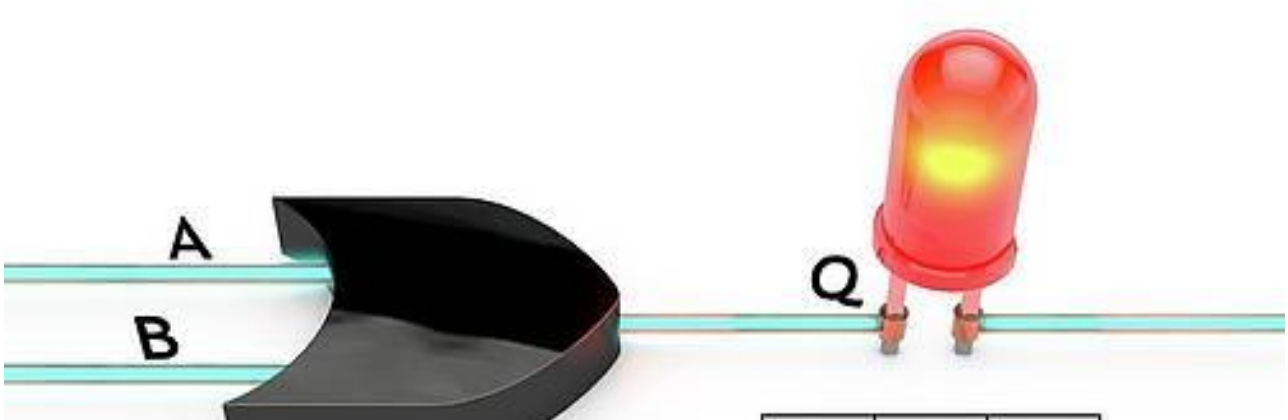
Logic Gates And Boolean Algebra

A/L ICT – Lesson 04



OR Gate

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	1

Computers often chain logic gates together, by taking the output from one gate and using it as the input to another gate. Circuits enables computers to do more complex operations than they could accomplish with just a single gate.

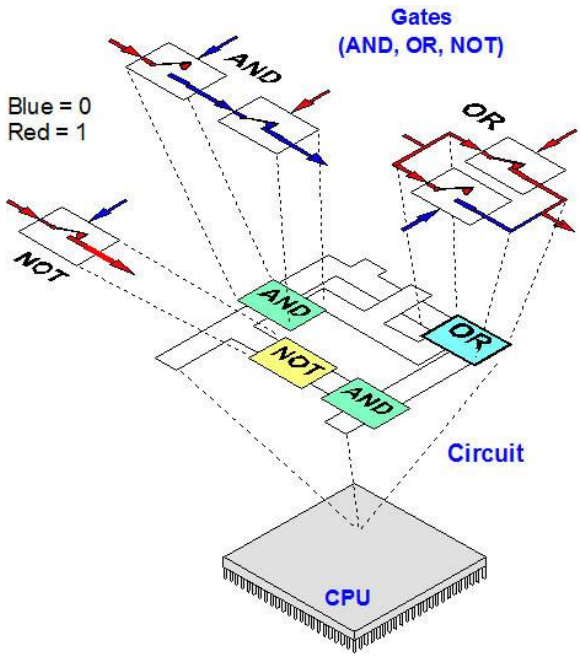
Competency 4: Uses logic gates to design basic digital circuits and devices.

Communication happens in various ways among living beings in day to day life. Drumming was used in the past, as a method of communication. Further, a railway guard in a station blows a horn waving a green flag to signal that a train is going to leave. A green light should be there for the train to start and if it is a red light, the signal is to stop.

Usually, two key are used to open a safe and both keys are needed to open it. Before you start a journey in a car, all its doors should be closed properly. If at least one door is not closed properly, a light will be on or there will be a sound for the driver to signal it. When all the doors are closed, this signal will be stopped. Further, you should wear seat belts if you are seated in the front seats of a car. Otherwise there will be a signal to indicate this. Thus, as we use signals in our day to day life to make decisions, the computer also uses signals.

Circuits which enable building of certain logical conditions using binary values and which enable making certain decisions, are called Logic Circuits. The Computer is made of a large number of complex digital circuits. These electronic circuits are designed as required connecting a large number of basic logical circuits called logic gates. Central Processing Unit is made up of a collection of a large number of logic gates.

The following Figure 4.1 shows a circuit made up of basic. logic gates; AND, OR, and NOT.



The function carried out by a logic gate is giving an output considering an input or several inputs. There are numerous of technical methods to produce logic gates and its internal circuit consists of devices such as transistors, diodes and resistors. According to the way that the circuits are used, logic gates can be classified, into two types.

1. Basic Logic Gates
2. Combinational Logic Gates

Basic Logic Gates

There are three types of basic logic gates. Those are,

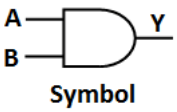
1. AND gate
2. OR gate
3. NOT gate

AND Gate

Let us consider the example given below to understand AND logic.

If the door of your computer lab is locked with a key and padlock, both door key and padlock key are needed to open that door. If both keys are there, you can open the door. If either door key or padlock key is used, you cannot open the door. Further, if keys are not there, you cannot open the door.

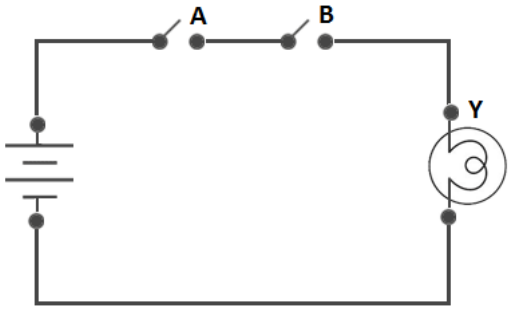
AND GATE



Symbol

A	B	Y = A * B
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table



Electrical Circuit

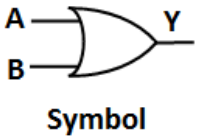
The output Y will be 1 when both the inputs A and B are 1 otherwise Y will be zero

OR Gate

Let us consider the following example to understand OR logic.

- A bus passenger who is in a two door bus can get down from the front or back door.
- If there are several routes to reach your home, you can use any of these routes.

