLEAN EXPRESSION

- Example: $F = (A + \overline{C}) \cdot B + 0$

  dual $F = (A \cdot \overline{C} + B) \cdot 1 = A \cdot \overline{C} + B$
- Example: $G = X \cdot Y + \overline{(W + Z)}$

  dual $G = (X+Y) \cdot \overline{(W \cdot Z)} = (X+Y) \cdot (\overline{W}+\overline{Z})$
- Example: $H = A \cdot B + A \cdot C + B \cdot C$

  dual $H = (A+B) \cdot (A+C) \cdot (B+C)$
- Unless it happens to be self-dual, the dual of an expression does not equal the expression itself
- Are any of these functions self-dual? **H is self-dual**

  $(A+B)(A+C)(B+C)=(A+BC)(B+C)=AB+AC+BC$

BOOLEAN OPERATOR PRECEDENCE

- **The order of evaluation is:**

  1. Parentheses
  2. NOT
  3. AND
  4. OR

- **Consequence: Parentheses appear around OR expressions**
- **Example:** $F = A(B + C)(C + \overline{D})$

Boolean Algebraic Proof – Example 1

- $A + A \cdot B = A$ (Absorption Theorem)

Proof Steps	Justification
$A + A \cdot B$	
$= A \cdot 1 + A \cdot B$	Identity element: $A \cdot 1 = A$
$= A \cdot (1 + B)$	Distributive
$= A \cdot 1$	$1 + B = 1$
$= A$	Identity element

- 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.

Boolean Algebraic Proof – Example 2

- AB + A'C + BC = AB + A'C (Consensus Theorem)

Proof Steps	Justification
= AB + A'C + BC	
= AB + A'C + 1 · BC	Identity element
= AB + A'C + (A + A') · BC	Complement
= AB + A'C + ABC + A'BC	Distributive
= AB + ABC + A'C + A'CB	Commutative
= AB · 1 + ABC + A'C · 1 + A'CB	Identity element
= AB (1+C) + A'C (1 + B)	Distributive
= AB . 1 + A'C . 1	1+X = 1
= AB + A'C	Identity element

USEFUL THEOREMS

- Minimization
  $X\,Y + \overline{X}\,Y = Y$
- Absorption
  $X + X\,Y = X$
- Simplification
  $X + \overline{X}\,Y = X + Y$
- DeMorgan's
- $\overline{X + Y} = \overline{X} \cdot \overline{Y}$

- Minimization (dual)
  $(X+Y)(\overline{X}+Y) = Y$
- Absorption (dual)
  $X \cdot (X + Y) = X$
- Simplification (dual)
  $X \cdot (\overline{X} + Y) = X \cdot Y$
- DeMorgan's (dual)
- $\overline{X \cdot Y} = \overline{X} + \overline{Y}$

TRUTH TABLE TO VERIFY DEMORGAN'S

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

X	Y	X·Y	X+Y	$\overline{X}$	$\overline{Y}$	$\overline{X+Y}$	$\overline{X} \cdot \overline{Y}$	$\overline{X \cdot Y}$	$\overline{X}+\overline{Y}$
0	0	0	0	1	1	1	1	1	1
0	1	0	1	1	0	0	0	1	1
1	0	0	1	0	1	0	0	1	1
1	1	1	1	0	0	0	0	0	0

- Generalized DeMorgan's Theorem:

$$\overline{X_1 + X_2 + \ldots + X_n} = \overline{X_1} \cdot \overline{X_2} \cdot \ldots \cdot \overline{X_n}$$

$$\overline{X_1 \cdot X_2 \cdot \ldots \cdot X_n} = \overline{X_1} + \overline{X_2} + \ldots + \overline{X_n}$$

EXPRESSION SIMPLIFICATION

- An application of Boolean algebra
- Simplify to contain the smallest number of literals (variables that may or may not be complemented)

$$AB + \overline{A}CD + \overline{A}BD + \overline{A}C\overline{D} + ABCD$$

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

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

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

$= B(A + D) + \overline{A}C$ (has only 5 literals)

COMPLEMENTING FUNCTIONS

- Use DeMorgan's Theorem:
  1. Interchange AND and OR operators
  2. Complement each constant 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 $G = (a + bc)d + e$

  $\bar{G} = (a\,(\bar{b} + \bar{c}) + d)\,\bar{e}$

CANONICAL FORMS

- Minterms and Maxterms
- Sum-of-Minterm (SOM) Canonical Form
- Product-of-Maxterm (POM) Canonical Form
- Representation of Complements of Functions
- Conversions between Representations

MINTERMS

- Minterms are AND terms with every variable present in either true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g. x'), there are 2^n minterms for n variables.
- Example: Two variables (X and Y) produce 2 x 2 = 4 combinations:

  xy (both normal)

  xy' (X normal, Y complemented)

  x'y (X complemented, Y normal)

  x'y' (both complemented)
- Thus there are four minterms of two variables.

MAXTERMS

- Maxterms are OR terms with every variable in true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n maxterms for n variables.
- Example: Two variables (X and Y) produce $2 \times 2 = 4$ combinations:

$X + Y$ (both normal)

$\overline{X} + Y$ (x normal, y complemented)

$X + \overline{Y}$ (x complemented, y normal)

$\overline{X} + \overline{Y}$ (both complemented)

MINTERMS & MAXTERMS FOR 2 VARIABLES

- Two variable minterms and maxterms.

x	y	Index	Minterm	Maxterm
0	0	0	$m_0 = \overline{x}\,\overline{y}$	$M_0 = x + y$
0	1	1	$m_1 = \overline{x}\,y$	$M_1 = x + \overline{y}$
1	0	2	$m_2 = x\,\overline{y}$	$M_2 = \overline{x} + y$
1	1	3	$m_3 = x\,y$	$M_3 = \overline{x} + \overline{y}$

- The minterm m_i should evaluate to 1 for each combination of x and y.
- The maxterm is the complement of the minterm

MINTERMS & MAXTERMS FOR 3 VARIABLES

x	y	z	Index	Minterm	Maxterm
0	0	0	0	m0 = $\bar{x}\,\bar{y}\,\bar{z}$	M0 = x + y + z
0	0	1	1	m1 = $\bar{x}\,\bar{y}$ z	M1 = x + y + $\bar{z}$
0	1	0	2	m2 = $\bar{x}$ y $\bar{z}$	M2 = x + $\bar{y}$ + z
0	1	1	3	m3 = $\bar{x}$ y z	M3 = x + $\bar{y}$ + $\bar{z}$
1	0	0	4	m4 = x $\bar{y}$ $\bar{z}$	M4 = $\bar{x}$ + y + z
1	0	1	5	m5 = x $\bar{y}$ z	M5 = $\bar{x}$ + y + $\bar{z}$
1	1	0	6	m6 = x y $\bar{z}$	M6 = $\bar{x}$ + $\bar{y}$ + z
1	1	1	7	m7 = x y z	M7 = $\bar{x}$ + $\bar{y}$ + $\bar{z}$

Maxterm M_i is the complement of minterm m_i

$M_i = \overline{m_i}$ and $mi = \overline{M_i}$

PURPOSE OF THE INDEX

- Minterms and Maxterms are designated with an index
- The index number corresponds to a binary pattern
- The index for the minterm or maxterm, expressed as a binary number, is used to determine whether the variable is shown in the true 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".

STANDARD ORDER

- All variables should be present in a minterm or maxterm and should be listed in the same order (usually alphabetically)
- Example: For variables a, b, c:
  - Maxterms $(a + b + c)$, $(\bar{a} + \bar{b} + \bar{c})$ are in standard order
  - However, $(b + a + c)$ is NOT in standard order

    $(a + \bar{c})$ does NOT contain all variables
  - Minterms $(a\ b\ \bar{c})$ and $(a\ \bar{b}\ c)$ are in standard order
  - However, $(b\ \bar{a}\ c)$ is not in standard order

    $(a\ \bar{c})$ does not contain all variables

SUM-OF-MINTERM (SOM)

- Sum-Of-Minterm (SOM) canonical form:
  Sum of minterms of entries that evaluate to '**1**'

x	y	z	F	Minterm
0	0	0	0	
0	0	1	**1**	$m_1 = \bar{x}\,\bar{y}\,z$
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	**1**	$m_6 = x\,y\,z$
1	1	1	**1**	$m_7 = x\,y\,\bar{z}$

FOCUS ON THE '**1**' ENTRIES

$$F = m1 + m6 + m7 = \sum (1, 6, 7) = \bar{x}\,\bar{y}\,z + x\,y\,\bar{z} + x\,y\,z$$

SUM-OF-MINTERM EXAMPLES

- $F(a, b, c, d) = \sum(2, 3, 6, 10, 11)$
- $F(a, b, c, d) = m_2 + m_3 + m_6 + m_{10} + m_{11}$

  $\bar{a}\,\bar{b}\,c\,\bar{d} + \bar{a}\,\bar{b}\,c\,d + \bar{a}\,b\,c\,\bar{d} + a\,\bar{b}\,c\,\bar{d} + a\,\bar{b}\,c\,d$
- $G(a, b, c, d) = \sum(0, 1, 12, 15)$
- $G(a, b, c, d) = m_0 + m_1 + m_{12} + m_{15}$

  $\bar{a}\,\bar{b}\,\bar{c}\,\bar{d} + \bar{a}\,\bar{b}\,\bar{c}\,d + a\,b\,\bar{c}\,\bar{d} + a\,b\,c\,d$

PRODUCT-OF-MAXTERM (POM)

- Product-Of-Maxterm (POM) canonical form:

  Product of maxterms of entries that evaluate to '**0**'

x	y	z	F	Maxterm
0	0	0	1	
0	0	1	1	
0	1	0	**0**	$M_2 = (x + \bar{y} + z)$
0	1	1	1	
1	0	0	**0**	$M_4 = (\bar{x} + y + z)$
1	0	1	1	
1	1	0	**0**	$M_6 = (\bar{x} + \bar{y} + z)$
1	1	1	1	

FOCUS ON THE '**0**' ENTRIES

$$F = M2 \cdot M4 \cdot M6 = \prod (2, 4, 6) = (x+\bar{y}+z)\ (\bar{x}+y+z)\ (\bar{x}+\bar{y}+z)$$

PRODUCT-OF-MAXTERM EXAMPLES

- $F(a, b, c, d) = \prod(1, 3, 6, 11)$
- $F(a, b, c, d) = M_1 \cdot M_3 \cdot M_6 \cdot M_{11}$

$(a+b+c+\bar{d})\ (a+b+\bar{c}+\bar{d})\ (a+\bar{b}+\bar{c}+d)\ (\bar{a}+b+\bar{c}+\bar{d})$

- $G(a, b, c, d) = \prod(0, 4, 12, 15)$
- $G(a, b, c, d) = M_0 \cdot M_4 \cdot M_{12} \cdot M_{15}$

$(a+b+c+d)(a+\bar{b}+c+d)\ (\bar{a}+\bar{b}+c+d)(\bar{a}+\bar{b}+\bar{c}+\bar{d})$

OBSERVATIONS

- We can implement any function by "ORing" the minterms corresponding to the '**1**' entries in the function table. A minterm evaluates to '**1**' for its corresponding entry.
- We can implement any function by "ANDing" the maxterms corresponding to '**0**' entries in the function table. A maxterm evaluates to '**0**' for its corresponding entry.
- The same Boolean function can be expressed in two canonical ways: Sum-of-Minterms (SOM) and Product-of-Maxterms (POM).
- If a Boolean function has fewer '**1**' entries then the SOM canonical form will contain fewer literals than POM. However, if it has fewer '**0**' entries then the POM form will have fewer literals than SOM.

