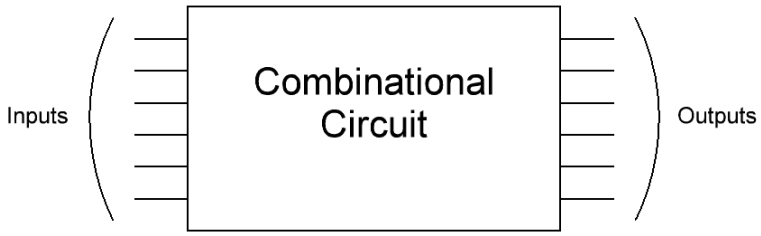
Chapter 2
Combinational Logic

Latest update: 26/09/2016

Table of Contents

I. Introduction

Combinational logic is the discipline used to design combinational circuits. A combinational circuit is a digital circuit whose outputs are functions of its present inputs only.



An input or an output can be either 0 or 1.

A combinational circuit can be defined by a truth table, which gives the values of its outputs according to the values of its inputs.

A combinational circuit is made up of logic gates only.

II. Logic Gates

Logic gates are the basic building blocks of combinational circuits.

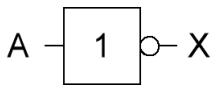
There are two symbols for each gate: rectangular-shape and distinctive-shape symbols. In this lesson, **we will use the distinctive-shape symbols**. Nevertheless, you should know the rectangular-shape symbols because they are widely used in a great number of books and datasheets.

1. The NOT Gate

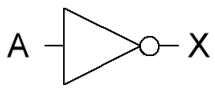
$\mathbf{X = \overline{A}}$

A	X
0	1
1	0

Truth Table



Rectangular-Shape Symbol



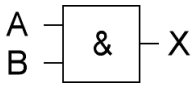
Distinctive-Shape Symbol

2. The AND Gate

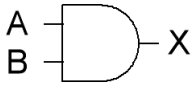
X = A.B

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table



Rectangular-Shape Symbol



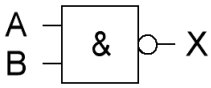
Distinctive-Shape Symbol

3. The NAND Gate

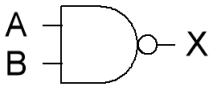
X = $\overline{\text{A.B}}$

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table



Rectangular-Shape Symbol



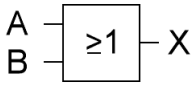
Distinctive-Shape Symbol

4. The OR Gate

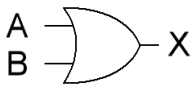
X = A + B

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table



Rectangular-Shape Symbol



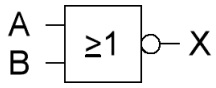
Distinctive-Shape Symbol

5. The NOR Gate

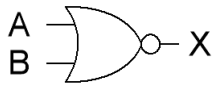
$$X = \overline{A + B}$$

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table



Rectangular-Shape Symbol



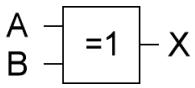
Distinctive-Shape Symbol

6. The EXCLUSIVE OR Gate (XOR)

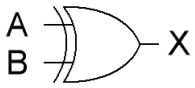
$$X = A \oplus B$$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table



Rectangular-Shape Symbol



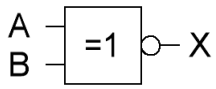
Distinctive-Shape Symbol

7. The EXCLUSIVE NOR Gate (XNOR)

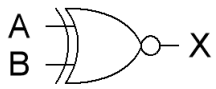
$$X = \overline{A \oplus B}$$

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table



Rectangular-Shape Symbol

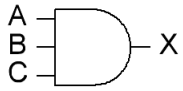


Distinctive-Shape Symbol

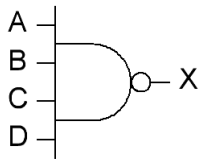
8. Multi-Input Gates

Logic gates can have more than two inputs.

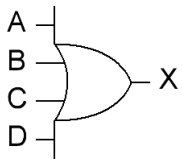
Examples:



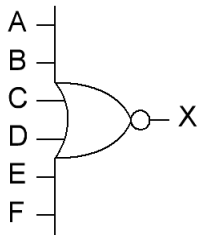
$X = A.B.C$



$X = \overline{A.B.C.D}$



$X = A + B + C + D$

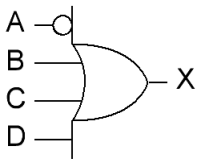


$X = \overline{A + B + C + D + E + F}$

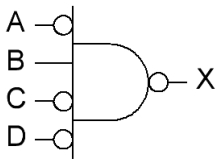
9. Bubbles and Gates

A circle on the symbol of a gate is known as an ‘inversion bubble’ or ‘bubble’. This bubble indicates a logical inversion and can be found either on inputs or outputs. Any combination is possible.

Examples:



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



$X = \overline{\overline{A}.B.\overline{C}.\overline{D}}$

10. Positive and Negative Gates

In positive logic we associate the value 0 with either the ‘inactive’ state or the ‘false’ state and the value 1 with either the ‘active’ state or the ‘true’ state.

For instance, we can say that:

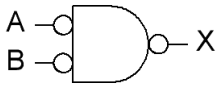
- The output of an OR gate is true (1) when at least one of its inputs is true (1).
- The output of an AND gate is true (1) only when all of its inputs are true (1).

Or:

- The output of an OR gate is active (1) when at least one of its inputs is active (1).
- The output of an AND gate is active (1) only when all of its inputs are active (1).

In negative logic we associate the value 0 with either the ‘active’ state or the ‘true’ state and the value 1 with either the ‘inactive’ state or the ‘false’ state.

For instance, have a look at the gate below:



$$X = \overline{\overline{A} . \overline{B}}$$

And let us draw its truth table:

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Obviously, we can recognize the truth table of an OR gate. Therefore, we can conclude that there is no difference between an OR gate and the gate above. Although their symbols are different, these two gates are physically the same:

OR gate Negative-AND gate

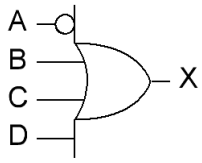
One symbol is called OR gate (or positive-OR gate) because in positive logic its output is true (1) when at least one of its inputs is true (1).

