Logic Circuit

Boolean Variables

- A Boolean Variable takes the value of either 0 or 1.
- In digital electronics, Boolean 0 and 1 correspond to binary 0 and 1.
- In logic, 0 and 1 are sometimes called FALSE and TRUE.
- We use symbols to represent Boolean variables.

  E.g.: A, B, C, X, Y, Z.

Logic Gate

- A logic gate is an elementary building block of a digital circuit .
- Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions *low* (0) or *high* (1), represented by different voltage levels.
- The logic state of a terminal can, and generally does, change often, as the circuit processes data.
- 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.
- There are seven basic logic gates: **AND, OR, XOR, NOT, NAND, NOR**, and **XNOR**.

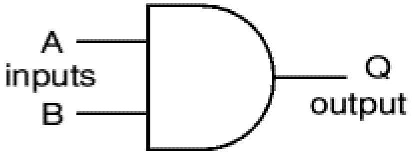
Truth Table

- A truth table is a good way to show the function of a logic gate. It shows the output states for every possible combination of input states. The symbols 0 (false) and 1 (true) are usually used in truth tables.

Logic Gate Symbols

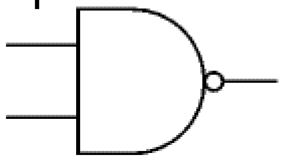
<u>Inputs and outputs</u>

- Gates have two or more inputs, except a NOT gate which has only one input. All gates have only one output. Usually the letters A, B, C and so on are used to label inputs, and Q is used to label the output. On this page the inputs are shown on the left and the output on the right.



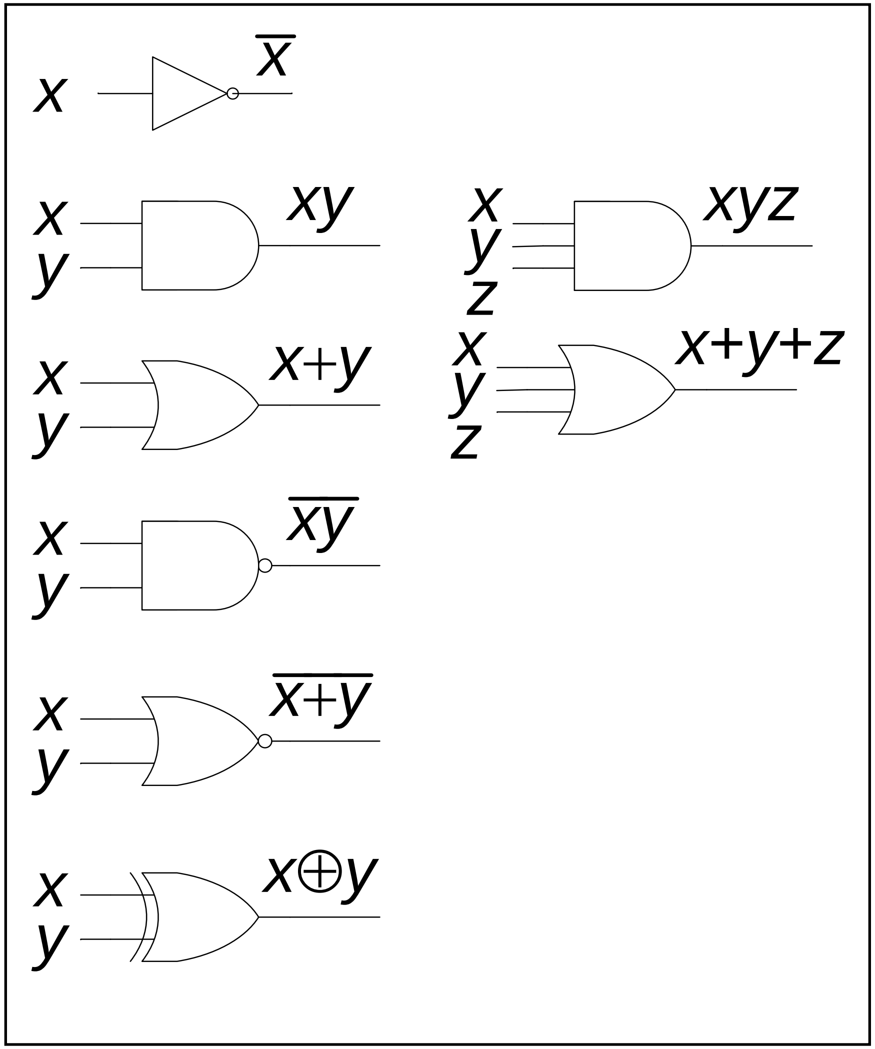
<u>The inverting circle (o)</u>

- Some gate symbols have a circle on their output which means that their function includes **inverting** of the output. It is equivalent to feeding the output through a NOT gate. For example the NAND (<u>N</u>ot <u>AND</u>) gate symbol shown on the right is the same as an AND gate symbol but with the addition of an inverting circle on the output.