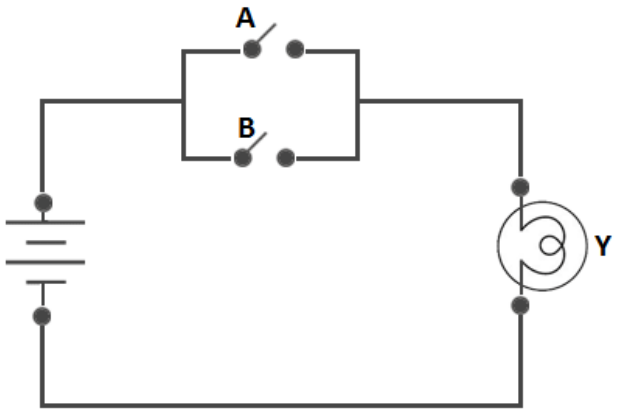
OR GATE



Symbol

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

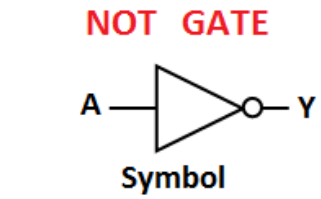


Electrical Circuit

The output Y will be 1 when the input A or B or both are 1. otherwise Y will be zero

NOT Gate

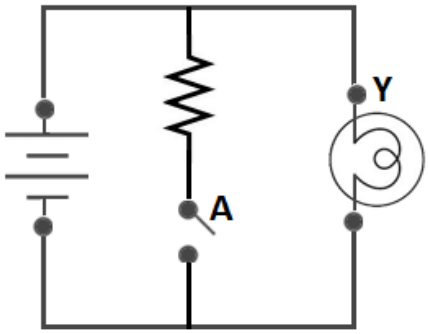
Let us learn about the complement symbol to understand the NOT gate. The task of the NOT gate is to provide the complement as the output. Thus, complement of “0” is “1” and the complement of “1” is “0”.



Symbol

A	$Y = \bar{A}$
0	1
1	0

Truth Table



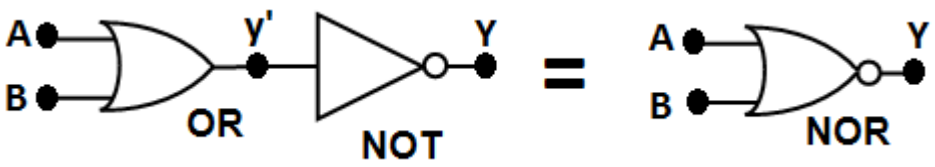
Electrical Circuit

Combinational Logic Gates

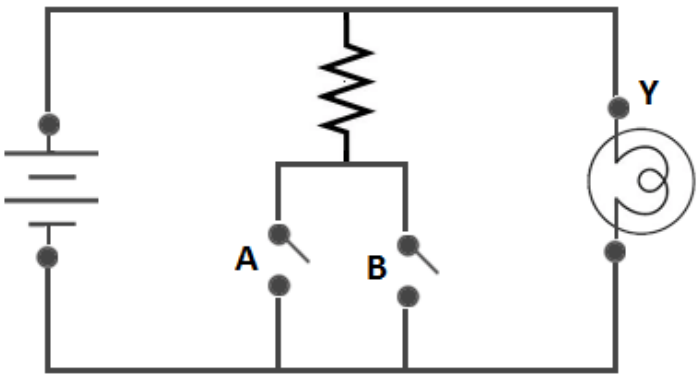
The function of devices such as the computer, calculator, washing machine, microwave oven, mobile phone, modern televisions, digital clock, air condition etc is based on the function of logic gates. There are circuits which are designed with various logic gates to get the required output. Such combinational logic circuit can be designed using basic logic gates.

NOR Gate

The logic gate which indicates NOT OR (i.e. complement operation of the OR operation) is called the NOR gate. The logic of the NOR gate is the combination of OR and NOT to get the output from NOT after leading the output to NOT from OR gate.



NOR GATE



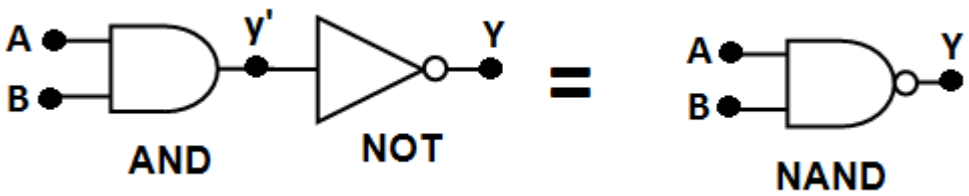
Electrical Circuit

A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

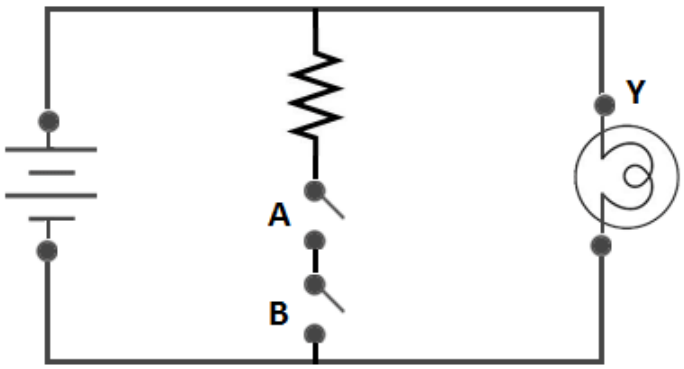
Truth Table

NAND Gate

The logic gate which indicates the complement operation of AND or NOT AND operation is called the NAND gate. This is equivalent to connecting output of AND gate to a NOT gate in series.



NAND GATE



Electrical Circuit

A	B	$Y = \overline{A*B}$
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

Other Gates

XOR Gate

The XOR gate (pronounced as Exclusive OR gate) is a digital logic gate that results a true if and only one of the inputs to the gate is true. Otherwise output is false.

BOOLEAN EXPRESSION

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

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

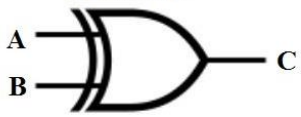
$C = A \oplus B$

Input1

Input2

Output

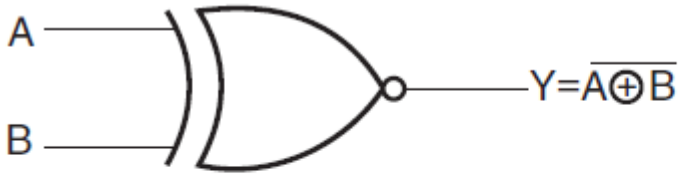
SYMBOL



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

The XNOR gate is a digital logic gate whose function is the logical complement of the XOR gate.



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

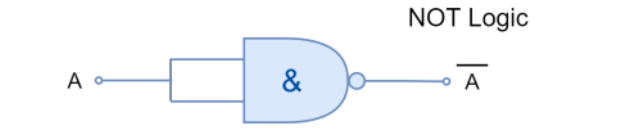
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Universal Gates

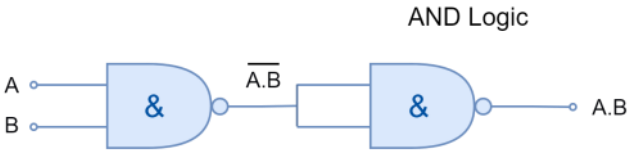
A universal logic gate is a logic gate that can be used to construct all other logic gates. The NAND gate and NOR gates can be considered as universal logic gates.