The other one is called negative-AND gate because in negative logic its output is true (0) when all of its inputs are true (0).

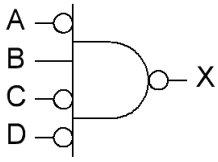
Actually, we usually mix positive and negative logic in the same circuit:

- When inputs or outputs have no bubble, they are active high. In other words, the value 0 represents a ‘false’ state and the value 1 represents a ‘true’ state.
- When inputs or outputs have bubbles, they are active low. In other words, the value 0 represents a ‘true’ state and the value 1 represents a ‘false’ state.

For example:



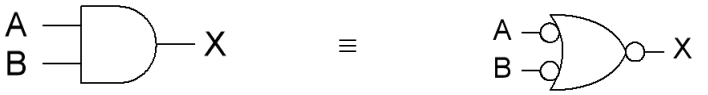
X is true (1) when *A* is true (0) or *B* is true (1) or *C* is true (1) or *D* is true (1).



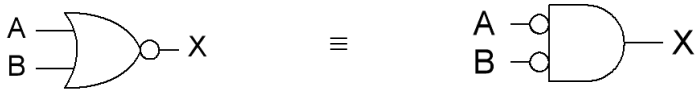
X is true (0) when *A* is true (0) and *B* is true (1) and *C* is true (0) and *D* is true (0).

In the same way, we can design other equivalent negative gates:

NOT gate Negative-NOT gate



AND gate Negative-OR gate



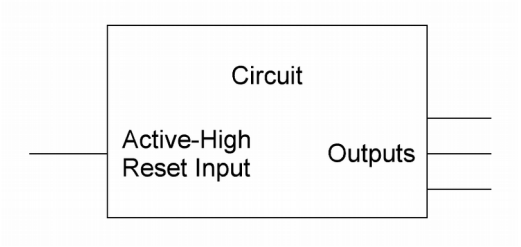
NOR gate Negative-NAND gate

NAND gate Negative-NOR gate

The problem with positive and negative logic is that sometimes we do not know what gates should be used. When should we use positive gates? When should we use negative gates? When should we add bubbles to inputs or outputs?

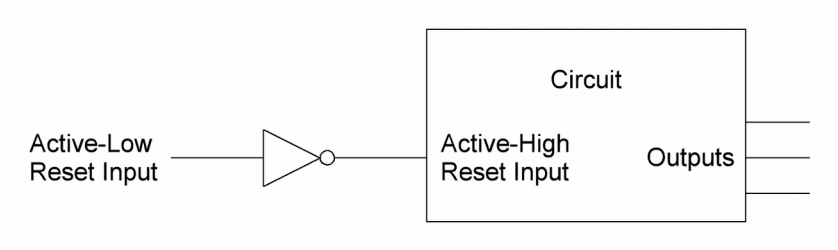
Actually, the key principle is quite simple: **when inputs or outputs are active high, they should not have any bubble. Otherwise, they should.**

For instance, let us consider the following circuit:



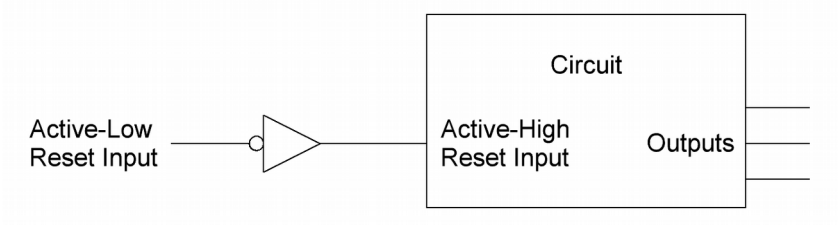
This circuit has an active-high reset input and three outputs:
- When the reset input is 0, this input is inactive.
- When the reset input is 1, this input is active and all the outputs are set to 0.

Now, let us assume that for some reason we want to turn this active-high reset input into an active-low reset input. To do so, a plain NOT gate is enough:



- When the active-low reset input is 0, this input is active and all the outputs are set to 0.
- When the active-low reset input is 1, this input is inactive.

Actually, nothing is wrong with this circuit but something is not clear. When we look at the NOT gate, we do not understand immediately that its input is active low. To make the circuit clearer, we should use a negative-NOT gate instead:

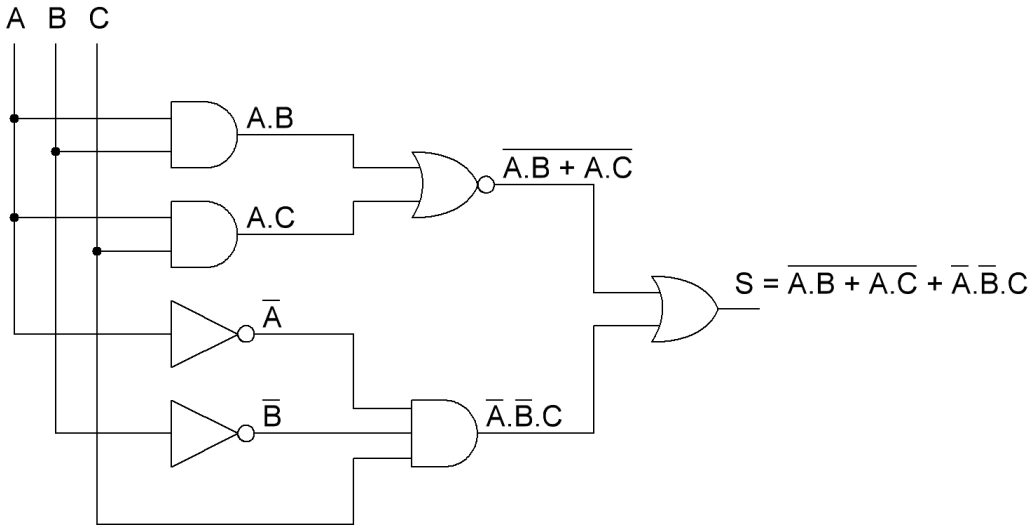


With this symbol, a brief look at the gate is enough to understand that the reset input is active low.