CONVERTING TO SUM-OF-MINTERMS FORM

- A function that is not in the Sum-of-Minterms form can be converted to that form by means of a truth table
- Consider $F = \bar{y} + \bar{x}\,\bar{z}$

x	y	z	F	Minterm
0	0	0	**1**	$m_0 = \bar{x}\,\bar{y}\,\bar{z}$
0	0	1	**1**	$m_1 = \bar{x}\,\bar{y}\,z$
0	1	0	**1**	$m_2 = \bar{x}\,y\,\bar{z}$
0	1	1	0	
1	0	0	**1**	$m_4 = x\,\bar{y}\,\bar{z}$
1	0	1	**1**	$m_5 = x\,\bar{y}\,z$
1	1	0	0	
1	1	1	0	

$F = \sum(0, 1, 2, 4, 5) =$

$m0 + m1 + m2 + m4 + m5 =$

$\bar{x}\,\bar{y}\,\bar{z} + \bar{x}\,\bar{y}\,z + \bar{x}\,y\,\bar{z} + x\,\bar{y}\,\bar{z} + x\,\bar{y}\,z$

CONVERTING TO PRODUCT-OF-MAXTERMS FORM

- A function that is not in the Product-of-Minterms form can be converted to that form by means of a truth table
- Consider again: $F = \bar{y} + \bar{x}\,\bar{z}$

x	y	z	F	Minterm
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	**0**	$M_3 = (x+\bar{y}+\bar{z})$
1	0	0	1	
1	0	1	1	
1	1	0	**0**	$M_6 = (\bar{x}+\bar{y}+z)$
1	1	1	**0**	$M_7 = (\bar{x}+\bar{y}+\bar{z})$

$F = \prod(3, 6, 7) =$

$M3 \cdot M6 \cdot M7 =$

$(x+\bar{y}+\bar{z})\ (\bar{x}+\bar{y}+z)\ (\bar{x}+\bar{y}+\bar{z})$

CONVERSIONS BETWEEN CANONICAL FORMS

x	y	z	F	Minterm	Maxterm
0	0	0	**0**		$M_0 = (x + y + z)$
0	0	1	**1**	$m_1 = \overline{x}\,\overline{y}\,z$	
0	1	0	**1**	$m_2 = \overline{x}\,y\,\overline{z}$	
0	1	1	**1**	$m_3 = \overline{x}\,y\,z$	
1	0	0	**0**		$M_4 = (\overline{x} + y + z)$
1	0	1	**1**	$m_5 = x\,\overline{y}\,z$	
1	1	0	**0**		$M_6 = (\overline{x} + \overline{y} + z)$
1	1	1	**1**	$m_7 = x\,y\,z$	

$$F = m_1 + m_2 + m_3 + m_5 + m_7 = \sum(1, 2, 3, 5, 7) =$$
$$\overline{x}\,\overline{y}\,z + \overline{x}\,y\,\overline{z} + \overline{x}\,y\,z + x\,\overline{y}\,z + x\,y\,z$$

$$F = M_0 \cdot M_4 \cdot M_6 = \prod(0, 4, 6) = (x+y+z)(\overline{x}+y+z)(\overline{x}+\overline{y}+z)$$

ALGEBRAIC CONVERSION TO SUM-OF-MINTERMS

- Expand all terms first to explicitly list all minterms
- AND any term missing a variable v with $(v + \bar{v})$
- Example 1: $f = x + \bar{x}\,\bar{y}$ (2 variables)

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

  $f = x\,y + x\,\bar{y} + \bar{x}\,\bar{y}$

  $f = m_3 + m_2 + m_0 = \sum(0, 2, 3)$
- Example 2: $g = a + b\,\bar{c}$ (3 variables)

  $g = a(b + \bar{b})(c + \bar{c}) + (a + \bar{a})\,b\,\bar{c}$

  $g = a\,b\,c + a\,b\,\bar{c} + a\,\bar{b}\,c + a\,\bar{b}\,\bar{c} + a\,b\,\bar{c} + \bar{a}\,b\,\bar{c}$

  $g = \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + a\,\bar{b}\,c + a\,b\,\bar{c} + a\,b\,c$

  $g = m_2 + m_4 + m_5 + m_6 + m_7 = \sum(2, 4, 5, 6, 7)$

ALGEBRAIC CONVERSION TO PRODUCT-OF-MAXTERMS

- Expand all terms first to explicitly list all maxterms
- OR any term missing a variable v with $v \cdot \bar{v}$
- Example 1: $f = x + \bar{x}\bar{y}$ (2 variables)

  Apply 2nd distributive law:

  $f = (x + \bar{x})(x + \bar{y}) = 1 \cdot (x + \bar{y}) = (x + \bar{y}) = M_1$
- Example 2: $g = a\bar{c} + bc + \bar{a}\bar{b}$ (3 variables)

  $g = (a\bar{c} + bc + \bar{a})(a\bar{c} + bc + \bar{b})$ (distributive)

  $g = (\bar{c} + bc + \bar{a})(a\bar{c} + c + \bar{b})$ $(x + \bar{x}y = x + y)$

  $g = (\bar{c} + b + \bar{a})(a + c + \bar{b})$ $(x + \bar{x}y = x + y)$

  $g = (\bar{a} + b + \bar{c})(a + \bar{b} + c) = M_5 \cdot M_2 = \prod(2, 5)$

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 form
- 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 $F(x, y, z) = \sum (1, 3, 5, 7)$

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

  $F(x, y, z) = \prod (0, 2, 4, 6)$

- A Simplification Example:

$$F(A,B,C) = \sum(1,4,5,6,7)$$

- Writing the minterm expression:

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

- Simplifying:

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

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

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

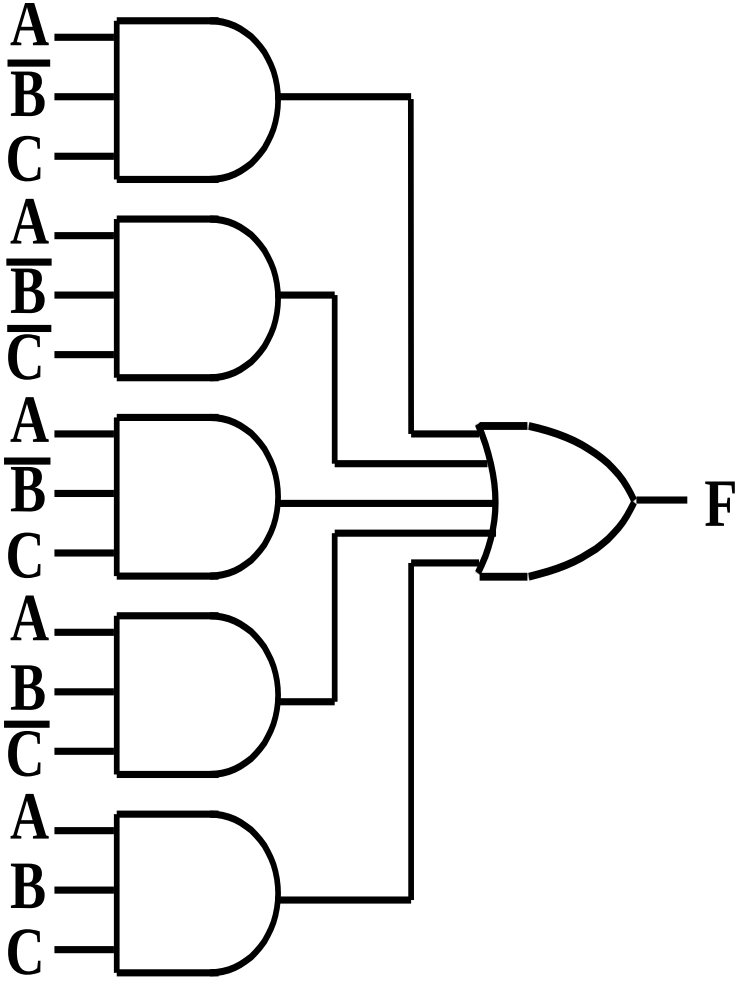
$F = \overline{A}\,\overline{B}\,C + A$

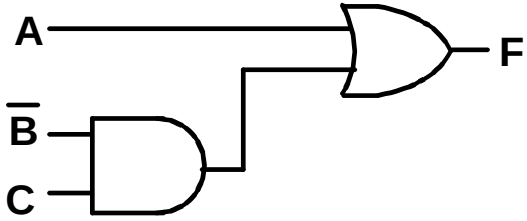
$F = \overline{B}\,C + A$

- Simplified F contains 3 literals

AND/OR TWO-LEVEL IMPLEMENTATION

- The two implementations for F are shown below





IT IS QUITE APPARENT WHICH IS SIMPLER!