The advantage of universal gates: NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

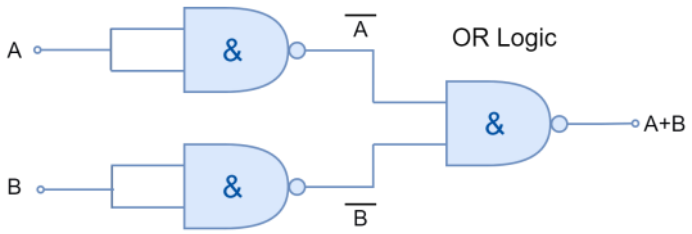
How to implement NOT,AND,OR gates using NAND gates?



A	Q
0	1
1	0

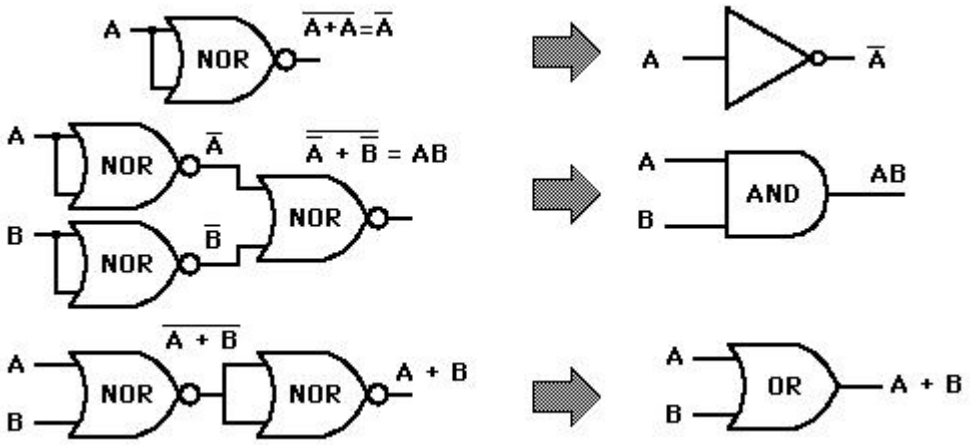


A	B	$\overline{A.B}$	Q
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1



A	B	$\overline{A}$	$\overline{B}$	Q
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

How to implement NOT,AND,OR gates using NOR gates?



Simplifies logic expressions using law of Boolean algebra and Karnaugh map.

Introduction to Boolean laws

What is the requirement of Boolean algebra simplification?

Simplification of Boolean expressions reduces the number of operations and number of logic gates for the implementation.

Most laws exist in two forms. One is multiplicative form and the other is additive form. At the multiplication, variables in the boolean expression are multiplied by each other and at the addition, variables in the boolean expression are added to each other.

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Examples

$A + \overline{A}B$

↓ Applying the previous rule to expand A term

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

↓ $A + AB = A$

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

↓ Factoring B out of 2nd and 3rd terms

$A + B(1)$

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

$A + B$

↓ Applying identity $1A = A$

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

↓ Breaking longest bar

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

↓ Applying identity $\overline{\overline{A}} = A$ wherever double bars of equal length are found

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

↓ Distributive property

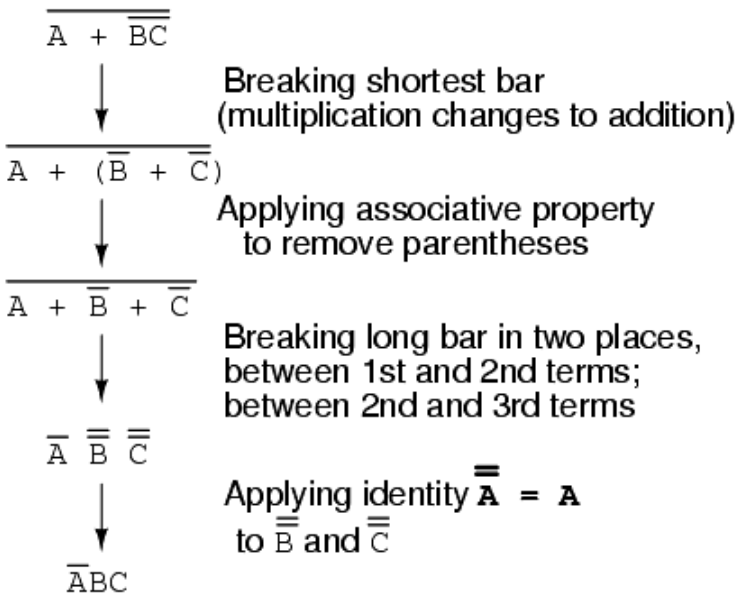
$AA\overline{B} + BCA\overline{B}$

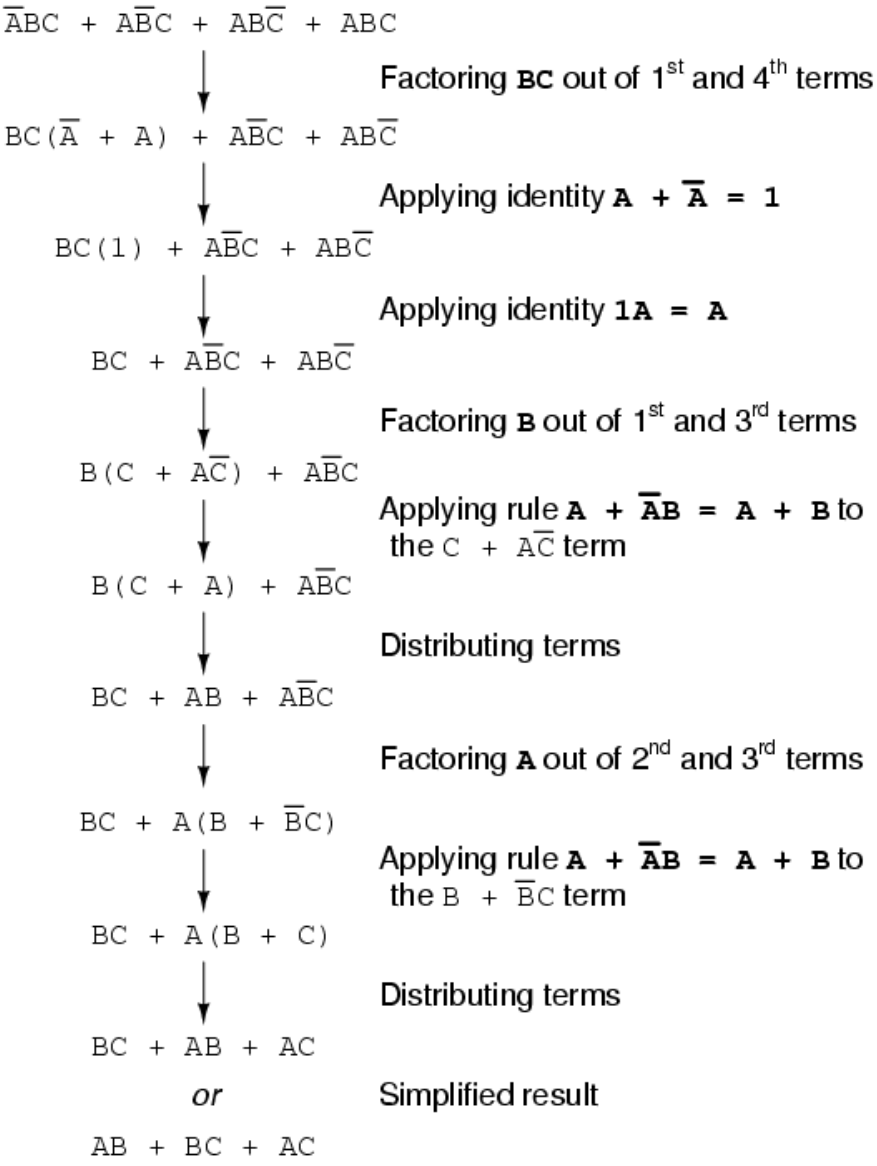
↓ Applying identity $AA = A$ to left term; applying identity $A\overline{A} = 0$ to B and $\overline{B}$ in right term