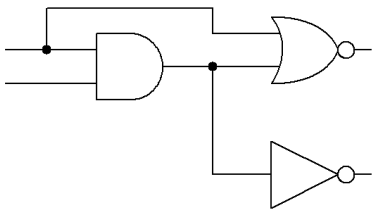
11. Combining Logic Gates

Logic gates can be connected to one another in order to design larger combinational circuits.

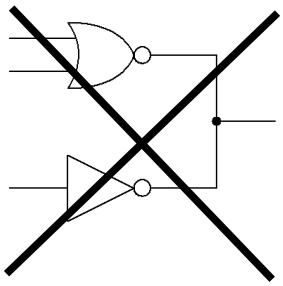
For example:



Outputs and inputs can be connected to different inputs:



But outputs cannot be connected to other outputs:



III. Boolean Algebra

1. Introduction

Boolean algebra deals with expressions made up of variables that can be either 0 or 1. It is used to describe and simplify combinational circuits.

The three fundamental operators of Boolean algebra are the NOT, AND and OR operators.

2. Main Axioms and Theorems

Priority	$A + (B.C) = A + B.C$
NOT	$\overline{\overline{A}} = A$ $\overline{A} + A = 1$ $\overline{A}.A = 0$
AND	$A.(B.C) = (A.B).C = A.B.C$ $A.B = B.A$ $A.A = A$ $A.1 = A$ $A.0 = 0$
OR	$A + (B + C) = (A + B) + C = A + B + C$ $A + B = B + A$ $A + A = A$ $A + 0 = A$ $A + 1 = 1$
EXCLUSIVE OR	$A \oplus B = \overline{A}.B + A.\overline{B} = \overline{A} \oplus \overline{B}$ $\overline{A \oplus B} = A.B + \overline{A}.\overline{B} = \overline{A} \oplus B = A \oplus \overline{B}$ $A \oplus (B \oplus C) = (A \oplus B) \oplus C = A \oplus B \oplus C$ $A \oplus B = B \oplus A$ $A \oplus 0 = A$ $A \oplus 1 = \overline{A}$ $A \oplus A = 0$ $A \oplus \overline{A} = 1$
Distributivity	$A.(B + C) = A.B + A.C$ $A + B.C = (A + B).(A + C)$
De Morgan's Theorems	$\overline{A.B} = \overline{A} + \overline{B}$ $\overline{A + B} = \overline{A}.\overline{B}$
Other Theorems	Theorem 1: $A + A.B = A$ Theorem 2: $A + \overline{A}.B = A + B$

Demonstration	
Theorem 1	**Theorem 2**
$A + A.B$ $= A.1 + A.B$ $= A.(1 + B)$ $= A.1$ $= A$	$A + \overline{A}.B$ $= (A + \overline{A}).(A + B)$ → *distributivity* $= 1.(A + B)$ $= A + B$

3. Principle of Duality

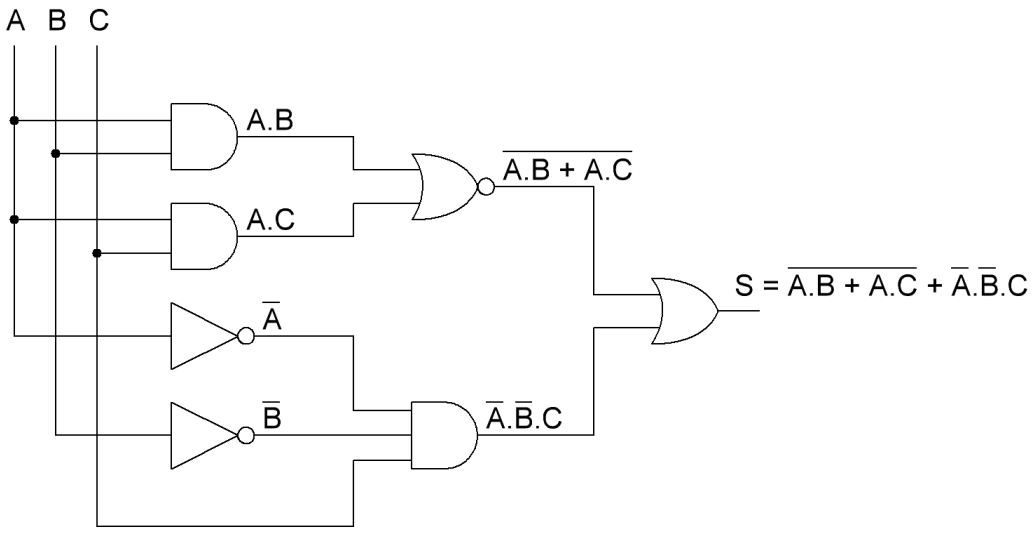
Boolean identities are still true if zeros are replaced by ones and AND operators are replaced by OR operators (and vice versa).

Examples		
$1 + 0 = 1$	↔	$0.1 = 0$
$X + 1 = 1$	↔	$X.0 = 0$
$X.\overline{X} = 0$	↔	$X + \overline{X} = 1$
$X + \overline{X}.Y = X + Y$	↔	$X.(\overline{X} + Y) = X.Y$

4. Algebraic Simplification

Axioms and theorems of Boolean algebra can be used to simplify Boolean expressions in order to reduce the number of gates of combinational circuits.