EXAMPLE

- Minimize the following Boolean function using sum of products (SOP):

- f(a,b,c,d) = ∑m(3,7,11,12,13,14,15)

	abcd	
3	0011	**a`b`cd**
7	0111	**a`bcd**
11	1011	**ab`cd**
12	1100	**abc`d`**
13	1101	**abc`d**
14	1110	**abcd`**
15	1111	**abcd**

$f(a,b,c,d) = \sum m(3,7,11,12,13,14,15)$

$= a`b`cd + a`bcd + ab`cd + abc`d` + abc`d + abcd` + abcd$

$= cd(a`b` + a`b + ab`) + ab(c`d` + c`d + cd` + cd)$

$= cd(a`[b` + b] + ab`) + ab(c`[d` + d] + c[d` + d])$

$= cd(a`[1] + ab`) + ab(c`[1] + c[1])$

$= ab + ab`cd + a`cd$

$= ab + cd(ab` + a`)$

$= ab + cd(a + a`)(a` + b`)$

$= ab + a`cd + b`cd$

$= ab + cd(a` + b`)$

EXAMPLE

$f(a,b,c,d) = \prod M(0,1,2,4,5,6,8,9,10)$

$= \sum m(3,7,11,12,13,14,15)$

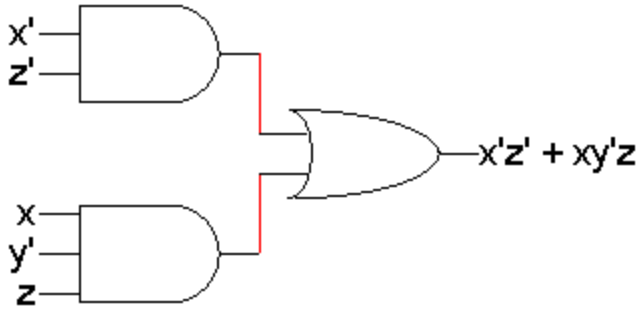
=[(a+b+c+d)(a+b+c+d`)(a+b`+c`+d`)

(a`+b+c`+d`)(a`+b`+c+ d)(a`+b`+c+ d`)

(a`+b`+c`+d)(a`+b`+c`+d`)]

KARNAUGH MAPS

- A graphical technique for simplifying an expression into a minimal sum of products (MSP) form:
  - There are a minimal number of product terms in the expression.
  - Each term has a minimal number of literals.
- Circuit-wise, this leads to a *minimal* two-level implementation.



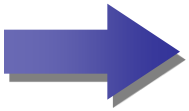
KARNAUGH MAPS (K-MAPS)

- Karnaugh maps -- A tool for representing Boolean functions of up to six variables.
- K-maps are tables of rows and columns with entries represent 1`s or 0`s of SOP and POS representations.
- An n-variable K-map has 2n cells with each cell corresponding to an n-variable truth table value.
- K-map cells are labeled with the corresponding truth-table row.
- K-map cells are arranged such that adjacent cells correspond to truth rows that differ in only one bit position (logical adjacency).
- If m_i is a minterm of f, then place a 1 in cell i of the K-map.
- If M_i is a maxterm of f, then place a 0 in cell i.
- If d_i is a don't care of f, then place a d or x in cell i.

2-Variable K-Map

- A two-variable function has four possible minterms. We can re-arrange these minterms into a Karnaugh map.

x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy



x \ y	0	1
0	x'y'	x'y
1	xy'	xy

- Now we can easily see which minterms contain common literals.
  - Minterms on the left and right sides contain y' and y respectively.
  - Minterms in the top and bottom rows contain x' and x respectively.

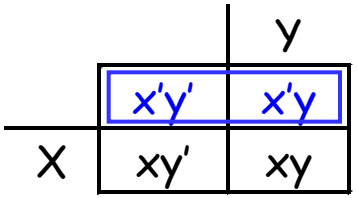
x \ y	0	1
0	x'y'	x'y
1	xy'	xy

	y'	y
x'	x'y'	x'y
x	xy'	xy

KARNAUGH MAP SIMPLIFICATIONS

- Imagine a two-variable sum of minterms:

  x'y' + x'y



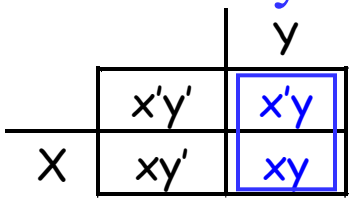
- Both of these minterms appear in the top row of a Karnaugh map, which means that they both contain the literal x'.

- What happens if you simplify this expression using Boolean algebra?

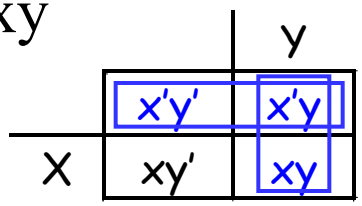
x'y' + x'y	= x'(y' + y)	[Distributive]
	= x' • 1	[y + y' = 1]
	= x'	[x • 1 = x]

MORE TWO-VARIABLE EXAMPLES

- Another example expression is x'y + xy.



  - Both minterms appear in the right side, where y is uncomplemented.
  - Thus, we can reduce x'y + xy to just y.
- How about x'y' + x'y + xy



  - We have x'y' + x'y in the top row, corresponding to x'.
  - There's also x'y + xy in the right side, corresponding to y.
  - This whole expression can be reduced to x' + y.

A Three-Variable Karnaugh Map

- For a three-variable expression with inputs x, y, z, the arrangement of minterms is more tricky:

X \ YZ	00	01	11	10
0	x′y′z′	x′y′z	x′yz	x′yz′
1	xy′z′	xy′z	xyz	xyz′

X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

- Another way to label the K-map (use whichever you like):

X \ Y	x′y′z′	x′y′z	x′yz	x′yz′
	xy′z′	xy′z	xyz	xyz′

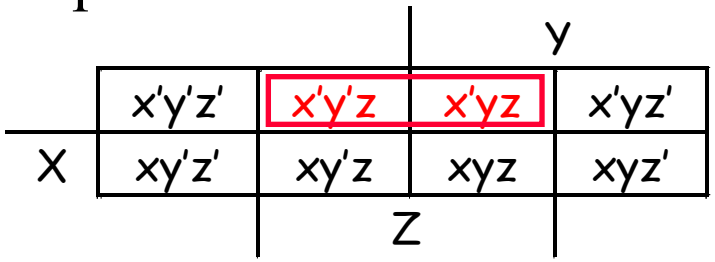
Z

X \ Y	m_0	m_1	m_3	m_2
	m_4	m_5	m_7	m_6

Z

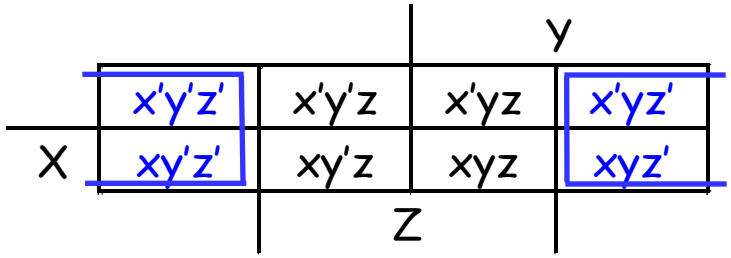
Why the funny ordering?

- With this ordering, any group of 2, 4 or 8 adjacent squares on the map contains common literals that can be factored out.



x'y'z + x'yz
= x'z(y' + y)
= x'z • 1
= x'z

- "Adjacency" includes wrapping around the left and right sides:



x'y'z' + xy'z' + x'yz' + xyz'
= z'(x'y' + xy' + x'y + xy)
= z'(y'(x' + x) + y(x' + x))
= z'(y'+y)
= z'

- We'll use this property of adjacent squares to do our simplifications.

EXAMPLE K-MAP SIMPLIFICATION

- Let's consider simplifying f(x,y,z) = xy + y'z + xz.
- First, you should convert the expression into a sum of minterms form, if it's not already.
  - The easiest way to do this is to make a truth table for the function, and then read off the minterms.
  - You can either write out the literals or use the minterm shorthand.
- Here is the truth table and sum of minterms for our example:

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

f(x,y,z) = x'y'z + xy'z + xyz' + xyz
= m_1 + m_5 + m_6 + m_7

Unsimplifying Expressions

- You can also convert the expression to a sum of minterms with Boolean algebra.
  - Apply the distributive law in reverse to add in missing variables.
  - Very few people actually do this, but it's occasionally useful.

$$
\begin{aligned}
xy + y'z + xz &= (xy \bullet 1) + (y'z \bullet 1) + (xz \bullet 1) \\
&= (xy \bullet (z' + z)) + (y'z \bullet (x' + x)) + (xz \bullet (y' + y)) \\
&= (xyz' + xyz) + (x'y'z + xy'z) + (xy'z + xyz) \\
&= xyz' + xyz + x'y'z + xy'z
\end{aligned}
$$

- In both cases, we're actually "unsimplifying" our example expression.
  - The resulting expression is larger than the original one!
  - But having all the individual minterms makes it easy to combine them together with the K-map.

MAKING THE EXAMPLE K-MAP

- Next up is drawing and filling in the K-map.
  - Put 1s in the map for each minterm, and 0s in the other squares.
  - You can use either the minterm products or the shorthand to show you where the 1s and 0s belong.
- In our example, we can write f(x,y,z) in two equivalent ways.

$f(x,y,z) = x'y'z + xy'z + xyz' + xyz$

		y		
	x'y'z'	x'y'z	x'yz	x'yz'
X	xy'z'	xy'z	xyz	xyz'
		z		

$f(x,y,z) = m_1 + m_5 + m_6 + m_7$

		y		
	m_0	m_1	m_3	m_2
X	m_4	m_5	m_7	m_6
		z		

- In either case, the resulting K-map is shown below.

		y		
	0	1	0	0
X	0	1	1	1
		z		

K-MAPS FROM TRUTH TABLES

- You can also fill in the K-map directly from a truth table.
  - The output in row *i* of the table goes into square m_i of the K-map.
  - Remember that the rightmost columns of the K-map are "switched."

