Module 2 Digital Logic

Boolean algebra is the mathematics of digital systems. It was developed in 1850's by George Boole. We will use Boolean algebra to minimize logic expressions. Karnaugh Maps will also be used as a graphical tool to minimize Boolean expressions. Boolean algebra has two very important applications – in digital circuitry and in programming.

2.1 Basic Logic Operators

The basic logical operators are AND, OR, NOT, NAND, NOR and XOR.

AND

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

T stands for True and F for False. In the computer environment T and F are changed respectively to 1 and 0.

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

The AND operator is also represented by a dot and this dot is sometimes left out too for example X AND Y can be represented by X.Y or even XY (without the dot between X and Y).

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

The OR operator is also represented by a + for example X OR Y can be represented by X + Y.

NOT

X	NOT X
1	0
0	1

One of the symbols of NOT is a bar over the variable as in $\overline{A}$.

XOR

XOR stands for Exclusive OR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

NAND

NAND stands for Not AND

X	Y	X NAND Y
1	1	0
1	0	1
0	1	1
0	0	1

NOR

NOR stands for Not OR

X	Y	X NOR Y
1	1	0
1	0	0
0	1	0
0	0	1

XNOR

XNOR stands for Not Exclusive OR

X	Y	X XNOR Y
1	1	1
1	0	0
0	1	0
0	0	1

2.2 Logical Expressions

The following are all Boolean expressions:

$$A + \overline{B}$$

$$\overline{AB} + C$$

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

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

2.3 Precedence of Operators

When more than one logical operator is used in a statement, NOT is evaluated first, then AND, and finally OR. As in arithmetic the brackets are worked out first. So let us value the above expressions when A=1, B=0 and C=1.

$A + \overline{B}$	Evaluates to A OR NOT B = 1 OR NOT 0 = 1 OR 1 = 1
$\overline{AB} + C$	Evaluates to NOT (A AND B) OR C = NOT (1 AND 0) OR 1 = NOT (0) OR 1 = 1 OR 1 = 1
$A(B + \overline{C})$	Evaluates to A AND (B OR NOT C) = 1 AND (0 OR NOT 1) = 1 AND (0 OR 0) = 1 AND 0 = 0
$(\overline{A} + B)(A + \overline{C})$	Evaluates to (NOT A OR B) AND (A OR NOT C) = (NOT 1 OR 0) AND (1 OR NOT 1) = (0 OR 0) AND (1 OR 0) = 0 AND 1 = 0

2.3.1 Exercise:

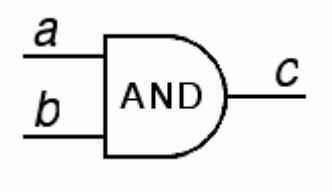
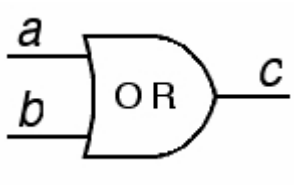
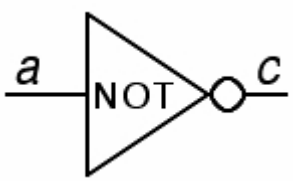
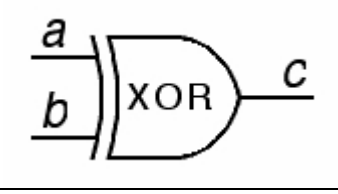
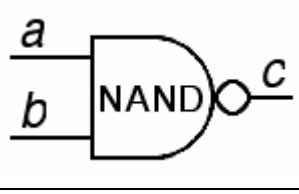
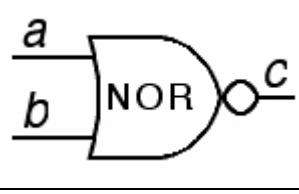
Evaluate the following Boolean expressions given that A=0, B=1 and C=1.

a) $\overline{A} + C$
b) $A + \overline{B}.\overline{C}$
c) $A(\overline{B} + \overline{AC})$
d) $(A + \overline{B})(A + \overline{C})$

2.4 Logic Gates

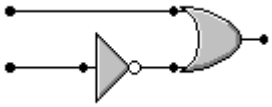
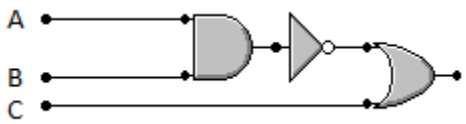
A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), implemented by different voltage levels. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

The following table shows the most common logic gates.