$A\overline{B} + 0$

↓ Applying identity $A + 0 = A$

$A\overline{B}$


$\overline{A + \overline{BC}}$
Breaking shortest bar
(multiplication changes to addition)
$\overline{A + (\overline{B} + \overline{C})}$
Applying associative property
to remove parentheses
$\overline{A + \overline{B} + \overline{C}}$
Breaking long bar in two places,
between 1st and 2nd terms;
between 2nd and 3rd terms
$\overline{A}\ \overline{\overline{B}}\ \overline{\overline{C}}$
Applying identity $\overline{\overline{A}} = A$
to $\overline{\overline{B}}$ and $\overline{\overline{C}}$
$\overline{A}BC$


$\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
Factoring **BC** out of 1st and 4th terms
$BC(\overline{A} + A) + A\overline{B}C + AB\overline{C}$
Applying identity $A + \overline{A} = 1$
$BC(1) + A\overline{B}C + AB\overline{C}$
Applying identity $1A = A$
$BC + A\overline{B}C + AB\overline{C}$
Factoring **B** out of 1st and 3rd terms
$B(C + A\overline{C}) + A\overline{B}C$
Applying rule $A + \overline{A}B = A + B$ to
the $C + A\overline{C}$ term
$B(C + A) + A\overline{B}C$
Distributing terms
$BC + AB + A\overline{B}C$
Factoring **A** out of 2nd and 3rd terms
$BC + A(B + \overline{B}C)$
Applying rule $A + \overline{A}B = A + B$ to
the $B + \overline{B}C$ term
$BC + A(B + C)$
Distributing terms
$BC + AB + AC$
or
Simplified result
$AB + BC + AC$

Minterms and Maxterms

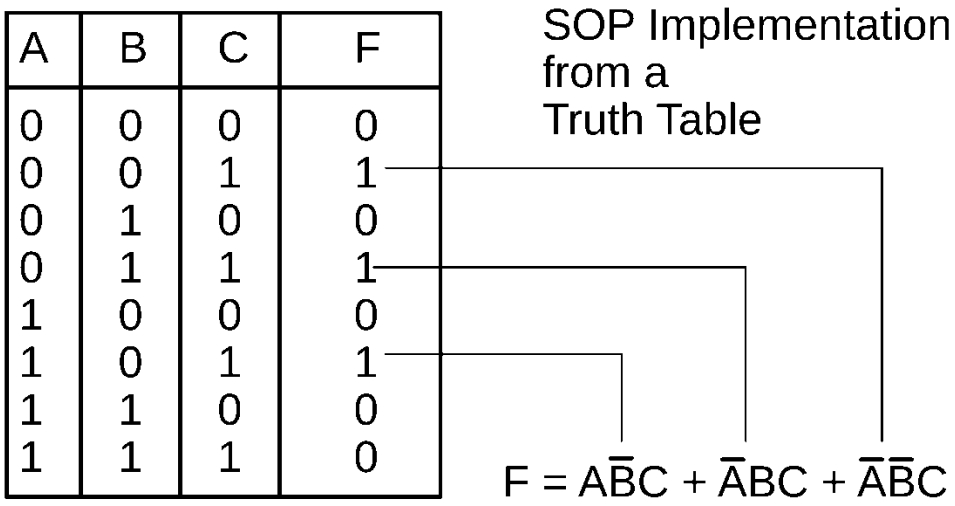
Minterms and Maxterms are the each individual term of SOP and POS form respectively. In Minterm, we look for the functions where the output results in “1” while in Maxterm we look for function where the output results in “0”.

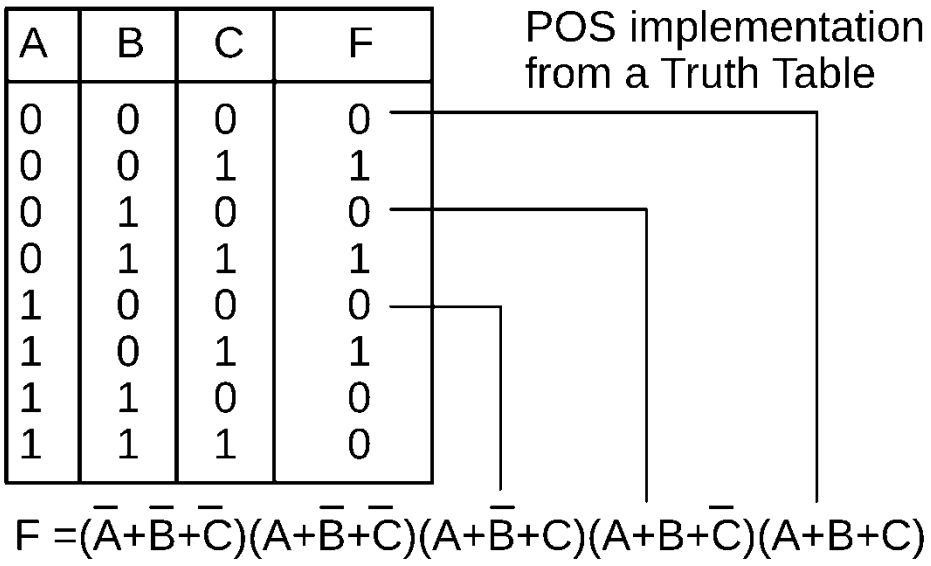
What is SOP?

SOP stands for Sum of Product. SOP form is a set of product(AND) terms that are summed(OR) together. When an expression or term is represented in a sum of binary terms known as minterms and sum of products.

What is POS?

POS stands for product of sum. A technique of explaining a Boolean expression through a set of max terms or sum terms, is known as POS.