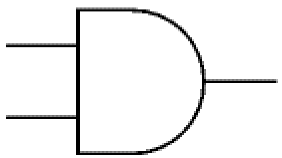
Basic logic gates

- Not
- And
- Or
- Nand
- Nor
- Xor



AND

- The AND gate requires two inputs and has one output.
- The AND gate only produces an output of 1 when BOTH the inputs are a 1, otherwise the output is 0.

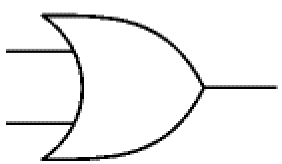


AND

X	Y	Z = X . Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

- The OR gate has two input lines and one output line. Basically, if either or both of the inputs are a 1, the resulting output value is a 1. Note in the truth table, the only time the output is 0 is when both [illegible]

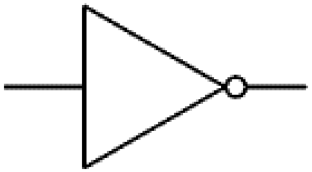


OR

X	Y	Z = X + Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

- The NOT gate is also known as an inverter, simply because it changes the input to its opposite (inverts it).
- The NOT gate accepts only one input and the output is the opposite of the input. In other words, a low-voltage input (0) is converted to a high-voltage output (1). It's that simple!
- A common way of using the NOT gate is to simply attach the circle to the front of another gate. This simplifies the circuit drawing and simply says: "Invert the output from this gate."



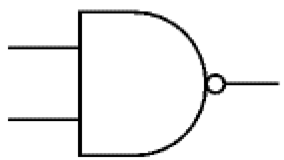
NOT

X	$Z = \overline{X}$
0	1
1	0

- The NAND and NOR gates possess a special property: they are ***universal***. That is, given enough gates, either type of gate is able to mimic the operation of *any* other gate type. Combinations of them can be used to accomplish any of the basic operations and can thus produce an inverter, an OR gate or an AND gate.

NAND

- This is an AND gate with the output inverted, as shown by the 'o' on the output.
- The output is true if input A AND input B are NOT both true: **Q = NOT (A AND B)**
- A NAND gate can have two or more inpu s true if NOT all inputs are true.

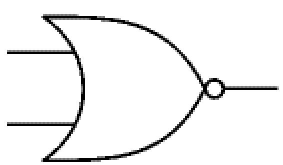


NAND

X	Y	Z = X . Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

- This is an OR gate with the output inverted, as shown by the 'o' on the output.
  The output Q is true if NOT inputs A OR B are true: **Q = NOT (A OR B)**
  A NOR gate can have two or more input true if no inputs are true.

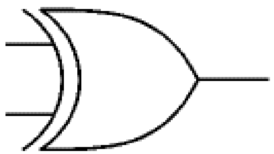


NOR

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

XOR

- The output Q is true if either input A is true OR input B is true, **but not when both of them are true**: **Q = (A AND NOT B) OR (B AND NOT A)**
- This is like an OR gate but excluding both inputs being true.
- The output is true if inputs A and B are **DIFFERENT**. EX-OR gates can only have 2 inputs.

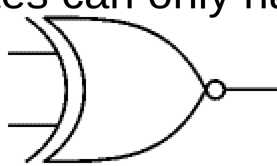


XOR

X	Y	Q
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

- This is an EX-OR gate with the output inverted, as shown by the 'o' on the output.
- The output Q is true if inputs A and B are the **SAME** (both true or both false): **Q = (A AND B) OR (NOT A AND NOT**
- EX-NOR gates can only have 2 inputs.



XNOR

X	Y	Q
0	0	1
0	1	0
1	0	0
1	1	1

Basic Boolean Identities

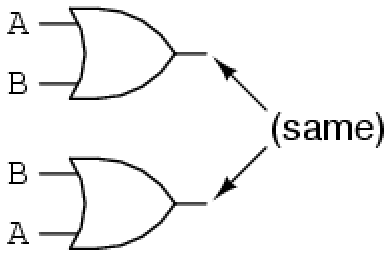
- As with algebra, there will be Boolean operations that we will want to simplify
  - We apply the following Boolean identities to help
    - For instance, in algebra, x = y * (z + 0) + (z * 0) can be simplified to just x = y * z

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0+x = x$
Null (or Dominance) Law	$0x = 0$	$1+x = 1$
Idempotent Law	$xx = x$	$x+x = x$
Inverse Law	$x\bar{x} = 0$	$x+\bar{x} = 1$
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$
Absorption Law	$x(x+y) = x$	$x+xy = x$
DeMorgan's Law	$(\overline{xy}) = \bar{x}+\bar{y}$	$(\overline{x+y}) = \bar{x}\bar{y}$
Double Complement Law	$\bar{\bar{x}} = x$	