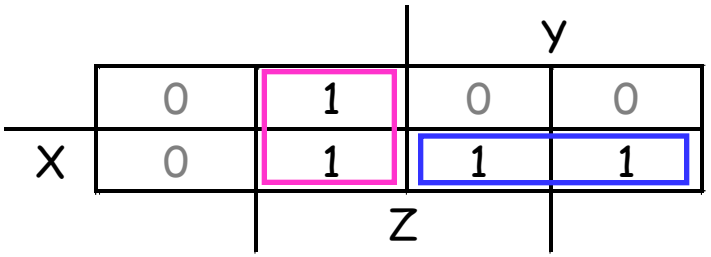
x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

	y			
	0	1	0	0
X	0	1	1	1
		Z		

	y			
	m_0	m_1	m_3	m_2
X	m_4	m_5	m_7	m_6
		Z		

GROUPING THE MINTERMS TOGETHER

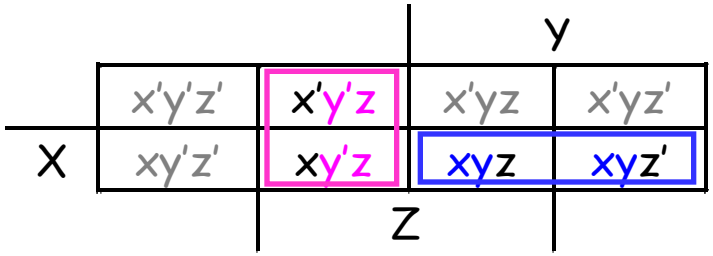
- The most difficult step is grouping together all the 1s in the K-map.
  - Make rectangles around groups of one, two, four or eight 1s.
  - All of the 1s in the map should be included in at least one rectangle.
  - Do *not* include any of the 0s.

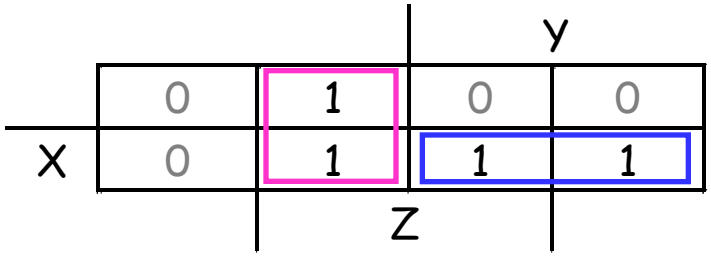


- Each group corresponds to one product term. For the simplest result:
  - Make as few rectangles as possible, to minimize the number of products in the final expression.
  - Make each rectangle as large as possible, to minimize the number of literals in each term.
  - It's all right for rectangles to overlap, if that makes them larger.

READING THE MSP FROM THE K-MAP

- Finally, you can find the MSP.
  - Each rectangle corresponds to one product term.
  - The product is determined by finding the common literals in that rectangle.

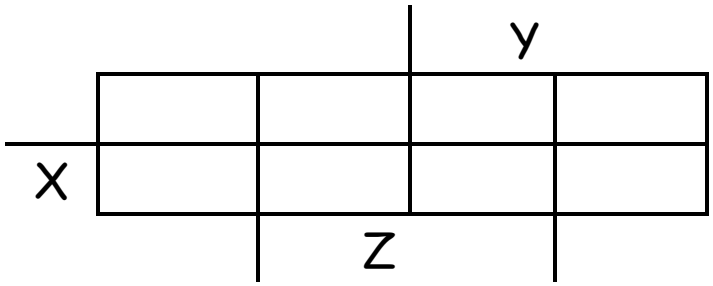


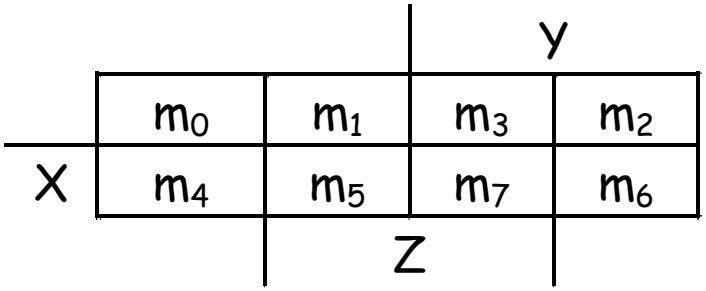


- For our example, we find that xy + y'z + xz = y'z + xy. (This is one of the additional algebraic laws from last time.)

PRACTICE K-MAP 1

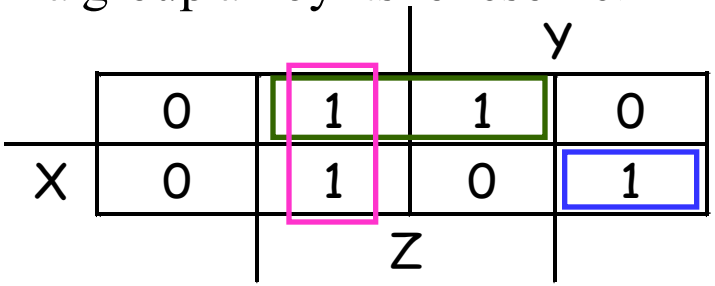
- Simplify the sum of minterms $m_1 + m_3 + m_5 + m_6$.





Solutions for Practice K-map 1

- Here is the filled in K-map, with all groups shown.
  - The magenta and green groups overlap, which makes each of them as large as possible.
  - Minterm m_6 is in a group all by its lonesome.



- Final MSP here is x'z + y'z + xyz'.

Four-Variable K-Maps

- We can do four-variable expressions too!
  - The minterms in the third and fourth columns, *and* in the third and fourth rows, are switched around.
  - Again, this ensures that adjacent squares have common literals.

W \ YZ	00	01	11	10
00	w′x′y′z′	w′x′y′z	w′x′yz	w′x′yz′
01	w′xy′z′	w′xy′z	w′xyz	w′xyz′
11	wxy′z′	wxy′z	wxyz	wxyz′
10	wx′y′z′	wx′y′z	wx′yz	wx′yz′

X

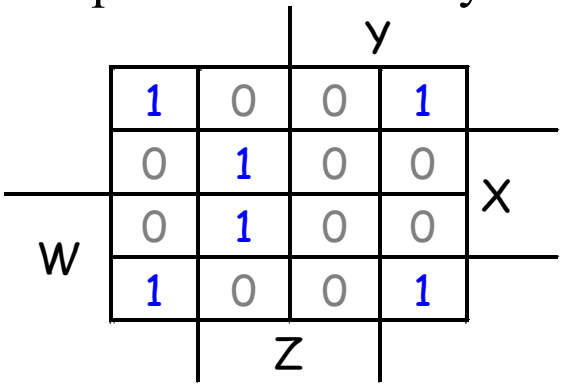
W \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

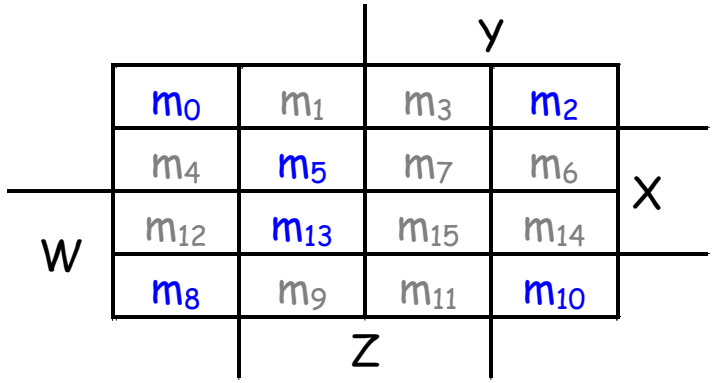
X

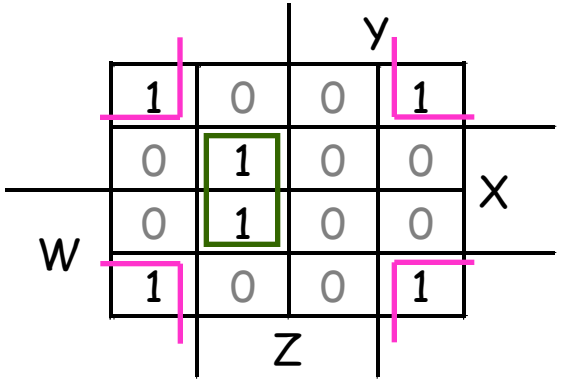
- Grouping minterms is similar to the three-variable case, but:
  - You can have rectangular groups of 1, 2, 4, 8 or 16 minterms.
  - You can wrap around *all four* sides.

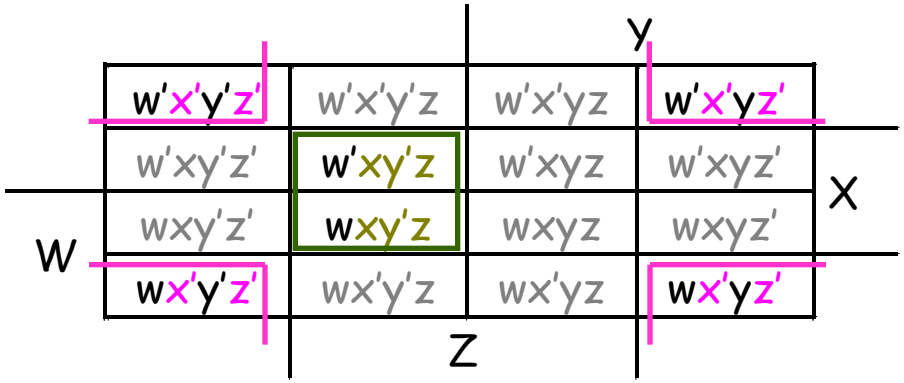
Example: Simplify $M_0+M_2+M_5+M_8+M_{10}+M_{13}$

- The expression is already a sum of minterms, so here's the K-map:





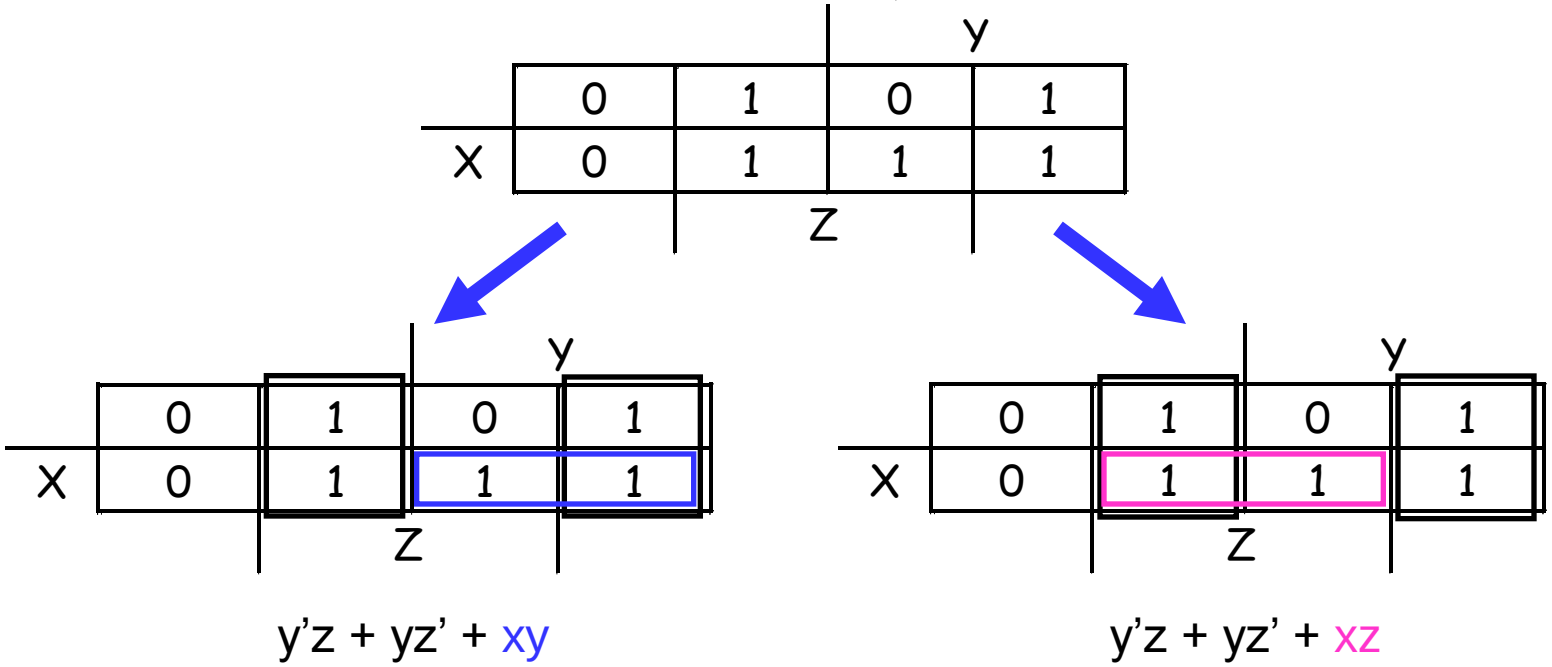




- We can make the following groups, resulting in the MSP x'z' + xy'z.

K-MAPS CAN BE TRICKY!

- There may not necessarily be a *unique* MSP. The K-map below yields two valid and equivalent MSPs, because there are two possible ways to include minterm m_7.



y'z + yz' + xy

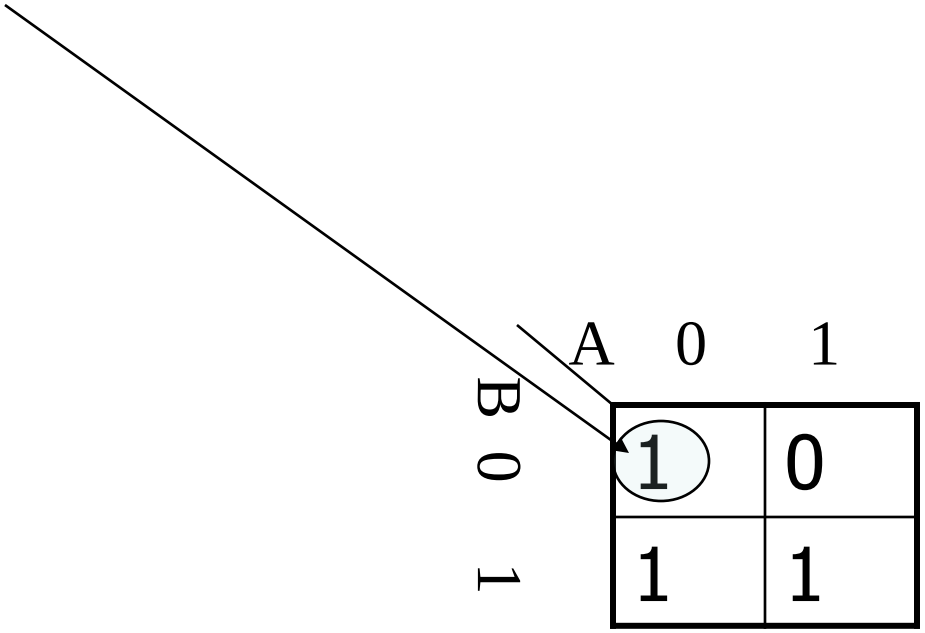
y'z + yz' + xz

- Remember that overlapping groups is possible, as shown above.

EXAMPLES

- *Two variable K-map*

  $f(A,B)=\sum m(0,1,3)=A`B`+A`B+AB$



EXAMPLES-THREE VARIABLE MAP

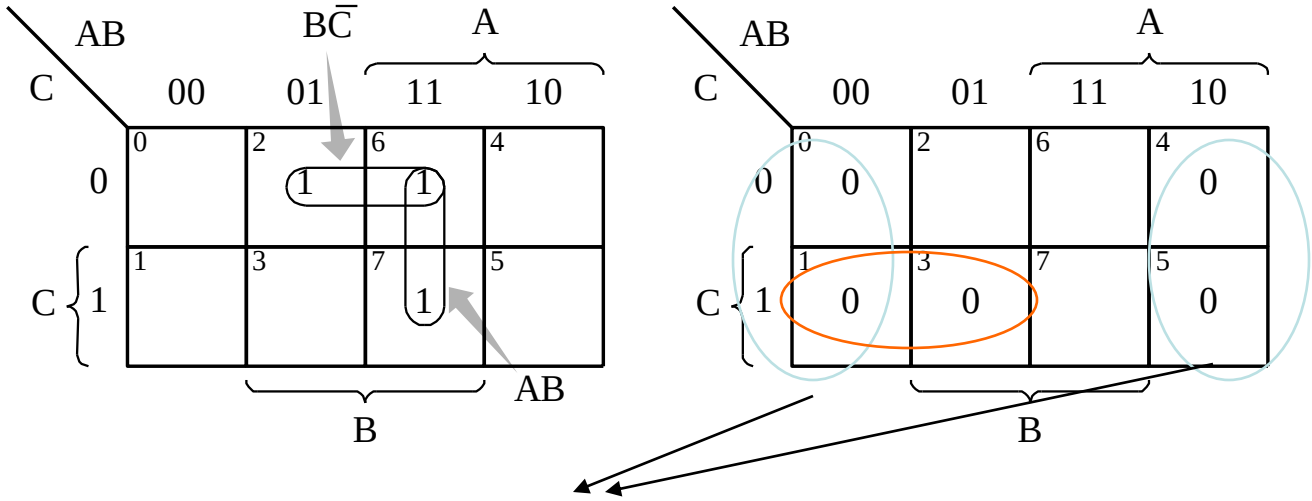
- $f(A,B,C) = \sum m(0,3,5) =$ A`B`C`+A`BC+AB`C

	A`B` 0 0	A`B 0 1	A B 1 1	A B` 1 0
C` 0	1 A`B`C`			
C 1		1 A`BC		1 AB`C

EXAMPLE

SIMPLIFY F= A`BC` + A B C` + A B C USING; (A) SUM OF MINTERMS. (B) MAXTERMS.

- Each cell of an *n*-variable K-map has *n* logically adjacent cells.



F`= B` + A`C  F = B(A+C`)