For instance, let us simplify the following circuit:



$S = \overline{A.B + A.C} + \overline{A}.\overline{B}.C$ → *De Morgan's Theorem*
$S = \overline{A.B}.\overline{A.C} + \overline{A}.\overline{B}.C$ → *De Morgan's Theorem*
$S = (\overline{A} + \overline{B}).(\overline{A} + \overline{C}) + \overline{A}.\overline{B}.C$
$S = \overline{A}.\overline{A} + \overline{A}.\overline{C} + \overline{A}.\overline{B} + \overline{B}.\overline{C} + \overline{A}.\overline{B}.C$
$S = \overline{A} + \overline{A}.\overline{C} + \overline{A}.\overline{B} + \overline{B}.\overline{C} + \overline{A}.\overline{B}.C$ → *Theorem 1:* $\overline{A} + \overline{A}.\overline{C} = \overline{A}$
$S = \overline{A} + \overline{A}.\overline{B} + \overline{B}.\overline{C} + \overline{A}.\overline{B}.C$ → *Theorem 1:* $\overline{A} + \overline{A}.\overline{B} = \overline{A}$
$S = \overline{A} + \overline{B}.\overline{C} + \overline{A}.\overline{B}.C$ → *Theorem 1:* $\overline{A} + \overline{A}.\overline{B}.C = \overline{A}$
$S = \overline{A} + \overline{B}.\overline{C}$

There are several ways to symbolize this circuit. Here are two examples:

Note: $\overline{A} + \overline{B}.\overline{C} = \overline{A} + \overline{B + C} = \overline{A.(B + C)}$

Other examples of algebraic simplifications:

$X1 = (A + B + C.D).(\overline{A} + B).(\overline{B} + C)$
$X1 = (A.\overline{A} + A.B + \overline{A}.B + B.B + \overline{A}.C.D + B.C.D).(\overline{B} + C)$
$X1 = (A.B + \overline{A}.B + B + \overline{A}.C.D + B.C.D).(\overline{B} + C)$ → *Theorem 1:* $B + A.B + \overline{A}.B + B.C.D = B$
$X1 = (B + \overline{A}.C.D).(\overline{B} + C)$
$X1 = B.\overline{B} + B.C + \overline{A}.\overline{B}.C.D + \overline{A}.C.C.D$
$X1 = B.C + \overline{A}.\overline{B}.C.D + \overline{A}.C.D$ → *Theorem 1:* $\overline{\boldsymbol{A}}.\overline{B}.\boldsymbol{C.D} + \overline{\boldsymbol{A}}\boldsymbol{.C.D} = \overline{A}.C.D$
$X1 = B.C + \overline{A}.C.D$

$X2 = \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$
$X2 = \overline{A}.B.(\overline{C} + C) + A.\overline{B}.C + A.B.(\overline{C} + C)$
$X2 = \overline{A}.B + A.\overline{B}.C + A.B$
$X2 = B.(\overline{A} + A) + A.\overline{B}.C$
$X2 = B + A.\overline{B}.C$ → *Theorem 2:* $\boldsymbol{B} + A.\overline{\boldsymbol{B}}.C = B + A.C$
$X2 = B + A.C$

$X3 = A.B.\overline{D} + \overline{B}.C.D + A.B.\overline{C} + B.C.D + \overline{A}.C.D$
$X3 = A.B.\overline{D} + C.D.(\overline{B} + B + \overline{A}) + A.B.\overline{C}$
$X3 = A.B.\overline{D} + C.D + A.B.\overline{C}$
$X3 = A.B.(\overline{C} + \overline{D}) + C.D$ → *De Morgan's Theorem*
$X3 = A.B.\overline{C.D} + C.D$ → *Theorem 2:* $A.B.\overline{\boldsymbol{C.D}} + \boldsymbol{C.D} = A.B + C.D$
$X3 = A.B + C.D$

5. Canonical Forms of Boolean Expressions

5.1. Sum of Products and Product of Sums

In Boolean algebra, the result of logical OR operations can be called a 'sum' and the result of logical AND operations can be called a 'product'.

For instance, the following expressions are sums of products:
- $A.B.C + B.C.D + A.\overline{B}.C.\overline{D}.E + A.B.C.\overline{D}.\overline{E}$
- $A + A.B + B.\overline{A} + B + \overline{A}.C.D + B.C.D$
- $\overline{B} + A.C + \overline{A}.B + B.\overline{C} + \overline{A}.\overline{B}.C$

And the following expressions are products of sums:
- $(A + \overline{B}).(B + C + D)$
- $(A + \overline{B} + C).(\overline{A} + D + \overline{E}).(C + D + \overline{E}).(A + \overline{B} + C + \overline{D} + E)$
- $A.(B + C + \overline{D}).(A + \overline{C}).(A + D)$

5.2. Minterm Canonical Form (Canonical Sum of Products)

5.2.1. Definition

Boolean expressions can be expressed in a multiplicity of forms, but each of them has only one minterm canonical form.

A 'minterm' is a product made up of all the variables of an expression and each variable can be either complemented or uncomplemented.

For instance, let us consider the following expression:
$A.B.C + B.C.D + A.\overline{B}.C.\overline{D}.E + A.B.C.\overline{D}.\overline{E}$

We can see that this expression is composed of five variables: *A*, *B*, *C*, *D* and *E*. Moreover, it is a sum of four products:
- The first product: *A.B.C* is not a minterm because it does not contain the *D* and *E* variables.
- The second product: *B.C.D* is not a minterm because it does not contain the *A* and *E* variables.
- The third product is a minterm because it contains all the variables of the expression.
- The fourth product is a minterm because it contains all the variables of the expression.

Now, let us consider the following expression:
$A.B.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.\overline{D}$

We can see that this expression is composed of four variables: *A*, *B*, *C* and *D*. Moreover, it is a sum of products and each product contains the four variables. Therefore, this expression is a sum of minterms. It is called either a **'minterm canonical form'** or a **'canonical sum of products'**.

5.2.2. Converting Non-Canonical Sum of Products into Minterm Canonical Form

A minterm canonical form can be determined from any non-canonical sum of products.