Commutative

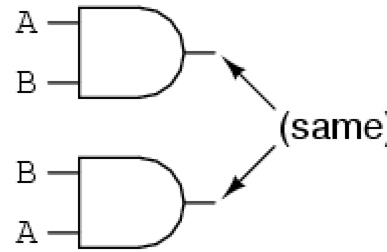
Commutative property of addition

A + B = B + A



Commutative property of multiplication

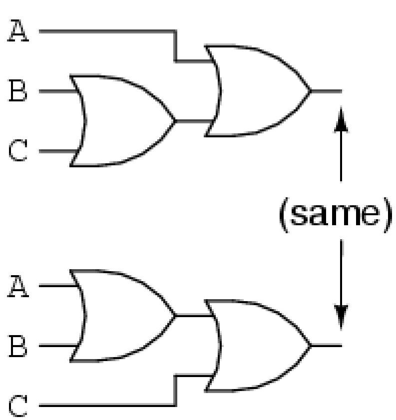
AB = BA



Associative

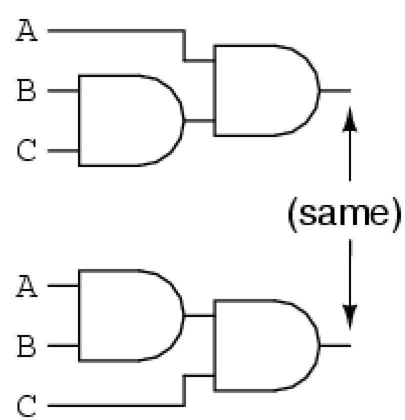
Associative property of addition

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



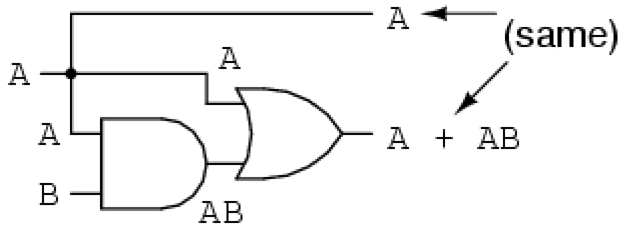
Associative property of multiplication

A(BC) = (AB)C



Simplification

A + AB = A



A + AB

↓ Factoring **A** out of both terms

A(1 + B)

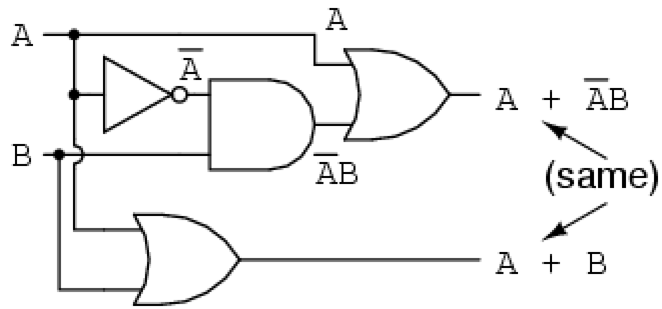
↓ Applying identity **A + 1 = 1**

A(1)