a- $f(A,B,C) = AB + BC'$ *b-* $f(A,B,C) = B(A + C')$

MAXTERM EXAMPLE

		(A+B) A`B`	(A+B`) A`B	(A`+B`) AB	(A`+B) AB`
C	C`		0	0	0
C`	C	0		0	

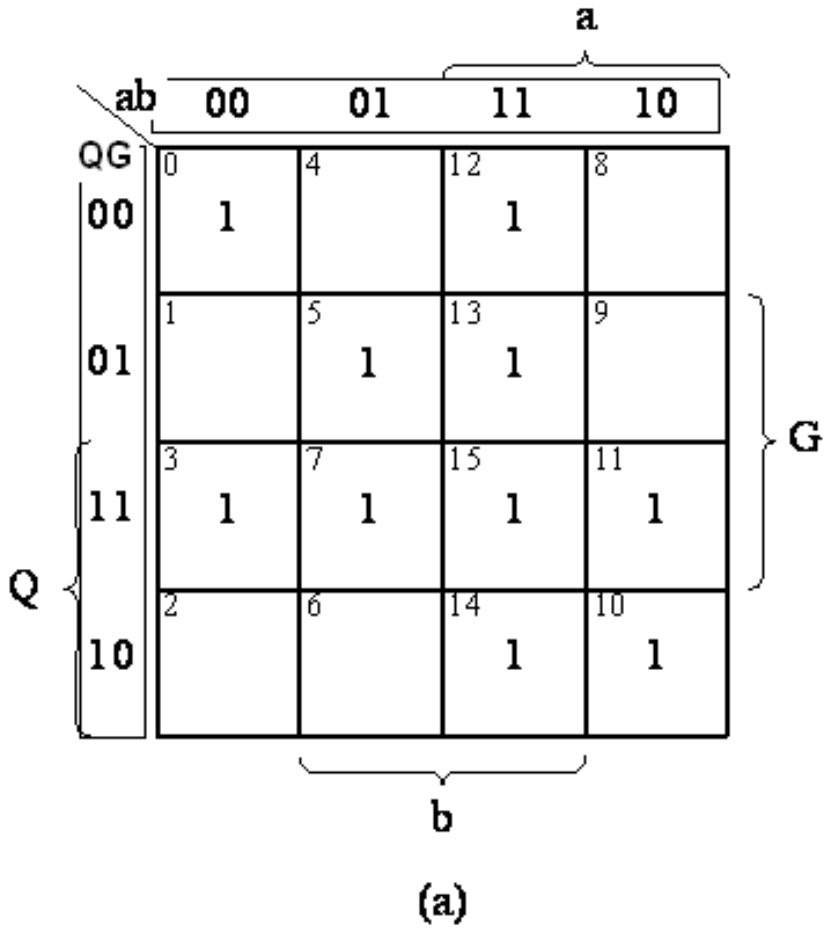
$f(A,B,C) = \prod M(1,2,4,6,7)$

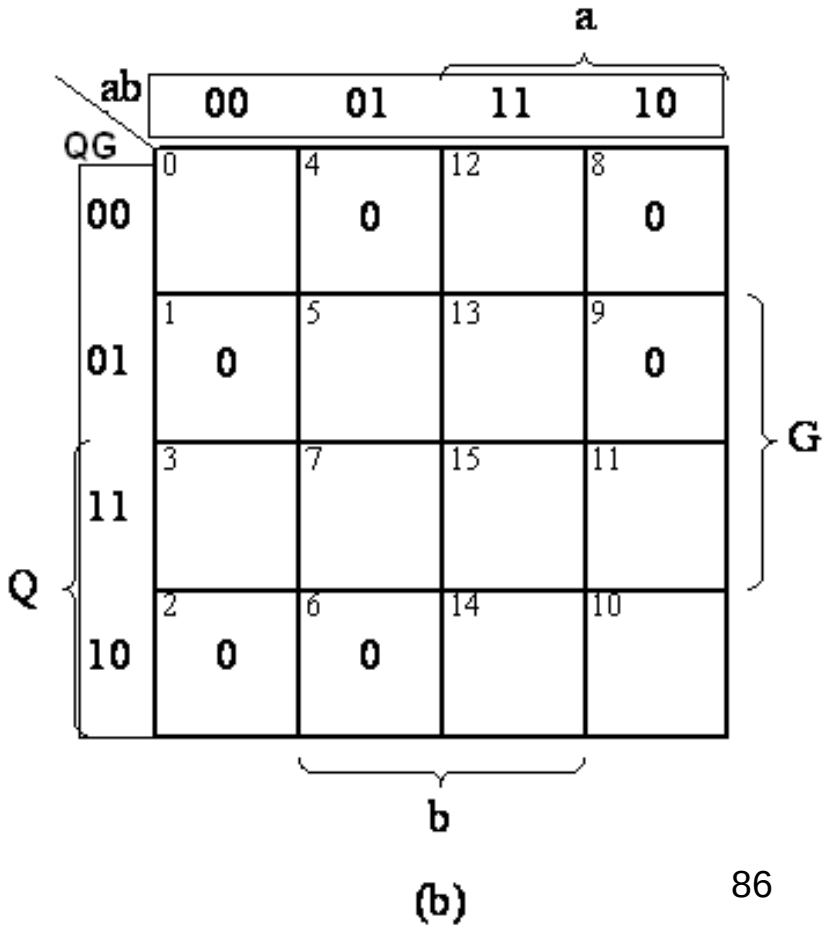
=(A+B+C`)(A+B`+C)(A`+B+C) )(A`+B`+C) (A`+B`+C`)

Note that the complements are (0,3,5) which are the minterms of the previous example

Four Variable Example
(a) Minterm Form. (b) Maxterm Form.

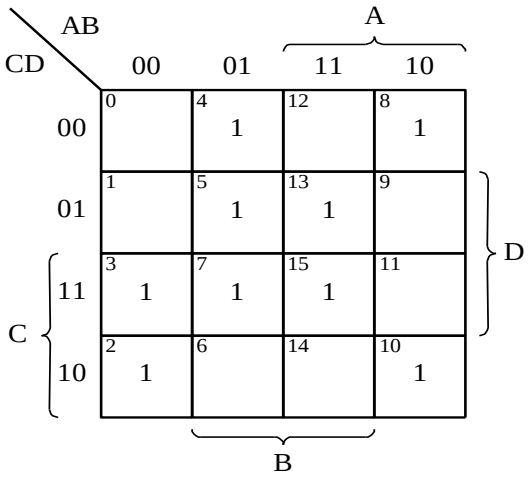
$$f(a,b,Q,G) = \sum m(0,3,5,7,10,11,12,13,14,15) = \prod M(1,2,4,6,8,9)$$



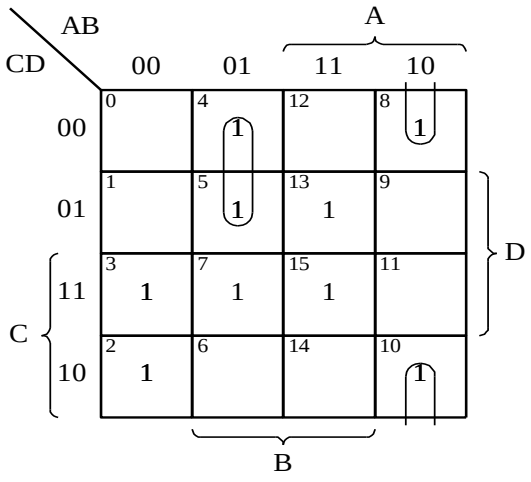


EXAMPLE SIMPLIFY

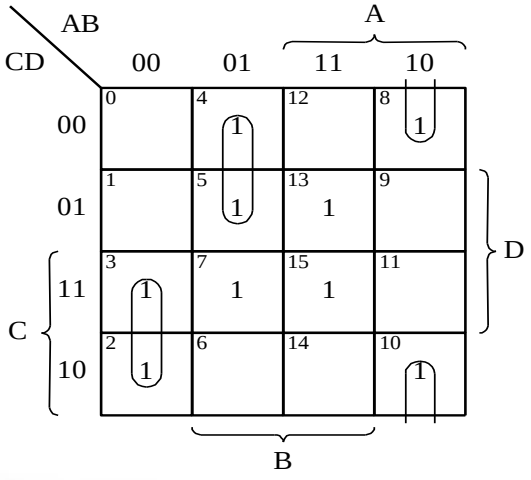
$$f(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15)$$



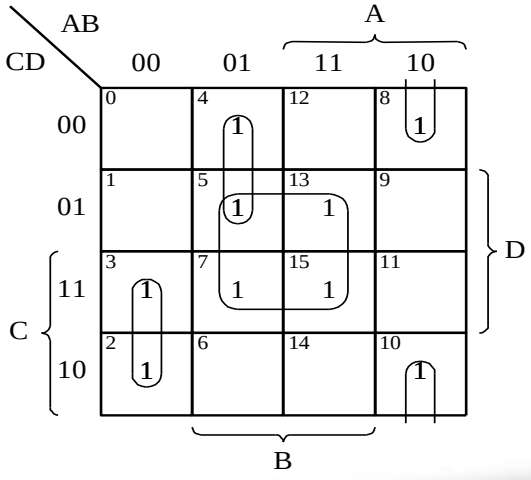
(a)



(b)



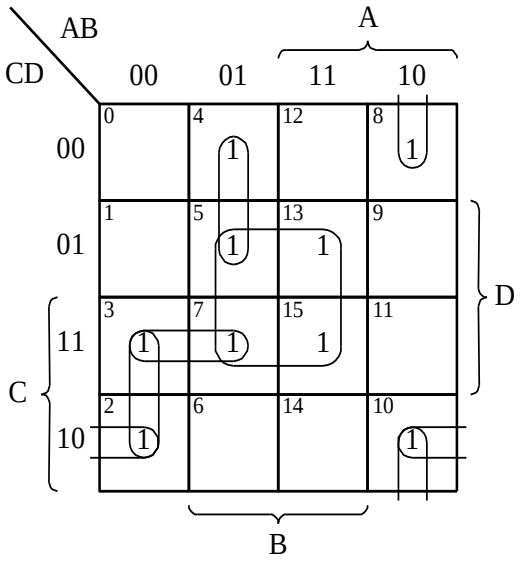
(c)



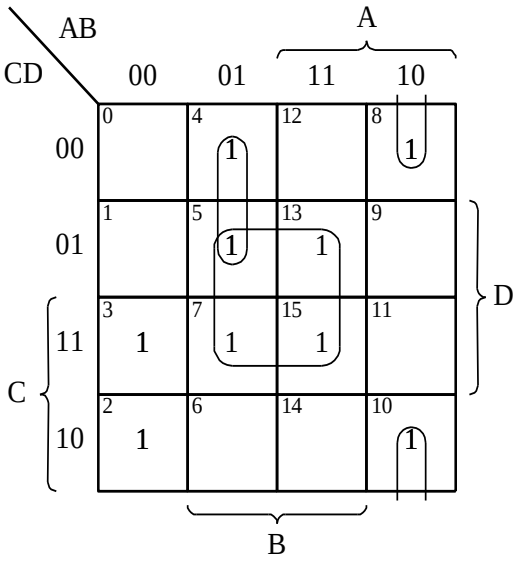
(d)

EXAMPLE MULTIPLE SELECTIONS

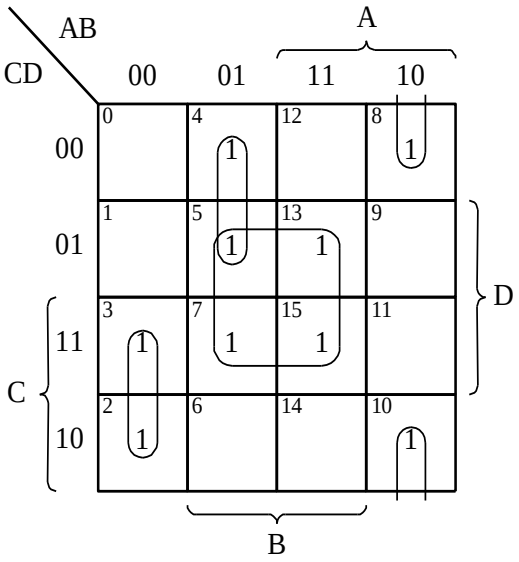
$$f(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15)$$



(a)



(b)