Standard SOP form

SOP expressions containing all **Variables** in the **Domain** in each term are in **Standard Form.** Standard product terms are also called Minterms.

- Any non-standard SOP expression may be converted to Standard form by applying Boolean Algebra

$$\begin{aligned}&\overline{A}B\overline{C}+A\overline{C}\\&=\overline{A}B\overline{C}+A\overline{C}(B+\overline{B})\\&=\overline{A}B\overline{C}+AB\overline{C}+A\overline{B}\overline{C}\end{aligned}$$

$$\begin{aligned}&\overline{A}B\overline{C}+A\\&=\overline{A}B\overline{C}+A(B+\overline{B})(C+\overline{C})\\&=\overline{A}B\overline{C}+A(BC+B\overline{C}+\overline{B}C+\overline{B}\overline{C})\\&=\overline{A}B\overline{C}+ABC+AB\overline{C}+A\overline{B}C+A\overline{B}\overline{C}\end{aligned}$$

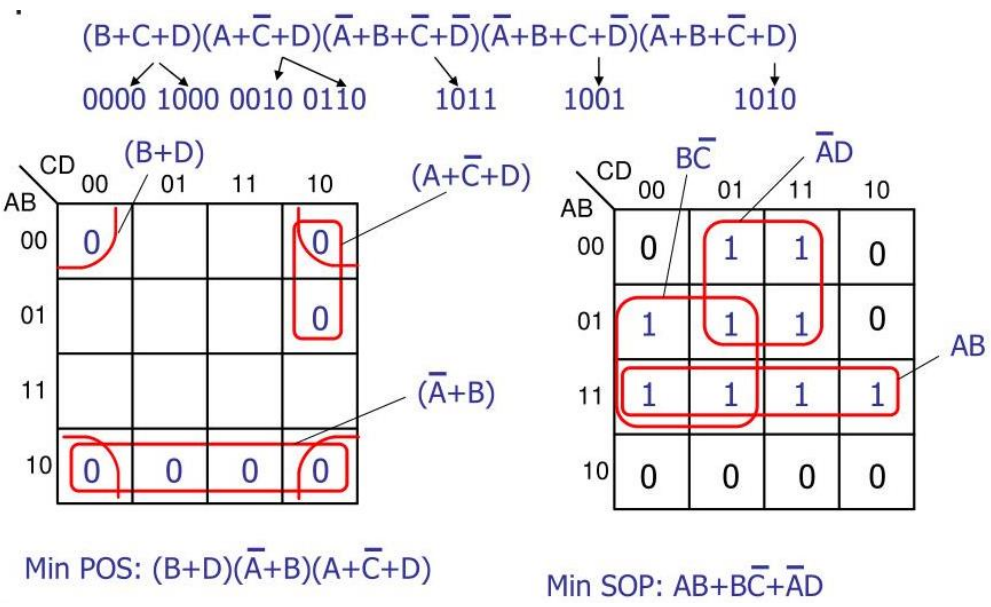
Standard POS form

POS expressions containing all **Variables** in the **Domain** in each term are in **Standard Form.** Standard sum terms are also called **Maxterms**.

- Any non-standard POS expression may be converted to Standard form by applying Boolean Algebra

$$\begin{aligned}&(\overline{A}+B+\overline{C})(A+\overline{C})\\&=(\overline{A}+B+\overline{C})(A+\overline{C}+B\overline{B})\\&=(\overline{A}+B+\overline{C})(A+\overline{C}+B)(A+\overline{C}+\overline{B})\\&=(\overline{A}+B+\overline{C})(A+B+\overline{C})(A+\overline{B}+\overline{C})\end{aligned}$$

Converting between SOP to POS



Kmaps

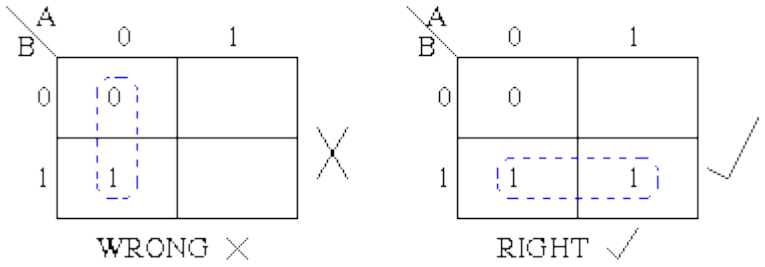
In many digital circuits and practical problems we need to find expression with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of problem. K-map is table like representation but it gives more information than truth table. We fill grid of K-map with 0's and 1's then solve it by making groups.

Steps to solve expression using K-map-

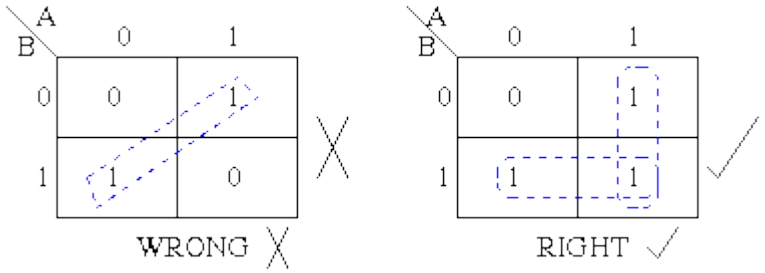
1. Select K-map according to the number of variables.
2. Identify minterms or maxterms as given in problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. For POS put 0's in blocks of K-map respective to the maxterms(1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

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

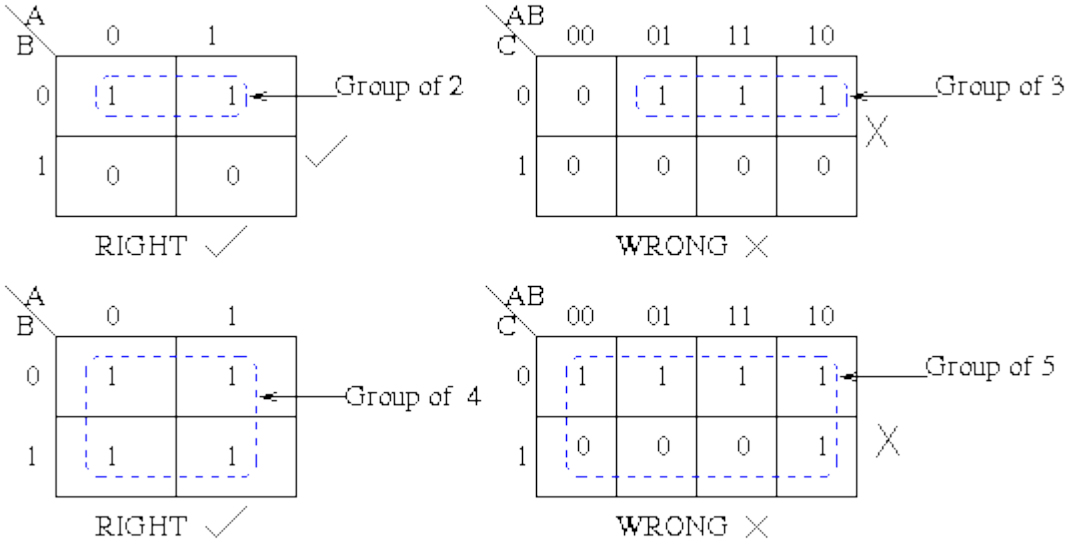
1. Groups may not include any cell containing a zero.