↓ Applying identity **1A = A**

A

$$A + \overline{A}B = A + B$$



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

Applying the previous rule to expand **A** term $\mathbf{A + AB = A}$

$$A + AB + \overline{A}B$$

Factoring **B** out of 2nd and 3rd terms

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

Applying identity $\mathbf{A + \overline{A} = 1}$

$$A + B(1)$$

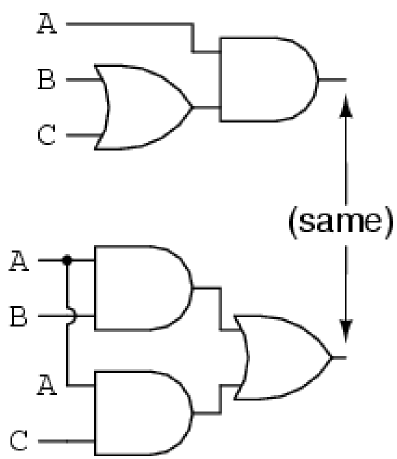
Applying identity $\mathbf{1A = A}$

$$A + B$$

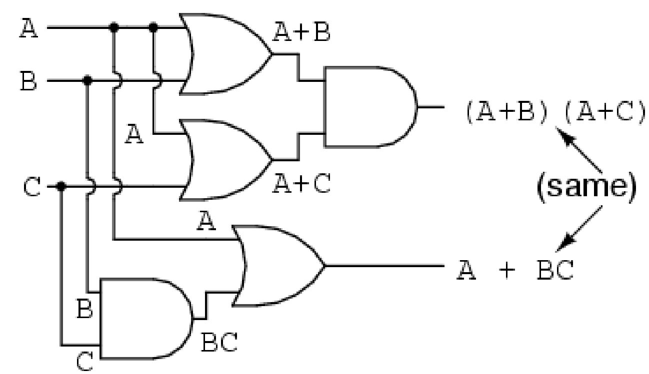
Distributive

Distributive property

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



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



$(A + B)(A + C)$

↓ Distributing terms

$AA + AC + AB + BC$

↓ Applying identity $\mathbf{AA = A}$

$A + AC + AB + BC$

↓ Applying rule $\mathbf{A + AB = A}$ to the $A + AC$ term

$A + AB + BC$

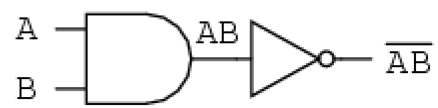
↓ Applying rule $\mathbf{A + AB = A}$ to the $A + AB$ term

$A + BC$

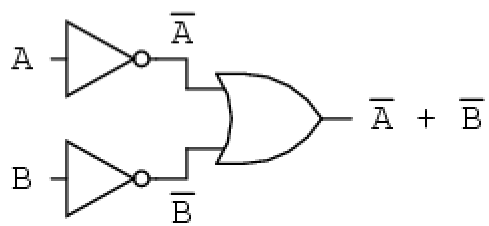
DeMorgan's

DeMorgan's Theorems

NAND to *Negative-OR* *NOR* to *Negative-AND*



. . . is equivalent to . . .



$$\overline{\mathbf{AB}} = \overline{\mathbf{A}} + \overline{\mathbf{B}}$$

Practice:

1. Demonstrate by means of truth tables the validities of the following identities:

   a. $(XYZ)' = X' + Y' + Z'$

   b. $X + YZ = (X + Y)(X + Z)$

   c. $X'Y + Y'Z + X'Z = XY' + YZ' + X'Z$

2. Prove the following identity of each of the following Boolean equations using algebraic manipulation:

   a. $A'B + B'C' + AB + B'C = 1$

   b. $Y + X'Z + XY' = X + Y + Z$

   c. $AB + BC'D' + A'BC + C'D = B + C'D$

Karnaugh Map

Minterms – Sum of Product (SOP)

- A product term in which all the variables appear exactly once (complemented or uncomplemented)
- The combination has the value 1
- E.g.: the four minterms for two variables, X and Y are X’Y’, X’Y, XY’ and XY.

- A Boolean function can be represented algebraically from a given truth table by forming the logical sum of all the minterms that produce 1 in the function.

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Expressed algebraically:

$$F = X'Y'Z' + X'YZ' + XY'Z + XYZ = m_0 + m_2 + m_5 + m_7$$

- In abbreviated form:

$$F(X, Y, Z) = \Sigma m(0, 2, 5, 7)$$

Maxterms – Product of Sum (POS)

- A sum term that contains all the variables in complemented or uncomplemented form
- The combination has the value of 0
- E.g.: the four minterms for two variables, X and Y are X’ + Y’, X’ + Y, X + Y’ and X + Y.

- A boolean function can be represented algebraically from a given truth table by forming the logical sum of all the maxterms that produce 0 in the function.

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Expressed algebraically:

$$F = (X + Y + Z')(X + Y' + Z')(X' + Y + Z)(X' + Y' + Z) = m_1 + m_3 + m_4 + m_6$$

- Abbreviated form:

$$F(X, Y, Z) = \prod M(1, 3, 4, 6)$$

Karnaugh Maps (K-maps)

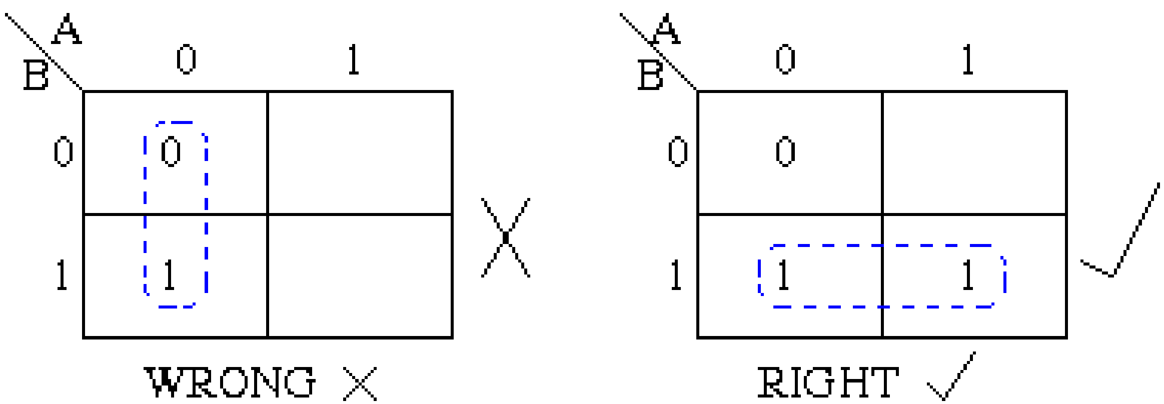
- A K-map is a collection of squares
  - Each square represents a minterm
  - The collection of squares is a graphical representation of a Boolean function
  - Adjacent squares differ in the value of one variable
  - Alternative algebraic expressions for the same function are derived by recognizing patterns 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