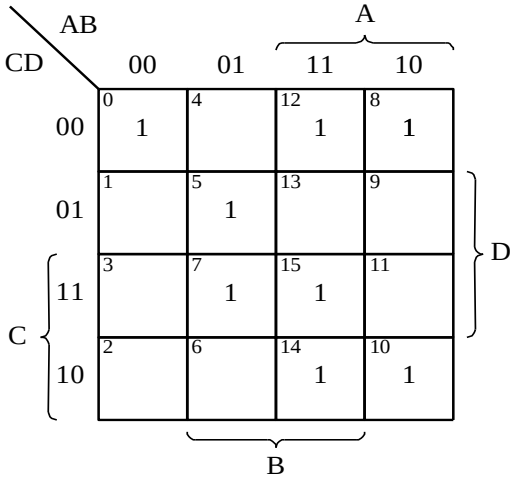


(c)

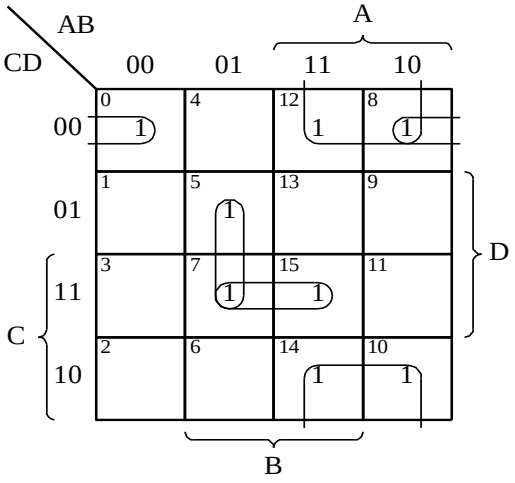
c produces less terms than a

EXAMPLE REDUNDANT SELECTIONS

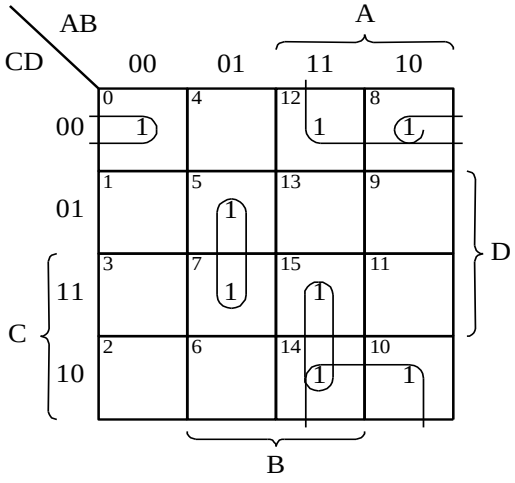
$f(A,B,C,D) = \sum m(0,5,7,8,10,12,14,15)$



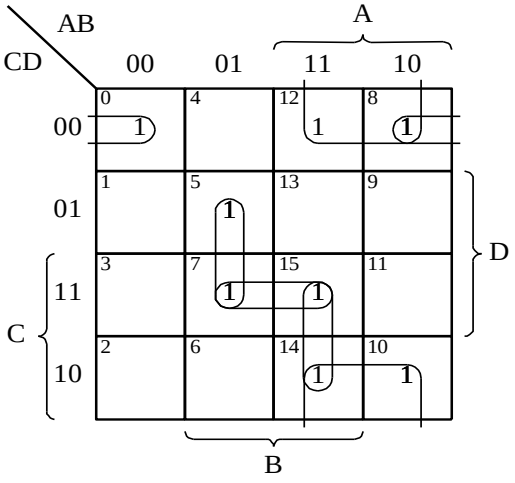
(a)



(b)

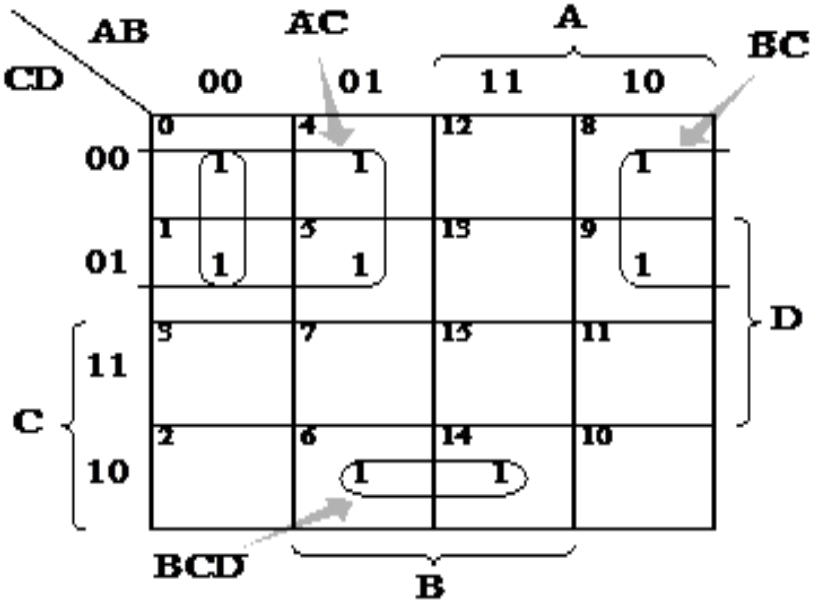


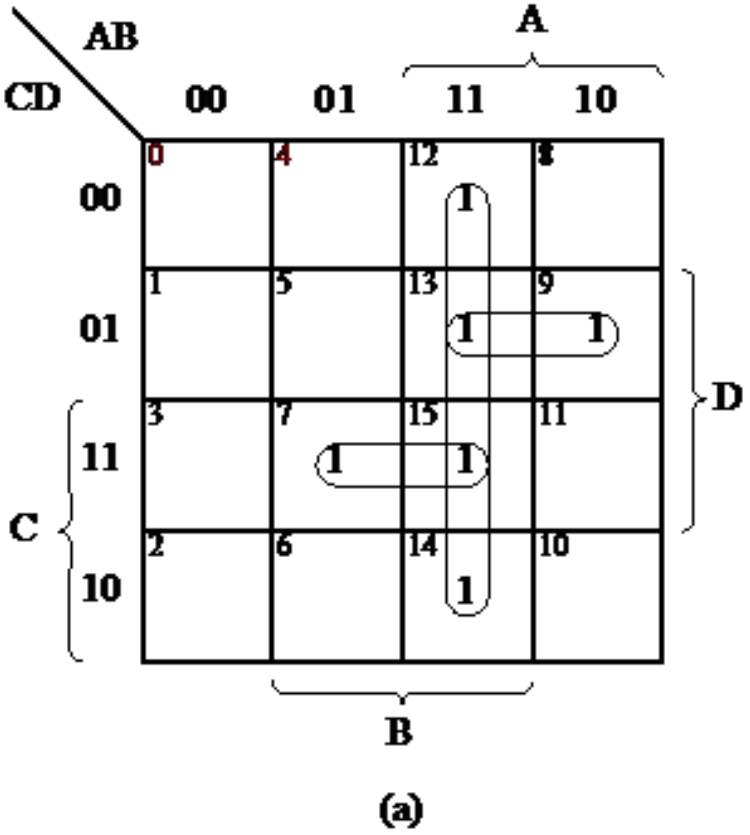
(c)



(d)

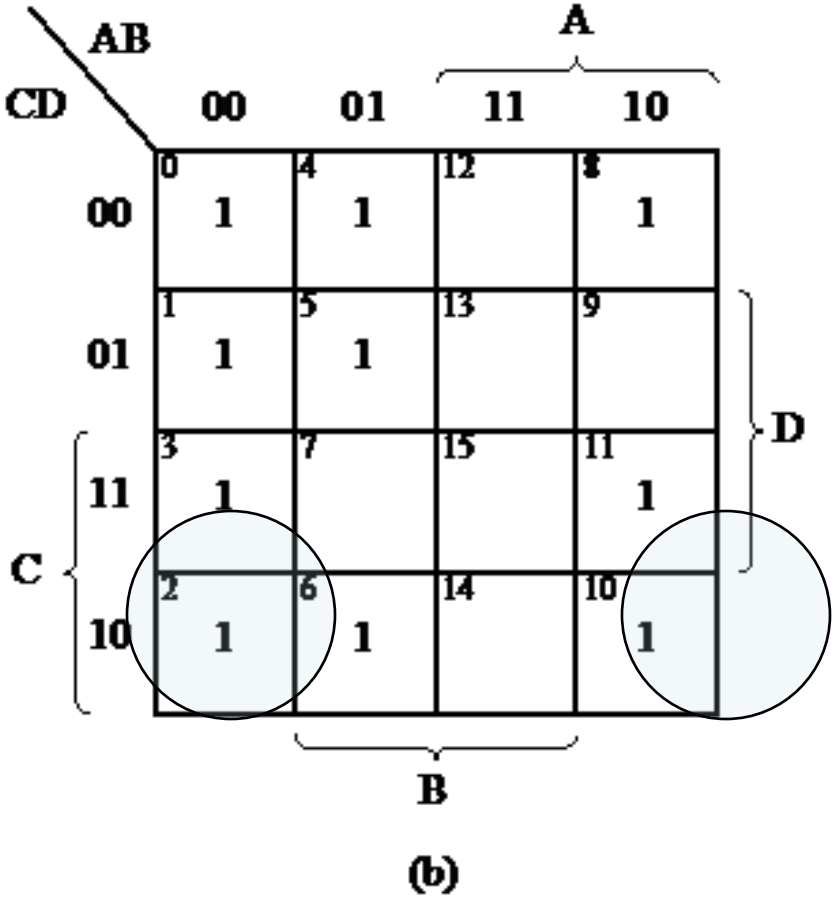
EXAMPLE



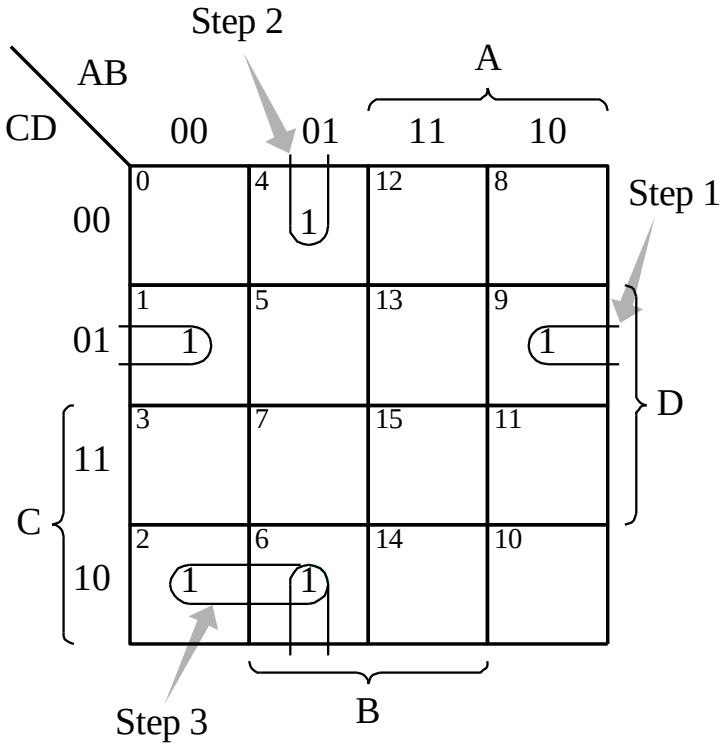


(a)