2.4.1 Logic Circuits

Boolean expressions can be expressed as logic circuits. Examples are shown hereunder:

Boolean Expression	Logic Circuit
$A + \overline{B}$	
$\overline{AB} + C$	

2.4.2 Exercise:

1) Draw the logic circuits of the Boolean expressions found in exercise 2.3.1.
2) Draw the truth tables of the same expressions.
3) Prove the following identities by showing that the truth table of the left-hand expression is identical to the truth table of the right-hand expression.
   a) $\overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.B.\overline{C} = \overline{A}.C + A.\overline{C}$
   b) $(A + B).(A + C) = A + B.C$
   c) Show that $\overline{A + \overline{B.C}} = \overline{A}.B.C$

2.5 Laws of (Boolean) Algebra

Commutative Laws

$$A + B = B + A$$
$$A.B = B.A$$

Associative Laws

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

Distributive Law

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

De Morgan's Laws

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

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

(Rule: break the bar and change the sign)

Laws of Absorption

$$X + XY = X$$
$$X(X + Y) = X$$

Double Complement Law

$$\overline{\overline{X}} = X$$

Laws of Tautology

$$X + X = X$$
$$X \,.\, X = X$$
$$X + \overline{X} = 1$$
$$X \,.\, \overline{X} = 0$$
$$X + 1 = 1$$
$$X \,.\, 1 = X$$
$$X + 0 = X$$
$$X \,.\, 0 = 0$$

2.5.1 Examples

(1) Show that $\overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.B.\overline{C} = \overline{A}.C + A.\overline{C}$

LHS	=	$\overline{A}.C.(\overline{B} + B) + A.\overline{C}.(\overline{B} + B)$	(distributive law)
	=	$\overline{A}.C.1 + A.\overline{C}.1$	(tautology law)
	=	$\overline{A}.C + A.\overline{C}$	(tautology law)

(2) Show that $(A + B).(A + C) = A + B.C$

LHS	=	$A.(A + C) + B.(A + C)$	(distributive law)
	=	$A.A + A.C + B.A + B.C$	(distributive law)
	=	$A + A.C + B.A + B.C$	(tautology law)
	=	$A.(1 + C + B) + BC$	(distributive law)
	=	$A.1 + B.C$	(tautology law)
	=	$A + B.C$	(tautology law)

(3) Show that $\overline{A + \overline{B.C}} = \overline{A}.B.C$

LHS	=	$\overline{A}.\overline{\overline{B.C}}$	(De Morgan's law)
	=	$\overline{A}.B.C$	(double complement law)

2.5.2 Exercise

Prove the following identities:

(1) $A + A.B = A$

(2) $C + \overline{B.C} = 1$

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

(4) $\overline{A}.\overline{B} + A.B + \overline{A}.B = B + \overline{A}$

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

(6) $\overline{A}.(A + B) + (B + A.A).(A + \overline{B}) = A + B$

(7) $\overline{A.B}.(\overline{A} + B).(\overline{B} + B) = \overline{A}$

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

2.6 NAND and NOR

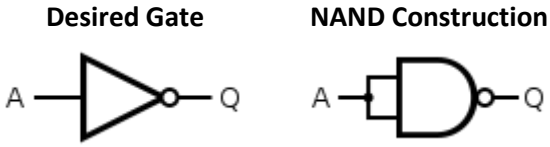
Both the NAND and NOR operations are functionally complete. This means that one can implement any logic circuit using only NAND or only NOR logic gates. Due to this property, NAND and NOR gates are sometimes called "universal gates".
A NAND gate is logically an inverted AND gate. By itself it has the following truth table:

Truth Table

Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

Making other gates by using NAND gates

A NOT gate is made by joining the inputs of a NAND gate.

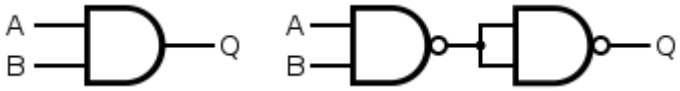


Truth Table

Input A	Output Q
0	1
1	0

An AND gate is made by following a NAND gate with a NOT gate as shown below.