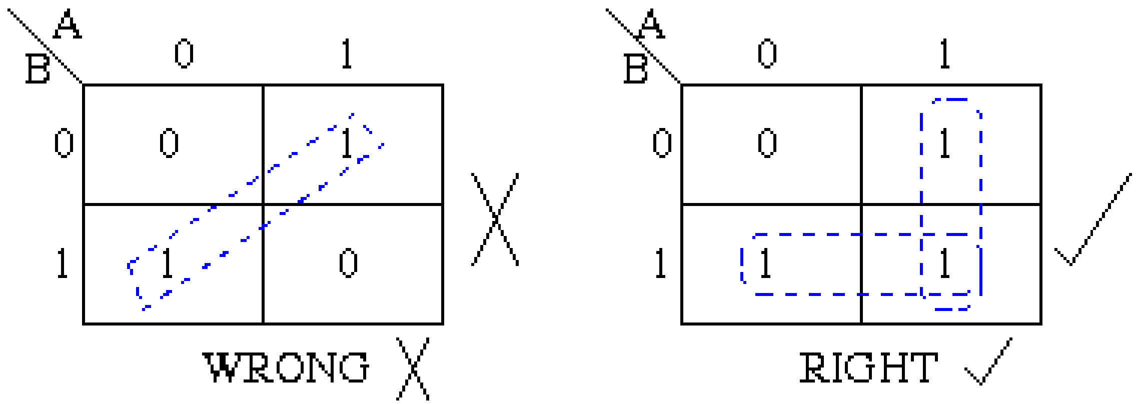
Karnaugh Maps - Rules of Simplification

- The Karnaugh map uses the following rules for the simplification of expressions by *grouping* together adjacent cells containing *ones*

1. Groups may not include any cell containing a zero

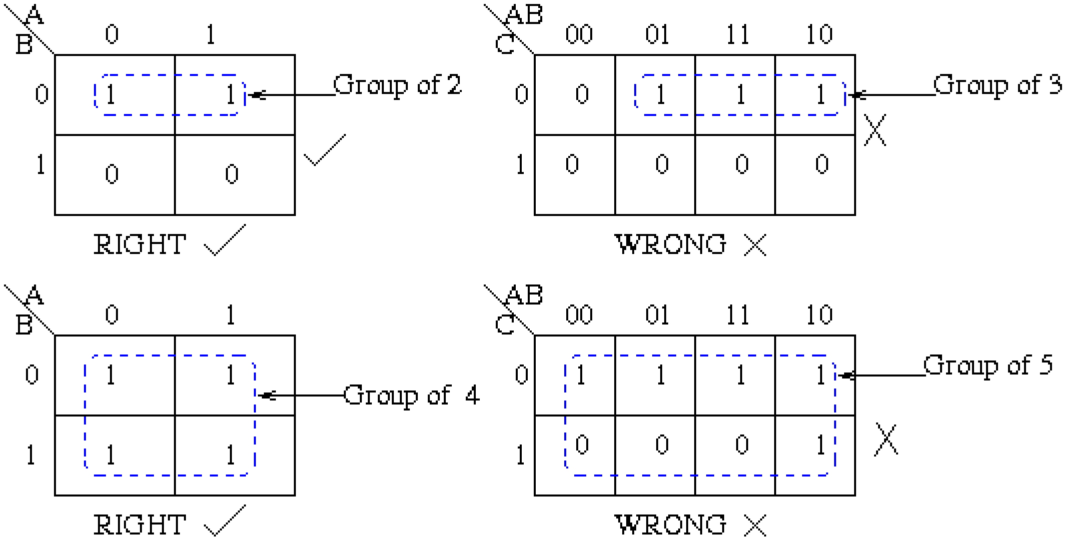


2. Groups may be horizontal or vertical, but not diagonal.

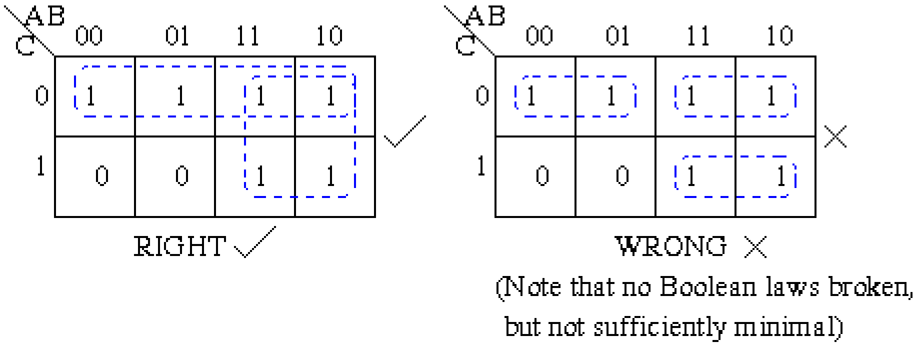


- Groups must contain 1, 2, 4, 8, or in general 2^n cells. That is if $n = 1$, a group will contain two 1's since $2^1 = 2$.