EXAMPLE



$f(A,B,C,D) = \sum m(1,2,4,6,9)$

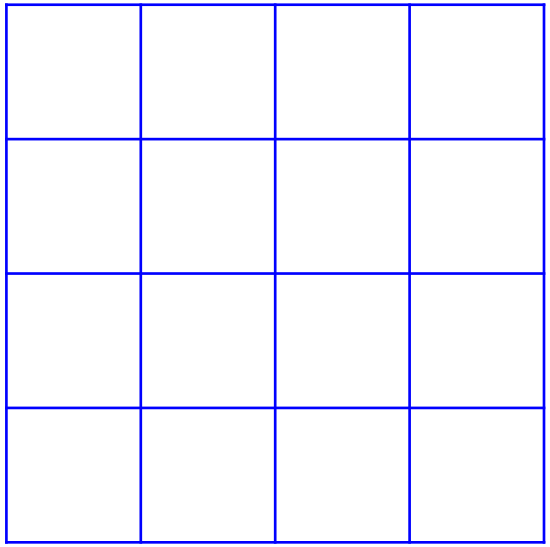


EXAMPLE : 4 VARIABLE K-MAP

Example:

After labeling and transferring the truth table data into the K-Map, write the simplified sum-of-products (SOP) logic expression for the logic function F_3.

R	S	T	U	F_3
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



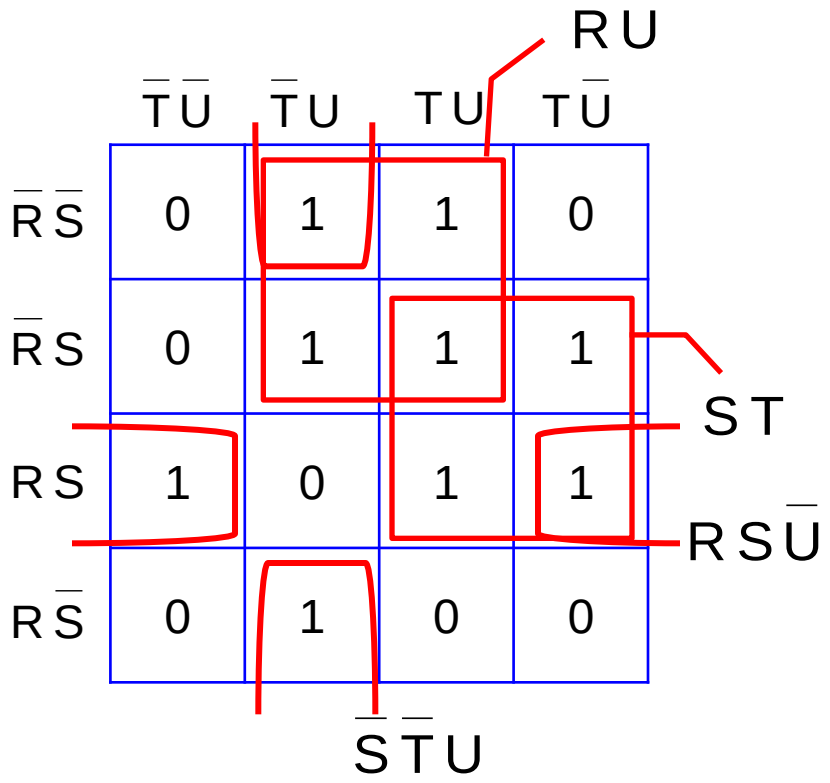
EXAMPLE : 4 VARIABLE K-MAP

Example:

After labeling and transferring the truth-table data into the K-Map, write the simplified sum-of-products (SOP) logic expression for the logic function F_3.

Solution:

R	S	T	U	F_3
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

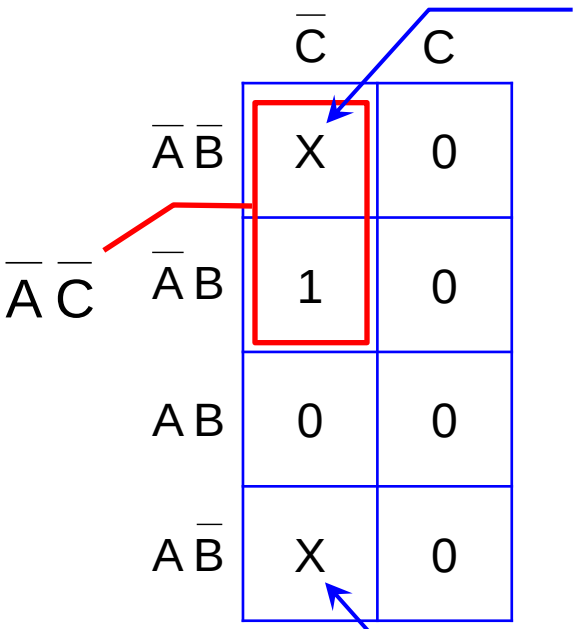


$$F_3 = RS\bar{U} + \bar{S}\bar{T}U + \bar{R}U + ST$$

DON'T CARE CONDITIONS

- A *don't care* condition, marked by (X) in the truth table, indicates a condition where the design doesn't care if the output is a (0) or a (1).
- A *don't care* condition can be treated as a (0) or a (1) in a K-Map.
- Treating a *don't care* as a (0) means that you do not need to group it.
- Treating a *don't care* as a (1) allows you to make a grouping larger, resulting in a simpler term in the SOP equation.

Some You Group, Some You Don't



This *don't care* condition was treated as a (1). This allowed the grouping of a single one to become a grouping of two, resulting in a simpler term.

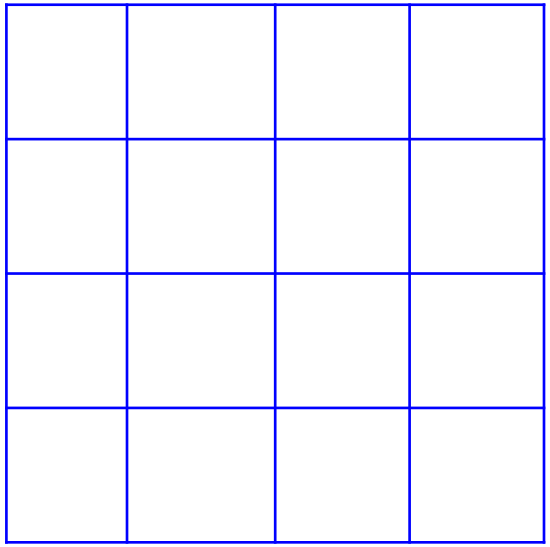
There was no advantage in treating this *don't care* condition as a (1), thus it was treated as a (0) and not grouped.

EXAMPLE : *DON'T CARE* CONDITIONS

Example:

After labeling and transferring the truth table data into the K-Map, write the simplified sum-of-products (SOP) logic expression for the logic function F_4. Be sure to take advantage of the *don't care* conditions.

R	S	T	U	F_4
0	0	0	0	X
0	0	0	1	0
0	0	1	0	1
0	0	1	1	X
0	1	0	0	0
0	1	0	1	X
0	1	1	0	X
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	X
1	1	0	0	X
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



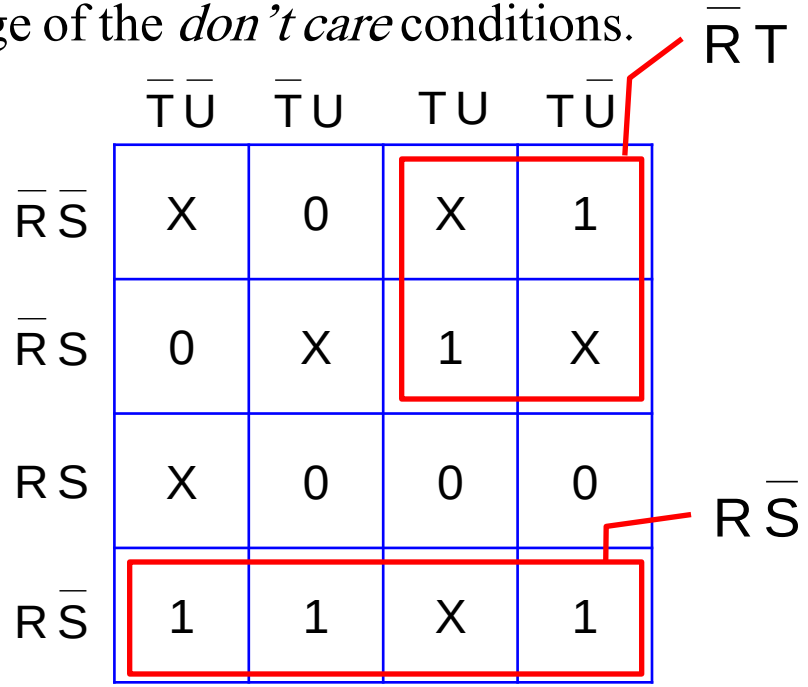
EXAMPLE : *DON'T CARE* CONDITIONS

Example:

After labeling and transferring the truth table data into the K-Map, write the simplified sum-of-products (SOP) logic expression for the logic function F_4. Be sure to take advantage of the *don't care* conditions.

Solution:

R	S	T	U	F_4
0	0	0	0	X
0	0	0	1	0
0	0	1	0	1
0	0	1	1	X
0	1	0	0	0
0	1	0	1	X
0	1	1	0	X
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	X
1	1	0	0	X
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



$$F_4 = \overline{R}\,T + R\,\overline{S}$$