For intance, let us determine the minterm canonical form of this expression:
$X = \overline{A}.B.C + A.\overline{B} + A.B.\overline{C}.\overline{D}$

To do so, we must convert each term into a minterm.

This expression is composed of four variables: *A*, *B*, *C* and *D*.

The first term does not contain the *D* variable (complemented or uncomplemented), but we can add it as shown below:
$X = \overline{A}.B.C.\mathbf{(D + \overline{D})} + A.\overline{B} + A.B.\overline{C}.\overline{D}$
$X = \overline{A}.B.C.\mathbf{D} + \overline{A}.B.C.\mathbf{\overline{D}} + A.\overline{B} + A.B.\overline{C}.\overline{D}$

The next term does not contain the *C* and *D* variables, but we can add them in the same way:
$X = \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.\overline{B}.\mathbf{(C + \overline{C})}.\mathbf{(D + \overline{D})} + A.B.\overline{C}.\overline{D}$
$X = \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + (A.\overline{B}.\mathbf{C} + A.\overline{B}.\mathbf{\overline{C}}).(\mathbf{D} + \mathbf{\overline{D}}) + A.B.\overline{C}.\overline{D}$
$X = \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.\overline{B}.\mathbf{C.D} + A.\overline{B}.\mathbf{C}.\overline{\mathbf{D}} + A.\overline{B}.\mathbf{\overline{C}.D} + A.\overline{B}.\mathbf{\overline{C}.\overline{D}} + A.B.\overline{C}.\overline{D}$

The last term is already a minterm.

Therefore, the minterm canonical form of *X* is:
$X = \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.\overline{B}.C.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} + A.B.\overline{C}.\overline{D}$

5.2.3. Relation Between Minterm Canonical Form and Truth Table

A minterm canonical form can be easily determined from a truth table and vice versa.

For instance, let us consider the following truth table:

A	B	C	X	
0	0	0	0	
0	0	1	1	← $\overline{A}.\overline{B}.C$
0	1	0	0	
0	1	1	1	← $\overline{A}.B.C$
1	0	0	0	
1	0	1	0	
1	1	0	1	← $A.B.\overline{C}$
1	1	1	1	← $A.B.C$

According to the truth table, convert each binary pattern **where *X* is 1** into a minterm. To do so, replace a '1' by the variable and a '0' by the complement of the variable.

For instance, here is what the minterm canonical form of *X* should look like:
$X = \overline{A}.\overline{B}.C + \overline{A}.B.C + A.B.\overline{C} + A.B.C$

5.3. Maxterm Canonical Form (Canonical Product of Sums)

5.3.1. Definition

Boolean expressions can be expressed in a multiplicity of forms, but each of them has only one maxterm canonical form.

A 'maxterm' is a sum made up of all the variables of an expression and each variable can be either complemented or uncomplemented.

For instance, let us consider the following expression:
$(A + B + C).(B + C + D).(A + \overline{B} + C + \overline{D} + E).(A + B + C + \overline{D} + \overline{E})$

We can see that this expression is composed of five variables: *A*, *B*, *C*, *D* and *E*. Moreover, it is a product of four sums:

- The first sum: $A + B + C$ is not a maxterm because it does not contain the *D* and *E* variables.
- The second sum: $B + C + D$ is not a maxterm because it does not contain the *A* and *E* variables.
- The third sum is a maxterm because it contains all the variables of the expression.
- The fourth sum is a maxterm because it contains all the variables of the expression.

Now, let us consider the following expression:
$(A + B + C + D).(\overline{A} + B + \overline{C} + D).(\overline{A} + B + C + \overline{D}).(\overline{A} + B + \overline{C} + \overline{D}).(\overline{A} + \overline{B} + \overline{C} + \overline{D})$

We can see that this expression is composed of four variables: *A*, *B*, *C* and *D*. Moreover, it is a product of sums and each sum contains the four variables. Therefore, this expression is a product of maxterms. It is called either a **'maxterm canonical form'** or a **'canonical product of sums'**.

5.3.2. Converting Non-Canonical Product of Sums into Maxterm Canonical Form

A maxterm canonical form can be determined from any non-canonical product of sums.

For intance, let us determine the maxterm canonical form of this expression:
$X = (\overline{A} + C).B$

To do so, we must convert each term into a maxterm.

This expression is composed of three variables: *A*, *B* and *C*.

The first term does not contain the *B* variable (complemented or uncomplemented), but we can add it as shown below:
$X = (\overline{A} + \mathbf{B}.\overline{\mathbf{B}} + C).B$

Then, we can use the distributive property:
$X = (\overline{A} + \mathbf{B} + C).(\overline{A} + \overline{\mathbf{B}} + C).B$

The next term does not contain the *A* and *C* variables, but we can add them in the same way:
$X = (\overline{A} + B + C).(\overline{A} + \overline{B} + C).(\mathbf{A}.\overline{\mathbf{A}} + B + \mathbf{C}.\overline{\mathbf{C}})$
$X = (\overline{A} + B + C).(\overline{A} + \overline{B} + C).(\mathbf{A} + B + \mathbf{C}.\overline{\mathbf{C}}).(\overline{\mathbf{A}} + B + \mathbf{C}.\overline{\mathbf{C}})$
$X = (\overline{A} + B + C).(\overline{A} + \overline{B} + C).(\mathbf{A} + B + \mathbf{C}).(\mathbf{A} + B + \overline{\mathbf{C}}).(\overline{\mathbf{A}} + B + \mathbf{C}).(\overline{\mathbf{A}} + B + \overline{\mathbf{C}})$

Therefore, the maxterm canonical form of *X* is:
$X = (\overline{A} + B + C).(\overline{A} + \overline{B} + C).(A + B + C).(A + B + \overline{C}).(\overline{A} + B + \overline{C})$

5.3.3. Relation Between Maxterm Canonical Form and Truth Table

A maxterm canonical form can be easily determined from a truth table and vice versa.