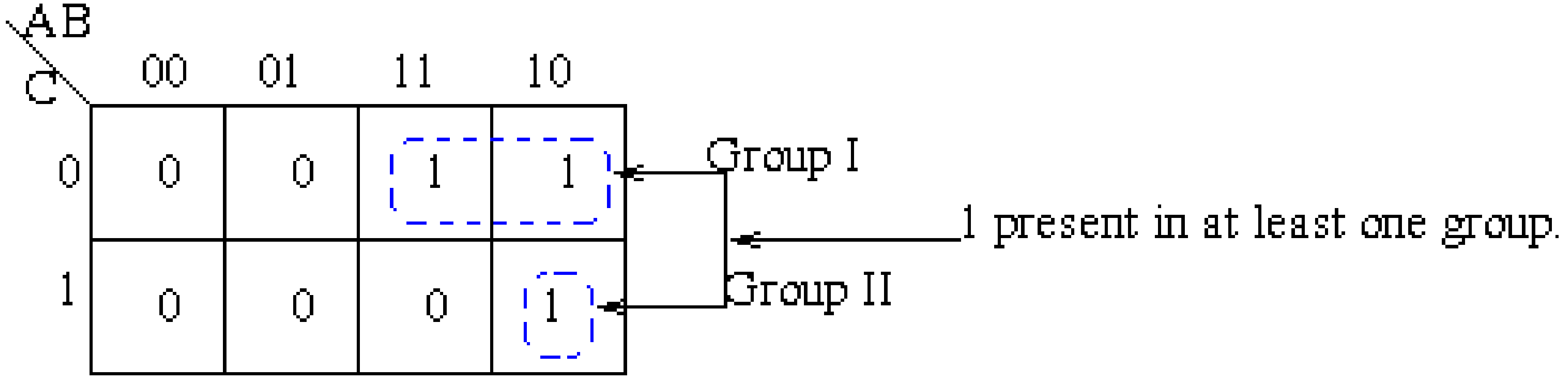
3. Grouping of 1 should be in 2^{n}. i.e. grouping should be in 1, 2's, 4's, 8's and so forth.



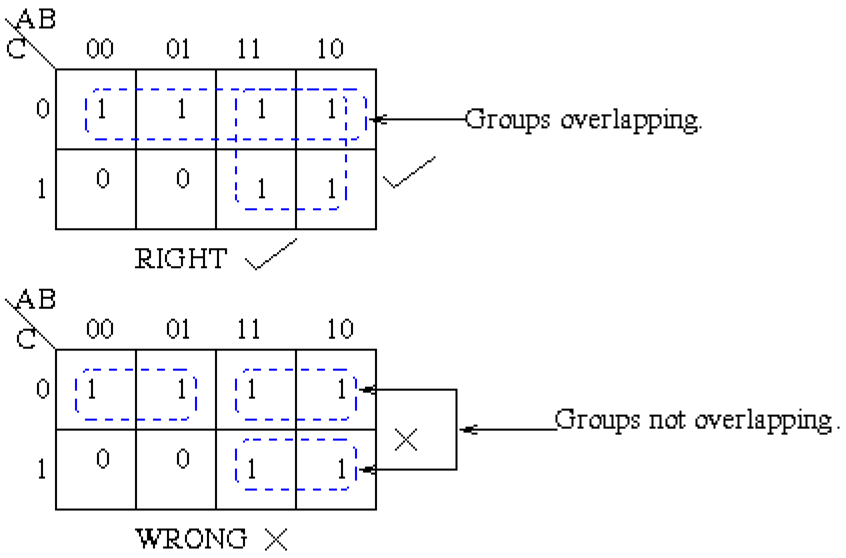
4. Each group should be as large as possible.