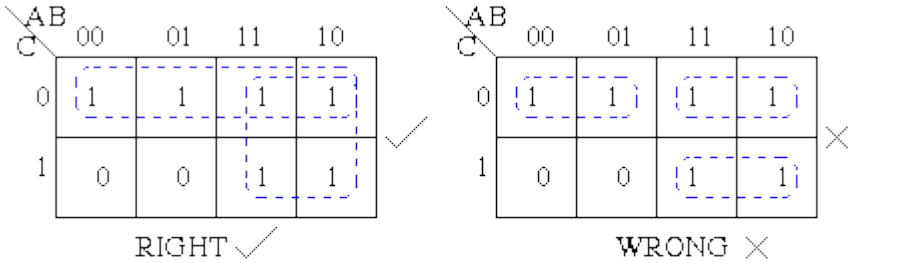
2. Groups may be horizontal or vertical, but not diagonal.



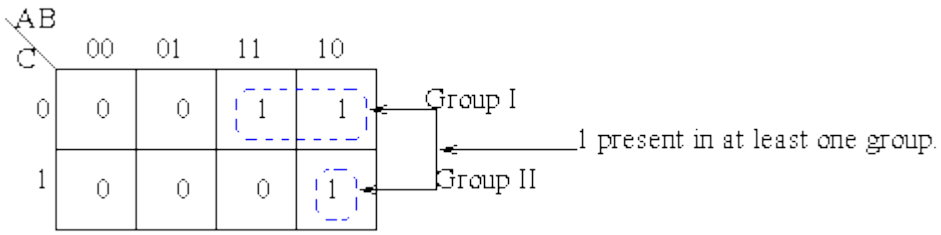
3. Groups must contain 1, 2, 4, 8, or in general 2^n cells. That is if n = 1, a group will contain two 1's since $2^1 = 2$. If n = 2, a group will contain four 1's since $2^2 = 4$.



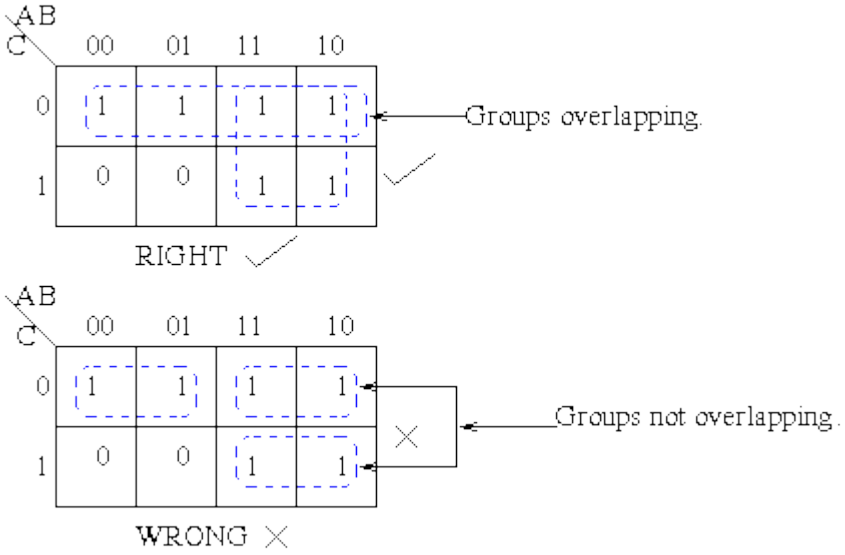
4. Each group should be as large as possible



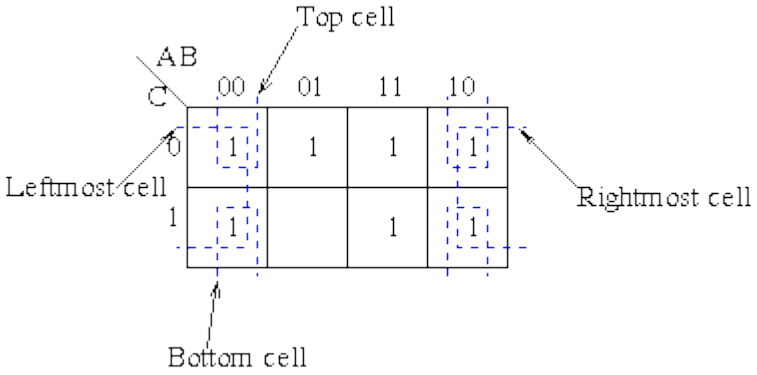
5. Each cell containing a *one* must be in at least one group.



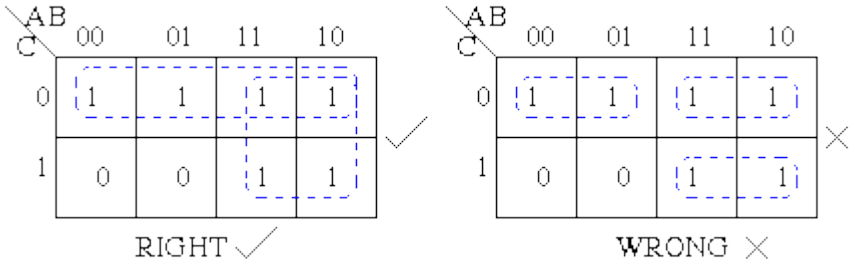
6. Groups may overlap



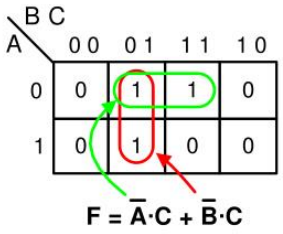
7. Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



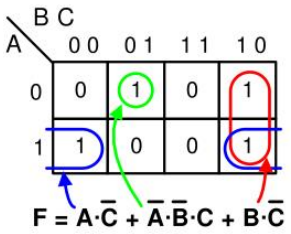
8. There should be as few groups as possible, as long as this does not contradict any of the previous rules.