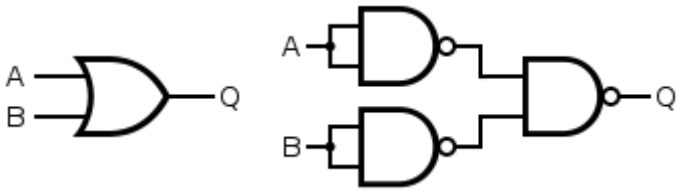
Desired Gate **NAND Construction**



Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

The OR gate is implemented as follows:

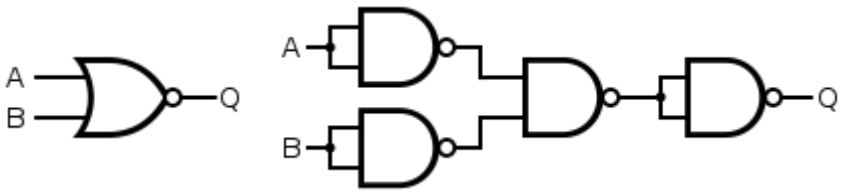
Desired Gate **NAND Construction**



Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

A NOR gate is simply an inverted OR gate.

Desired Gate **NAND Construction**

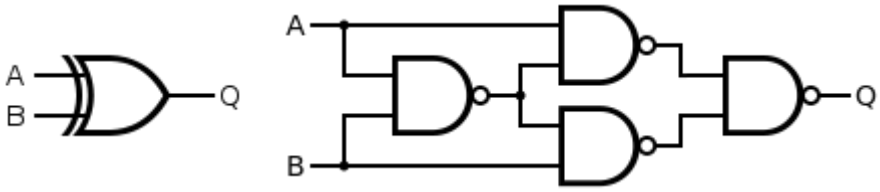


Truth Table

Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	0

The XOR gate is implemented as shown:

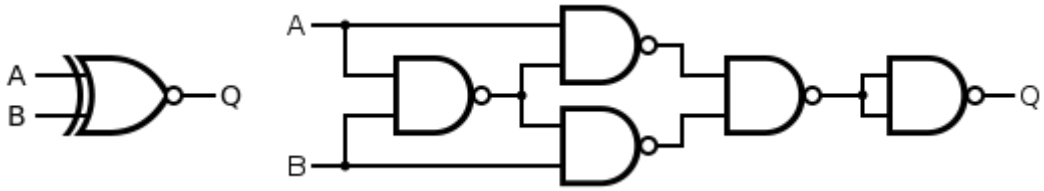
Desired Gate **NAND Construction**



Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	0

An XNOR gate is simply an XOR gate with an inverted output:

Desired Gate **NAND Construction**



Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	1

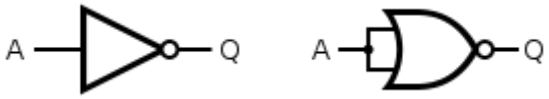
A NOR gate is logically an inverted OR gate. By itself has the following truth table:

Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	0

Making other gates by using NOR gates

The following is the implementation of a NOT gate.

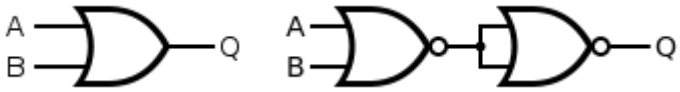
Desired Gate **NOR Construction**



Input A	Output Q
0	1
1	0

The OR gate is simply a NOR gate followed by a NOT gate.

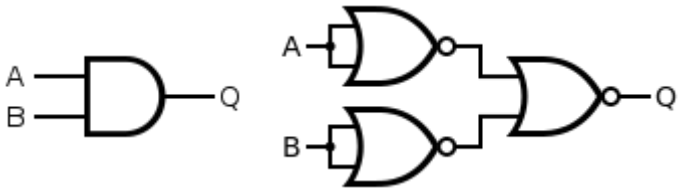
Desired Gate **NOR Construction**



Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

This is the implementation of the AND gate.

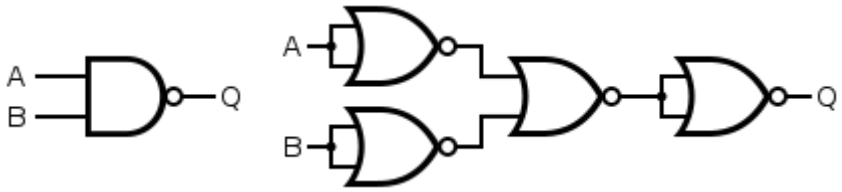
Desired Gate **NOR Construction**



Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

A NAND gate is made using an AND gate in series with a NOT gate:

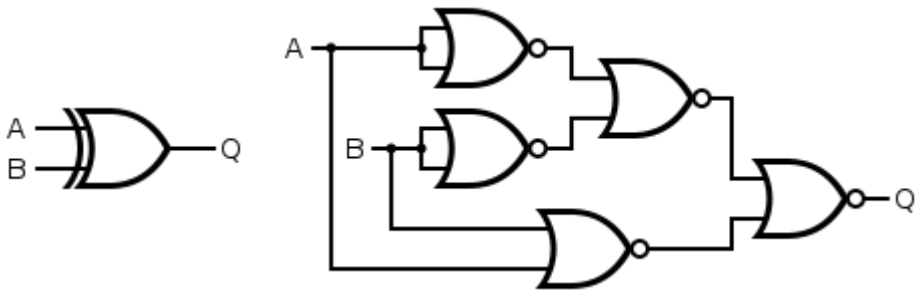
Desired Gate **NOR Construction**



Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

An XOR gate is made by the following connections:

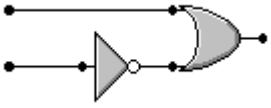
Desired Gate **NOR Construction**



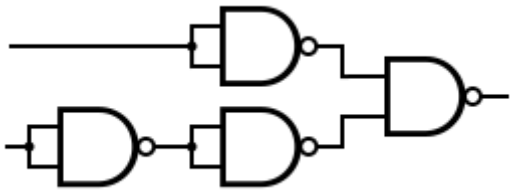
Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	0

Circuit Diagrams using only NAND and NOR Gates

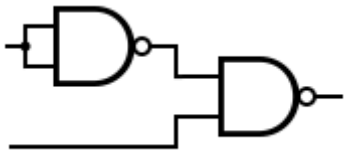
How can we implement the following circuit in (i) NAND gates, and (ii) NOR gates?



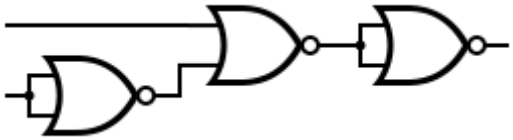
Solution with NAND gates:



Note that the above circuit can be simplified into the following:



Solution with NOR gates:



Exercise

Draw the circuit diagram of the following circuits (first draw them using the AND, OR and NOT gates, then using only NAND gates and finally using only NOR gates).

a) $A + \overline{B}C$
b) $\overline{A}.\overline{B} + C$

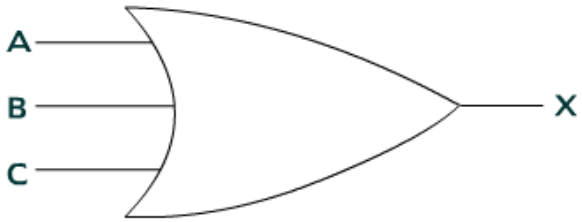
2.7 Gates with Three inputs

AND Gate

$$X = A \bullet B \bullet C$$

Inputs			Output
C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

OR Gate

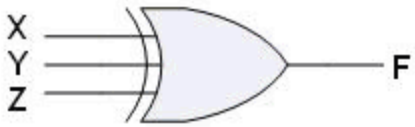


$$X = A + B + C$$

Inputs			Output
C	B	A	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

XOR Gate

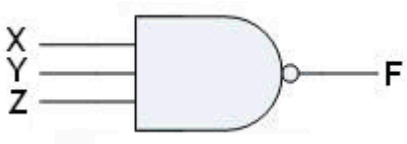
$F = X \oplus Y \oplus Z$



3-input XOR gate			
A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

NAND Gate

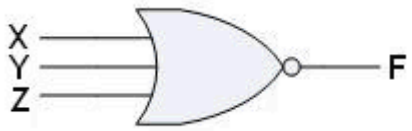
$F = \overline{X \cdot Y \cdot Z}$



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

NOR Gate

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



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

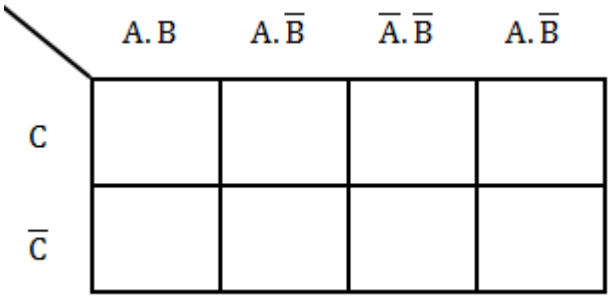
2.8 Karnaugh Maps

Karnaugh map is a minimisation technique for logic circuits. It is simply a method of plotting or mapping all the conditions given in the truth table.