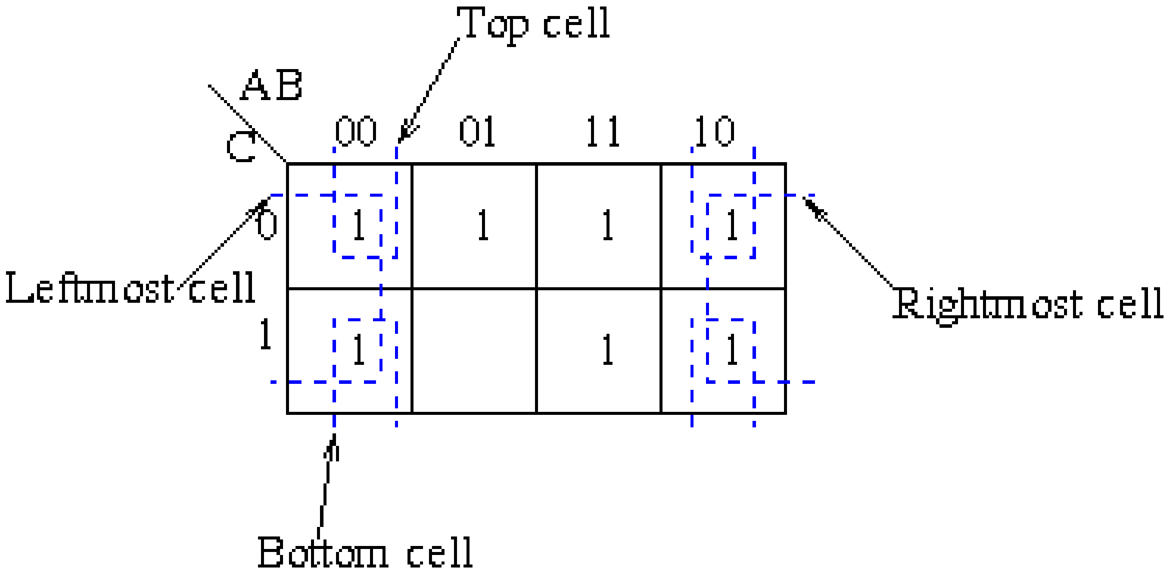
5. Each cell containing a *one* must be in at least one group.



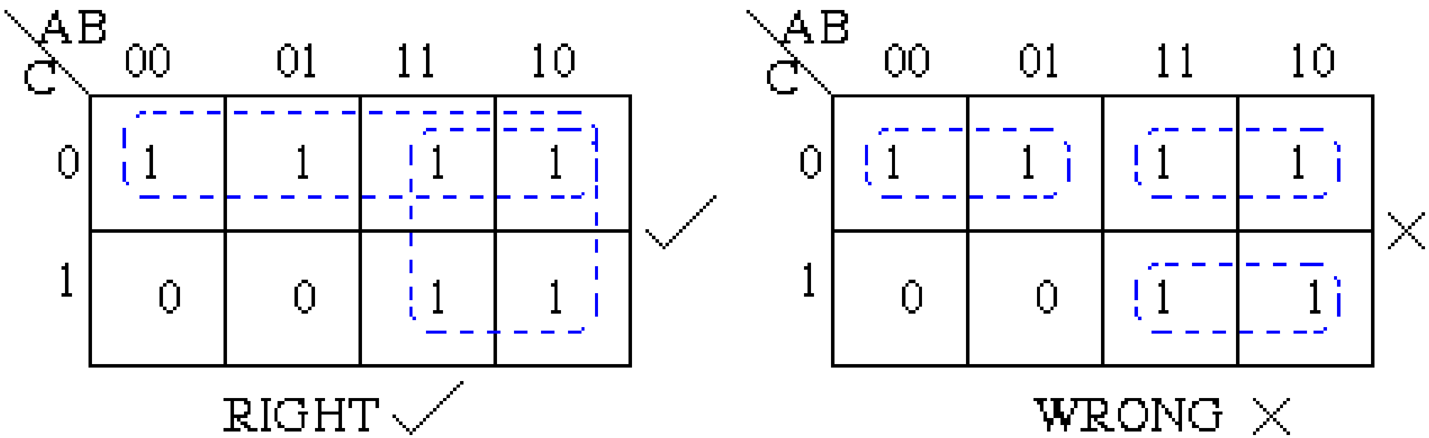
6. Groups may overlap



7. Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



8. There should be as few groups as possible, as long as this does not contradict any of the previous rules.



Re-arranging the truth table

- 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

		y = 0	y = 1
x	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.

		y = 0	y = 1
x	0	x'y'	x'y
	1	xy'	xy

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

- Imagine a two-variable sum of minterms:

$$x'y' + x'y$$

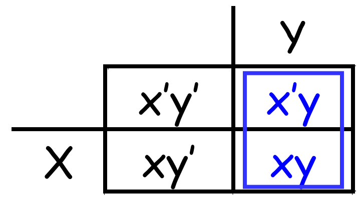
- Both of these minterms appear in the top row of a Karnaugh map, which means that they both contain the literal x'.

		y
	$x'y'$	$x'y$
X	xy'	xy

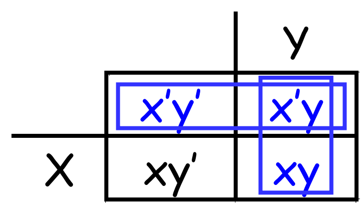
- What happens if you simplify this expression using Boolean algebra?

$$\begin{aligned} x'y' + x'y &= x'(y' + y) && [\text{ Distributive }] \\ &= x' \bullet 1 && [\ y + y' = 1\ ] \\ &= x' && [\ x \bullet 1 = x\ ] \end{aligned}$$

- 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.