For instance, let us consider the following truth table:

A	B	C	X	
0	0	0	0	← $A + B + C$
0	0	1	1	
0	1	0	0	← $A + \overline{B} + C$
0	1	1	1	
1	0	0	0	← $\overline{A} + B + C$
1	0	1	0	← $\overline{A} + B + \overline{C}$
1	1	0	1	
1	1	1	1	

According to the truth table, convert each binary pattern **where *X* is 0** into a maxterm. To do so, replace a '0' by the variable and a '1' by the complement of the variable.

For instance, here is what the maxterm canonical form of *X* should look like:
$X = (A + B + C).(A + \overline{B} + C).(\overline{A} + B + C).(\overline{A} + B + \overline{C})$

5.4. Converting Between Canonical Forms

To convert a canonical form into another, list the binary patterns that are not included in a canonical form; then, deduce the other canonical form from these patterns.

For example, let us convert the following minterm canonical form into a maxterm canonical form:
$\overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C}$

The associated binary patterns are:
001, 010, 011, 101, 110

The missing binary patterns are:
000, 100, 111

Therefore, the maxterm canonical form is:
$(A + B + C).(\overline{A} + B + C).(\overline{A} + \overline{B} + \overline{C})$

6. Karnaugh Maps

6.1. Definition

The ‘Karnaugh map’ or ‘K-map’ is a graphical technique used to simplify Boolean expressions.

A Karnaugh map is like a truth table in the sense that it gives the values of outputs according to the values of inputs.

For instance, have a look at the truth table below and its associated Karnaugh map.

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

	BC				
A	**X**	**00**	**01**	**11**	**10** ← Gray Code
	0	1	1	1	0
	1	0	1	1	1

↑ Gray Code

Assuming that n is the number of inputs of a truth table, the number of cells of a Karnaugh map is 2^n. The truth table above has three inputs (A, B, C), therefore its associated Karnaugh map has eight cells.

The cells are divided into two rows and four columns. However, the opposite is also possible (i.e. four rows and two columns). **Lines and rows are ordered in Gray code.**

Each cell contains the value of the output (X) according to the value of the inputs (A, B, C):

- For each cell of the first row, the value of A is 0.
- For each cell of the second row, the value of A is 1.
- For each cell of the first column, the value of B is 0 and the value of C is 0.
- For each cell of the second column, the value of B is 0 and the value of C is 1.
- For each cell of the third column, the value of B is 1 and the value of C is 1.
- For each cell of the fourth column, the value of B is 1 and the value of C is 0.

For instance:

- The top left cell gives the value of X when $A = B = C = 0$.
- The bottom right cell gives the value of X when $A = B = 1$ and $C = 0$.

Let us consider that the three inputs (A, B, C) make up an unsigned integer called N. The cells of the Karnaugh map can be numbered in decimal as follows:

N	A	B	C	X
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

		BC			
	N	**00**	**01**	**11**	**10**
A	**0**	*0*	*1*	*3*	*2*
	1	*4*	*5*	*7*	*6*

As said previously, it is possible to organize the same Karnaugh map with four rows and two columns.

		C	
	N	**0**	**1**
AB	**00**	*0*	*1*
	01	*2*	*3*
	11	*6*	*7*
	10	*4*	*5*

		C	
	X	**0**	**1**
AB	**00**	1	1
	01	0	1
	11	1	1
	10	0	1

In a Karnaugh map, it is preferable to write down the input variables in the same order as in the truth table; in other words, write down the input variables so that they can be read from left to right in the same order (here *A*, *B*, *C*). That way, it will be easier to complete a Karnaugh map from a truth table. For instance, the following Karnaugh map is equivalent to those above but more difficult to fill in from the truth table:

AC \ B	0	1
00	1	0
01	1	1
11	1	1
10	0	1

Karnaugh maps can be used to simplify two-, three-, four- and five-variable expressions. In this chapter, we will limit our study to four variables. Here are some examples of Karnaugh maps. Each cell contains its associated decimal number.

A \ B	0	1
0	*0*	*1*
1	*2*	*3*

Two-variable Karnaugh map

AB \ C	0	1
00	*0*	*1*
01	*2*	*3*
11	*6*	*7*
10	*4*	*5*

Three-variable Karnaugh map

A \ BC	00	01	11	10
0	*0*	*1*	*3*	*2*
1	*4*	*5*	*7*	*6*

Three-variable Karnaugh map

AB \ CD	00	01	11	10
00	*0*	*1*	*3*	*2*
01	*4*	*5*	*7*	*6*
11	*12*	*13*	*15*	*14*
10	*8*	*9*	*11*	*10*

Four-variable Karnaugh map

6.2. Simplifying Boolean Expressions

Let us find the most simplified expression of X according to the following truth table:

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

As shown previously, one of its associated Karnaug maps is:

		BC			
	X	**00**	**01**	**11**	**10**
A	**0**	1	1	1	0
	1	0	1	1	1

Now, we have to encircle groups of adjacent 1 by observing the four following rules:

- The number of cells of a group must be a power of two. That is to say, we can group one, two, four, eight ones, etc.
- All the ones (and only the ones) have to be encircled.
- The number of circles has to be as small as possible.
- The size of a circle has to be as large as possible.

		BC			
	X	**00**	**01**	**11**	**10**
A	**0**	1	1	1	0
	1	0	1	1	1

The number of circles gives the number of terms of the Boolean expression (a term is a product). In our example, we have three circles, therefore the Boolean expression will have three terms. In other words, the most simplified expression of X will be a sum of three products.

Now, each group has to be converted into a product by observing the three following rules:

- If a variable has different values in the group (i.e. 0 and 1), this variable is ignored and does not appear in the product.
- If a variable is always 0 in the group, the complemented variable appears in the product.
- If a variable is always 1 in the group, the uncomplemented variable appears in the product.