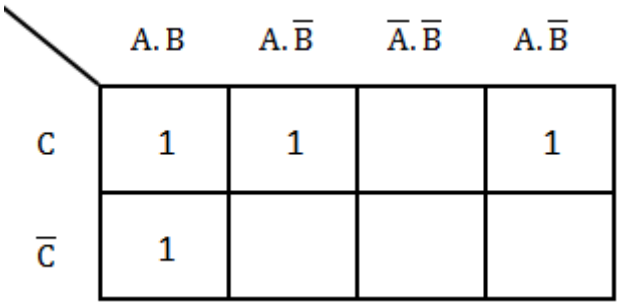
Using Karnaugh Maps to Simplify an Expression

Suppose we are given the expression $\overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$ then we perform the following steps:

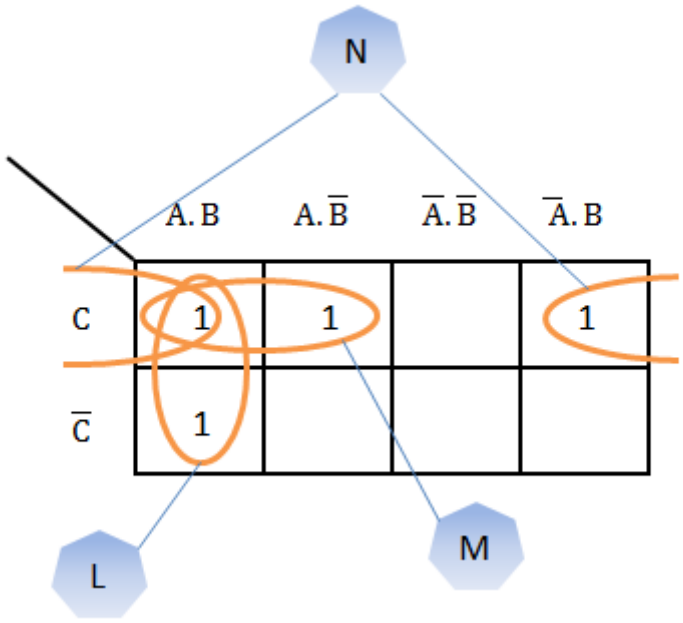
1. Draw a 3-variable Karnaugh Map.



2. Fill it with ones and zeros according to the expression.



3. Group ones in groups of 1, 2, 4 or 8 adjacent cells.



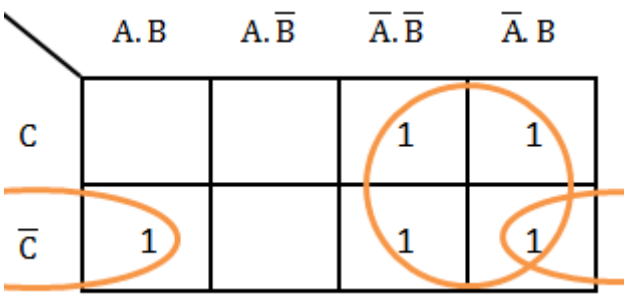
4. Write the simplified expression.

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

Using Karnaugh Maps from Truth Tables

If you are given the truth table shown below then you fill the ones from the table and proceed as we did above.

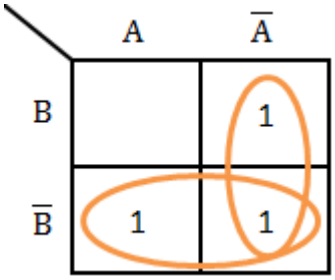
A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Another Example

Simplify the expression $\overline{A}.B + A.\overline{B} + \overline{A}.\overline{B}$.

This gives:



Therefore the simplified expression is $\overline{A} + \overline{B}$.

2.9 Half and full Adders

In electronics, an adder or summer is a digital circuit that performs addition of numbers.

Half adder

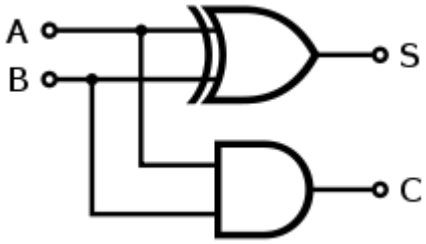
The half-adder adds two bits and has two outputs:

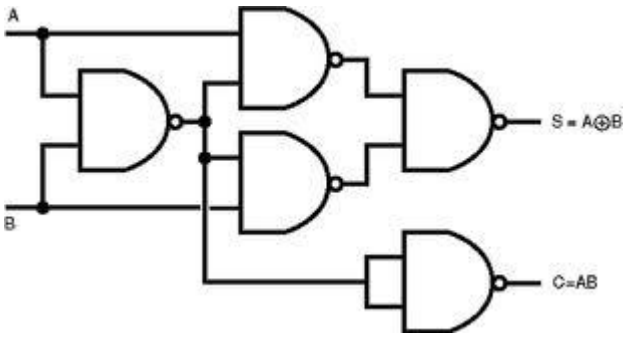
- S: the sum, and
- C: the carry.