Three-variable Karnaugh map

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

		yz			
		00	01	11	10
x	0	x'y'z'	x'y'z	x'yz	x'yz'
	1	xy'z'	xy'z	xyz	xyz'

		yz			
		00	01	11	10
x	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):

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

			y	
	m_0	m_1	m_3	m_2
x	m_4	m_5	m_7	m_6
		z		

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

			y	
	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
x	$xy'z'$	$xy'z$	xyz	xyz'
		z		

$$
\begin{aligned}
& x'y'z + x'yz \\
= \quad & x'z(y' + y) \\
= \quad & x'z \bullet 1 \\
= \quad & x'z
\end{aligned}
$$

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

			y	
	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
x	$xy'z'$	$xy'z$	xyz	xyz'
		z		

$$
\begin{aligned}
& x'y'z' + xy'z' + x'yz' + xyz' \\
= \quad & z'(x'y' + xy' + x'y + xy) \\
= \quad & z'(y'(x' + x) + y(x' + x)) \\
= \quad & z'(y'+y) \\
= \quad & z'
\end{aligned}
$$

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

- Three variables maps exhibit the following characteristics:
  - One square represents a minterm of three literals
  - A rectangle of two squares represents a product term of two literals (or variables)
  - A rectangle of four squares represents a product term of one literal (or variable)
  - A rectangle of eight squares encompasses the entire map and produces a function that is always equal to logic 1.

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

$$
\begin{aligned}
f(x,y,z) &= x'y'z + xy'z + xyz' + xyz \\
&= m_1 + m_5 + m_6 + m_7
\end{aligned}
$$

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."

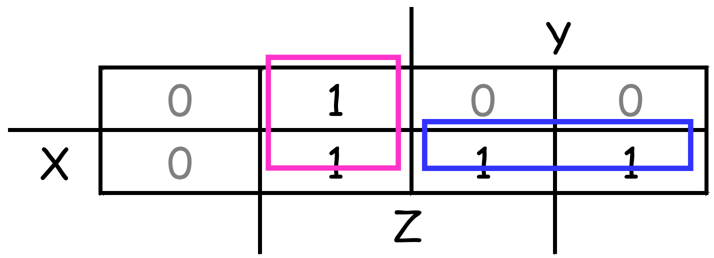
				y	y
	m_0	m_1	m_3	m_2
X	m_4	m_5	m_7	m_6
		Z	Z	

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	y
	0	1	0	0
X	0	1	1	1
		Z	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.

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.

			y	y	
	w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'	
	w'xy'z'	w'xy'z	w'xyz	w'xyz'	x
w	wxy'z'	wxy'z	wxyz	wxyz'	x
w	wx'y'z'	wx'y'z	wx'yz	wx'yz'	
		z	z		

			y	y	
	m_0	m_1	m_3	m_2	
	m_4	m_5	m_7	m_6	x
w	m_{12}	m_{13}	m_{15}	m_{14}	x
w	m_8	m_9	m_{11}	m_{10}	
		z	z		

- 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.

- Four variables maps exhibit the following characteristics:
  - A rectangle of 2 squares represents a product term of three literals (or variables)
  - A rectangle of 4 squares represents a product term of two literals (or variables)
  - A rectangle of 8 squares represents a product term of one literal (or variable)
  - A rectangle of 4 squares produces a function that is always equal to logic 1

Don’t Care Conditions

- In a Kmap, a don’t care condition is identified by an *X* in the cell of the minterm(s) for the don’t care inputs, as shown below.
- In performing the simplification, we are free to include or ignore the *X*’s when creating our groups.

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

Don’t Care Conditions

- In one grouping in the Kmap below, we have the function:

$$F(W,X,Y,Z) = \overline{W}\,\overline{X} + YZ$$

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

Don’t Care Conditions

- A different grouping gives us the function:

$$F(W,X,Y,Z) = \bar{W}Z + YZ$$

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

Don’t Care Conditions

- The truth table of:

$$F(W,X,Y,Z) = \bar{W}\bar{X} + YZ$$

is different from the truth table of:

$$F(W,X,Y,Z) = \bar{W}Z + YZ$$

- However, the values for which they differ, are the inputs for which we have don’t care conditions.

WX \ YZ	00	01	11	10
00	×	1	1	×
01		×	1	
11	×		1	
10			1	

WX \ YZ	00	01	11	10
00	×	1	1	×
01		×	1	
11	×		1	
10			1	