For instance, let us start by considering the leftmost circle:

		BC			
	X	**00**	**01**	**11**	**10**
A	**0**	1	1	1	0
	1	0	1	1	1

- The A variable is always 0. Therefore, $\overline{A}$ will appear in the product.
- The B variable is always 0. Therefore, $\overline{B}$ will appear in the product.
- The C variable is both 0 and 1. Therefore, this variable is ignored.

Consequently, the product associated with this circle is: $\mathbf{\overline{A}.\overline{B}}$

Now, let us consider the rightmost circle:

		BC			
	X	**00**	**01**	**11**	**10**
A	**0**	1	1	1	0
	1	0	1	1	1

- The A variable is always 1. Therefore, A will appear in the product.
- The B variable is always 1. Therefore, B will appear in the product.
- The C variable is both 0 and 1. Therefore, this variable is ignored.

Consequently, the product associated with this circle is: **A.B**

Finally, let us consider the central circle:

		BC			
	X	**00**	**01**	**11**	**10**
A	**0**	1	1	1	0
	1	0	1	1	1

- The *A* variable is both 0 and 1. Therefore, this variable is ignored.
- The *B* variable is both 0 and 1. Therefore, this variable is ignored.
- The *C* variable is always 1. Therefore, *C* will appear in the product.

Consequently, the product associated with this circle is *C*.

According to this Karnaugh map, we can conclude that the most simplified expression of *X* is:
$X = \overline{A}.\overline{B} + A.B + C$

The Karnaugh map gives the most simplified expression with fundamental operators only. Therefore, once this expression has been obtained, it may be possible to find further simplification by using the EXCLUSIVE OR operator.

So, we can write down that:
$X = \overline{A \oplus B} + C$

Note:
It is noteworthy that a Karnaugh map can be considered as a cylinder (both vertically and horizontally). In other words, the leftmost column is adjacent to the rightmost column and the top line is adjacent to the bottom line. Therefore, it is possible to group some 1s to the left with some 1s to the right and some 1s to the top with some 1s to the bottom.

Example:

		CD			
	X	**00**	**01**	**11**	**10**
AB	**00**	1	0	0	0
	01	0	0	0	0
	11	1	0	0	1
	10	1	0	0	0

$X = A.B.\overline{D} + \overline{B}.\overline{C}.\overline{D}$

In fact, two cells are adjacent when only one variable changes between them.

IV. Solving Problems and Designing Combinational Circuits

In order to illustrate the main techniques of solving problems, we will go through a practical example.

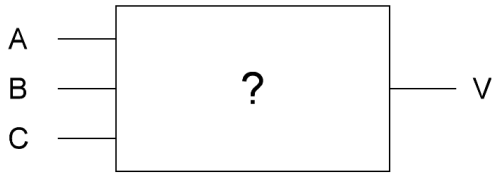
Oil Tanker

The hold of an oil tanker contains three oil tanks (*a*, *b*, *c*) that can be filled independently. Once the tanks have been filled, an indicator light is switched on if the trim is correct; that is to say, if the oil has been evenly distributed amongst the three tanks. There is one sensor per tank (*A*, *B*, *C*) that indicates whether or not a tank is full.

The trim is correct if the three tanks are full, or if the three tanks are empty, or if the *b* tank is full and the others empty, or if the *b* tank is empty and the others full.

System inputs	System output
$A = 1$, if *a* is full	$V = 1$, if the trim is correct
$B = 1$, if *b* is full	
$C = 1$, if *c* is full	

We have to determine the simplest expression of the *V* output in order to design the combinational circuit that solves this problem:



First of all, let us determine the truth table of the *V* output:

A	B	C	V	
0	0	0	**1**	← The three tanks are empty
0	0	1	0	
0	1	0	**1**	← The *b* tank is full and the others empty
0	1	1	0	
1	0	0	0	
1	0	1	**1**	← The *b* tank is empty and the others full
1	1	0	0	
1	1	1	**1**	← The three tanks are full

Then, let us draw the Karnaugh map of the *V* output:

		BC			
	V	**00**	**01**	**11**	**10**
A	**0**	1	0	0	1
	1	0	1	1	0

According to this Karnaugh map, we can determine the most simplified expression of *V* (with the fundamental operators only).

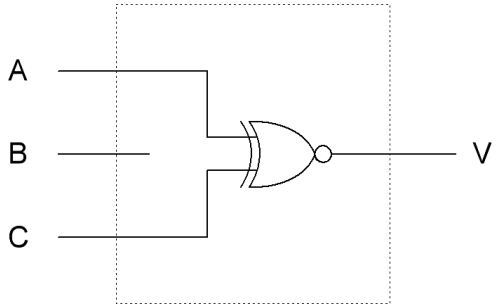
		BC			
	V	**00**	**01**	**11**	**10**
A	**0**	1	0	0	1
	1	0	1	1	0

$$\mathbf{V = \overline{A}.\overline{C} + A.C}$$

We can find further simplification by using the EXCLUSIVE OR operator:

$$\mathbf{V = \overline{A \oplus C}}$$

Finally, here is what the circuit diagram should look like:



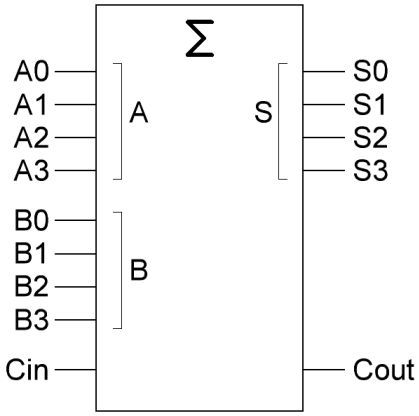
V. Main Combinational Circuits

1. Binary Adder

A 'binary adder' is an electronic device that performs additions between two binary numbers. It usually has the following I/O:

- Two *n*-bit parallel inputs, which contain two numbers to add.
- An input carry.
- An *n*-bit parallel output, which contains the result of the addition (the sum).
- An output carry.