Input		Output	
A	*B*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

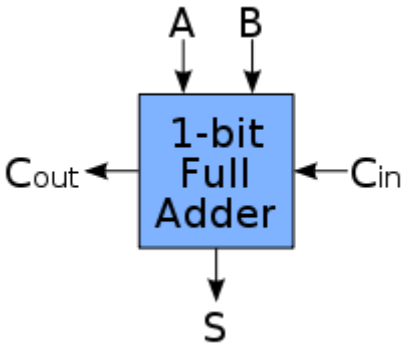
Truth table

Let us find the circuit diagram for C and S. In this case these can be done by observation. C is clearly equal to A AND B. S is A XOR B. The two solutions below give the required circuit diagrams of S and C. The one on the left uses an AND gate and an XOR gate while the one on the right gives the solution in NAND gates.





Full adder

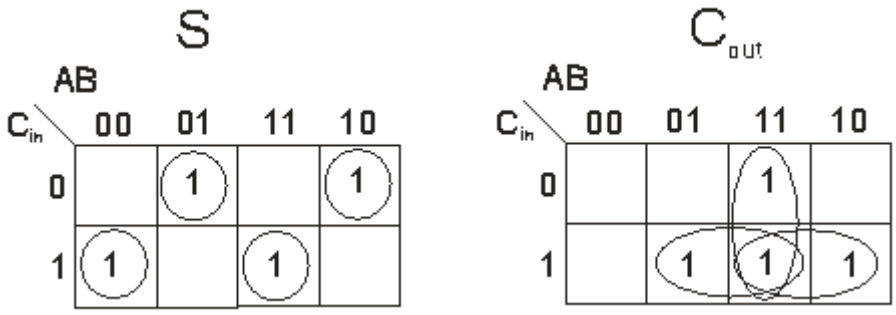


This diagram is a schematic symbol for a 1-bit full adder with C_{in} and C_{out} drawn on sides of block to emphasize their use in a multi-bit adder.

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A, B, and C_{in}; A and B are the operands, and C_{in} is a bit carried in from the next less significant stage. The one-bit full adder's truth table is:

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Let us construct the circuit diagram for C_{out} and S. The diagram below shows the Karnaugh maps for C_{out} and S.

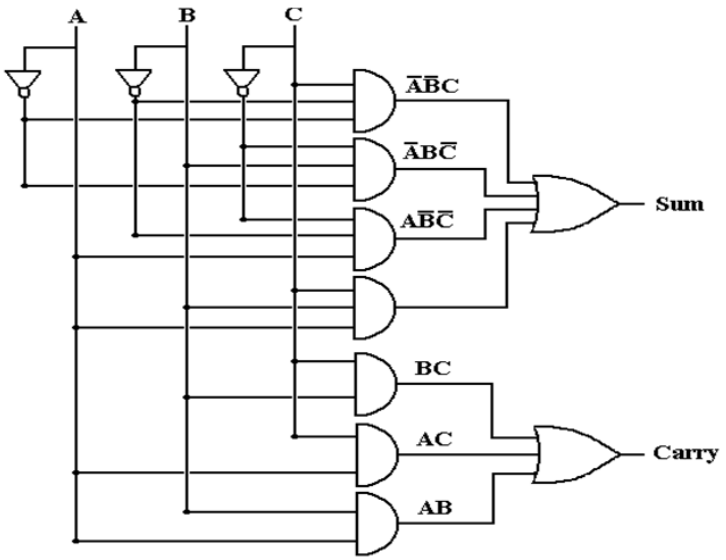


The above Karnaugh maps give the following Boolean equations:

$$S = \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + ABC + A.\overline{B}.\overline{C}$$

$$C_{out} = AB + BC + AC$$

The circuit diagram is shown hereunder. Note that here we are using logic gates with three inputs. This simplifies the diagram.



Another (simpler) circuit that makes use of the XOR gate is shown below.