Example of a 4-bit adder:



For instance, if A = **1010**, B = **1100** and Cin = **1**:

```
    1 ← Cin
 1010 ← A
+1100 ← B
10111 ← S
↑
Cout
```

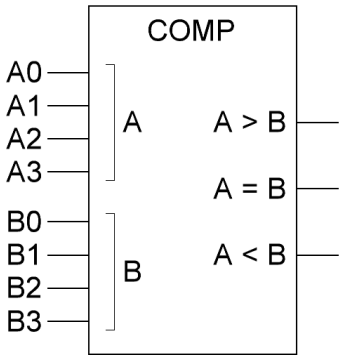
Note:
A, *B* and *S* are 4-bit binary numbers. *A0*, *B0* and *S0* are their LSB respectively.

2. Digital Comparator

A 'digital comparator' is an electronic device that compares two unsigned binary numbers. It usually has the following I/O:

- Two *n*-bit parallel inputs, which contain two numbers to compare.
- An output which specifies if the first number is higher than the second number.
- An output which specifies if the first number is equal to the second number.
- An output which specifies if the first number is lower than the second number.

Example of a 4-bit digital comparator:



For instance:

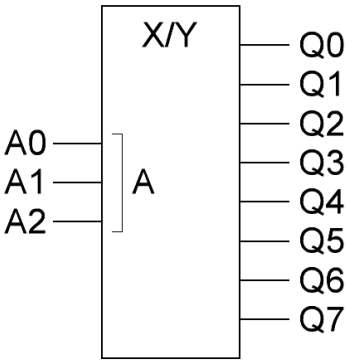
- If $A = \mathbf{0110}$ and $B = \mathbf{0011}$, then $'A > B' = 1$, $'A = B' = 0$ and $'A < B' = 0$.
- If $A = \mathbf{0110}$ and $B = \mathbf{0110}$, then $'A > B' = 0$, $'A = B' = 1$ and $'A < B' = 0$.
- If $A = \mathbf{0110}$ and $B = \mathbf{1110}$, then $'A > B' = 0$, $'A = B' = 0$ and $'A < B' = 1$.

3. Decoder

A ‘decoder’ is an electronic device that detects a particular pattern of bits on its inputs. It usually has the following I/O.

- An n-bit parallel input, which contains a binary number to decode.
- A 2^n-bit parallel output, which specifies the binary number that is currently on the input.

Example of a 1-of-8 decoder (3 input lines, 8 output lines):



A	**A2**	**A1**	**A0**	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
0	0	0	0	**1**	0	0	0	0	0	0	0
1	0	0	1	0	**1**	0	0	0	0	0	0
2	0	1	0	0	0	**1**	0	0	0	0	0
3	0	1	1	0	0	0	**1**	0	0	0	0
4	1	0	0	0	0	0	0	**1**	0	0	0
5	1	0	1	0	0	0	0	0	**1**	0	0
6	1	1	0	0	0	0	0	0	0	**1**	0
7	1	1	1	0	0	0	0	0	0	0	**1**

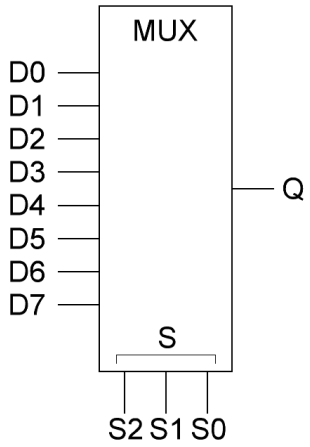
4. Multiplexer

A ‘multiplexer’ is an electronic device that selects one of several digital input lines. It usually has the following I/O:

- n selection lines.
- 2^n data input lines.
- A single data output line.

The selection lines select which data line is sent to the output line.

Example of a multiplexer 8-to-1 (3 selection lines, 8 data input lines, 1 data output line):



S	**S2**	**S1**	**S0**	**Q**
0	0	0	0	D0
1	0	0	1	D1
2	0	1	0	D2
3	0	1	1	D3
4	1	0	0	D4
5	1	0	1	D5
6	1	1	0	D6
7	1	1	1	D7

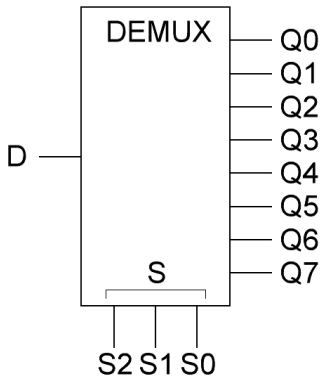
Multiplexers are commonly used to perform parallel-to-serial conversions. Inside a computer, data is transferred by groups of bits (e.g. 8, 16, 32, 64 bits). In other words, several bits are conveyed simultaneously. This is called ‘parallel communication’. However, when it comes to exchanging data between computer peripherals (e.g. printers, cameras, etc.), it would not be convenient to use cables made up of 64 wires. That is the reason why serial communication, which conveys one bit at a time, is preferable. For instance, USB (Universal Serial Bus) is a standard for serial communication.

5. Demultiplexer

A demultiplexer does the reverse operation of a multiplexer. It distributes a single input line to several output lines. It usually has the following I/O:

- A single data input line.
- n selection lines.
- 2^n data output lines.

Example of a 1-of-8 demultiplexer (3 selection lines, 1 data input line, 8 data output lines):



S	S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	0	**D**	0	0	0	0	0	0	0
1	0	0	1	0	**D**	0	0	0	0	0	0
2	0	1	0	0	0	**D**	0	0	0	0	0
3	0	1	1	0	0	0	**D**	0	0	0	0
4	1	0	0	0	0	0	0	**D**	0	0	0
5	1	0	1	0	0	0	0	0	**D**	0	0
6	1	1	0	0	0	0	0	0	0	**D**	0
7	1	1	1	0	0	0	0	0	0	0	**D**

Note:
When the *D* input is always 1, a demultiplexer behaves like a decoder.