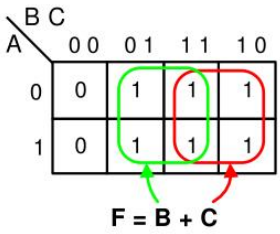
Examples



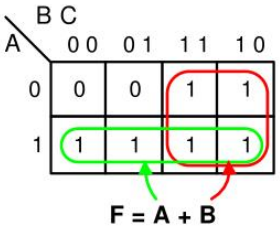
$F = \bar{A}\cdot C + \bar{B}\cdot C$



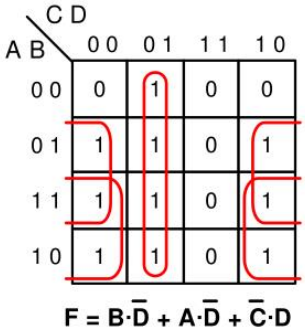
$F = A\cdot\bar{C} + \bar{A}\cdot\bar{B}\cdot C + B\cdot\bar{C}$



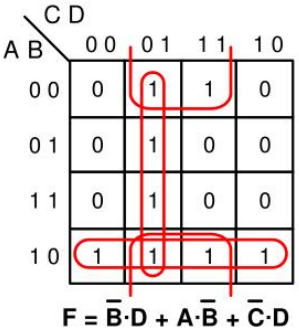
$F = B + C$



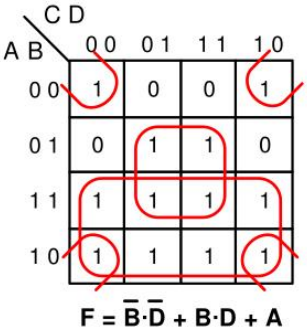
$F = A + B$



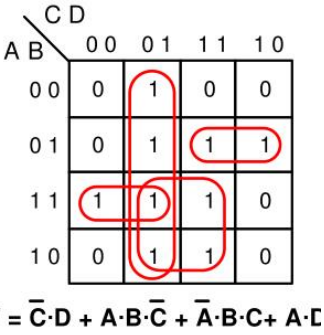
$F = B\cdot\bar{D} + A\cdot\bar{D} + \bar{C}\cdot D$



$F = \bar{B}\cdot D + A\cdot\bar{B} + \bar{C}\cdot D$

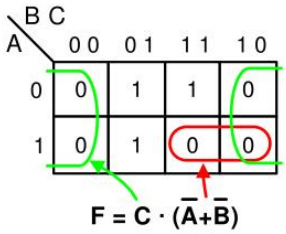


$F = \bar{B}\cdot\bar{D} + B\cdot D + A$

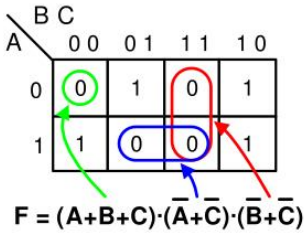


$F = \bar{C}\cdot D + A\cdot B\cdot\bar{C} + \bar{A}\cdot B\cdot C + A\cdot D$

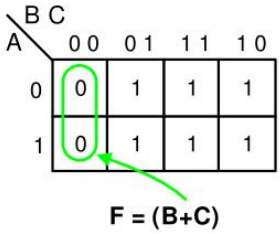
SOP LOOPS ⇧ ⇩ **POS LOOPS**



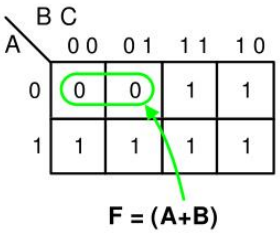
$F = C\cdot(\bar{A}+\bar{B})$



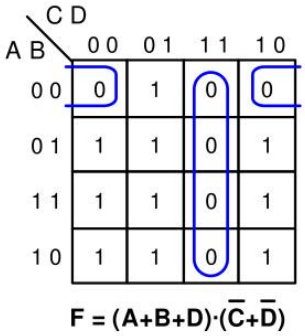
$F = (A+B+C)\cdot(\bar{A}+\bar{C})\cdot(\bar{B}+\bar{C})$



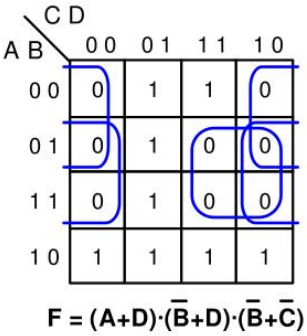
$F = (B+C)$



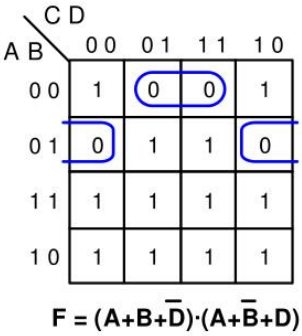
$F = (A+B)$



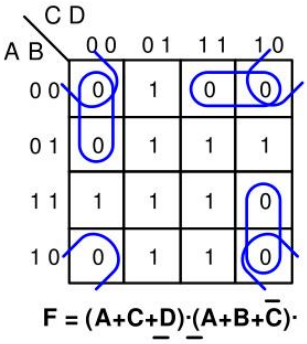
$F = (A+B+D)\cdot(\bar{C}+\bar{D})$



$F = (A+D)\cdot(\bar{B}+D)\cdot(\bar{B}+\bar{C})$



$F = (A+B+\bar{D})\cdot(A+\bar{B}+D)$

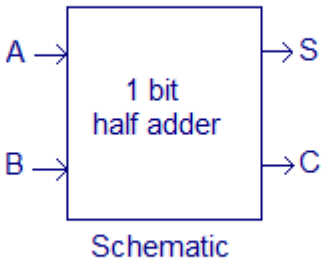


$F = (A+C+D)\cdot(A+B+\bar{C})\cdot(\bar{A}+\bar{C}+D)\cdot(B+D)$

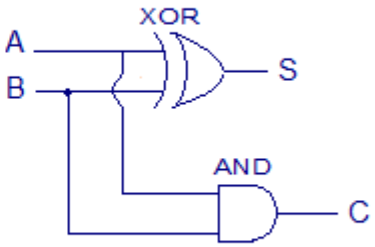
Half Adder is a combinational logic circuit which is designed by connecting one EX-OR gate and one AND gate. The half adder circuit has two inputs: A and B, which add two input digits and generates a carry and a sum. The output obtained from the EX-OR gate is the **sum of the two numbers** while that obtained by AND gate is the carry. There will be no forwarding of carry addition because there is no logic gate to process that. Thus, this is called the Half Adder circuit.

Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table

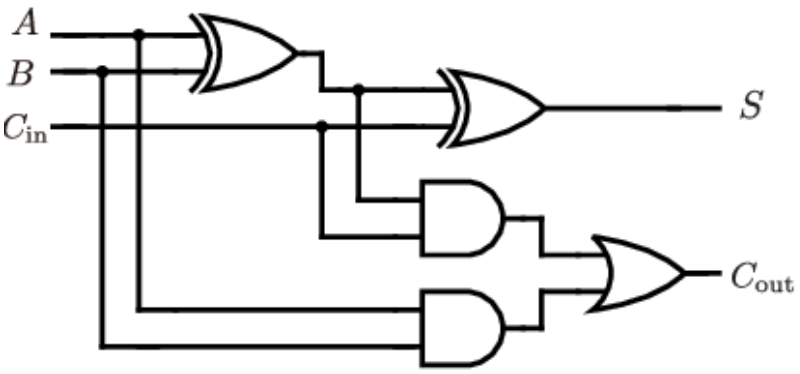


Schematic



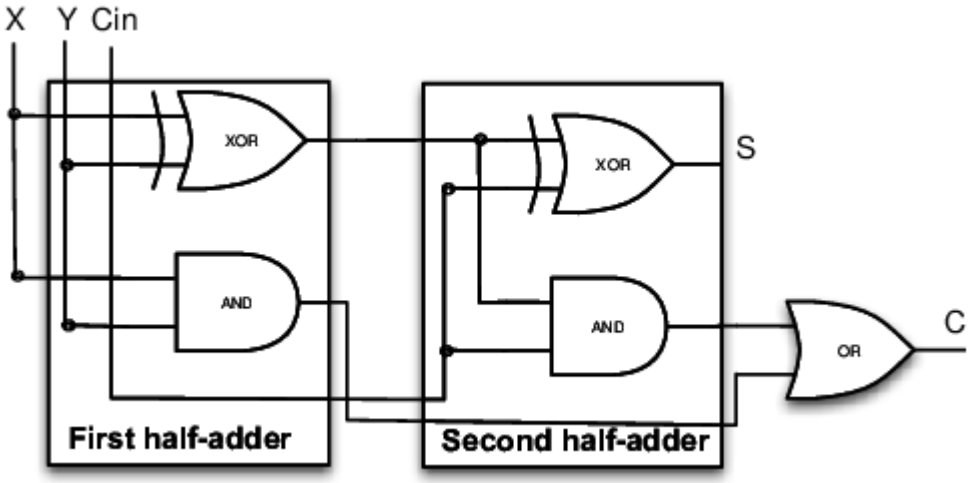
Realization

Full Adder is the circuit that consists of two EX-OR gates, two AND gates, and one OR gate. Full Adder is the adder that adds three inputs and produces two outputs which consist of two EX-OR gates, two AND gates, and one OR gate. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

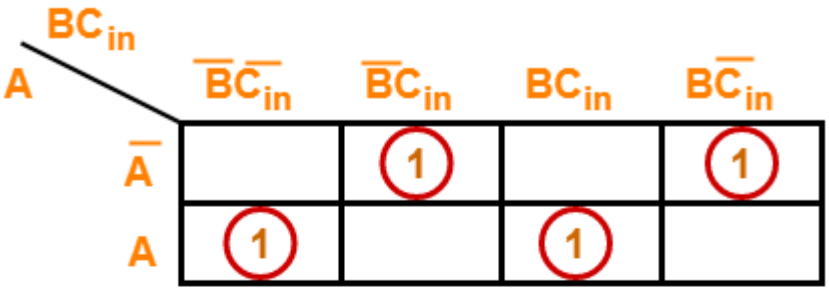


Inputs			**Outputs**	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full adder using two half adders

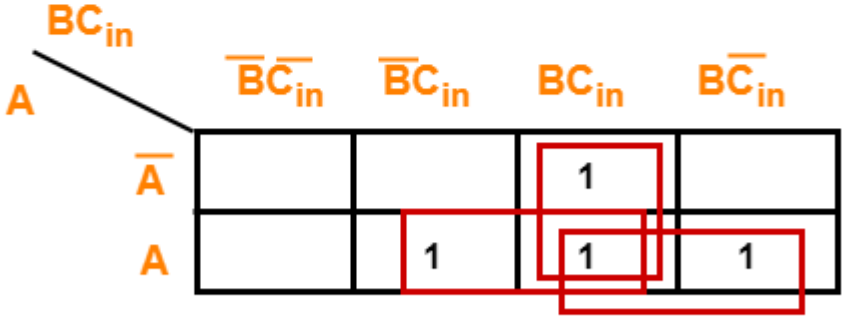


For S:



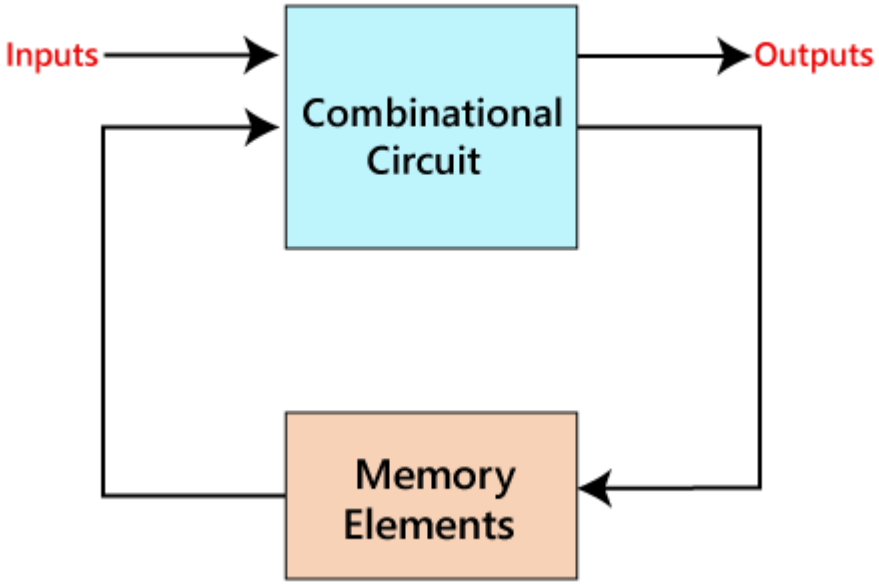
$$S = A \oplus B \oplus C_{in}$$

For C_{in}:



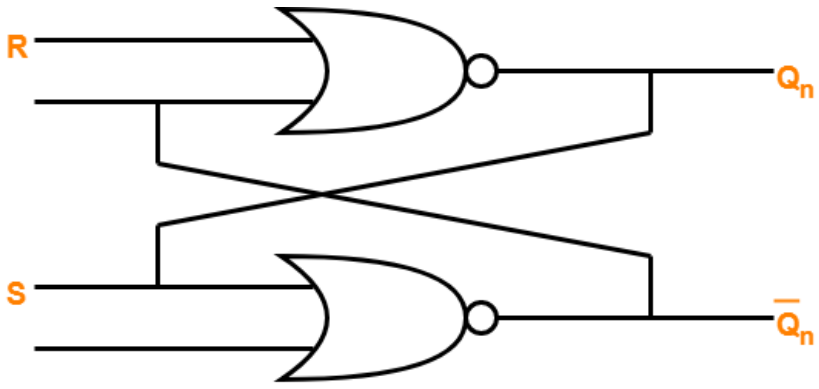
$$C_{out} = AB + BC_{in} + C_{in}A$$

Sequential circuit is a combinational logic circuit that consists of inputs variable (X), logic gates (Computational circuit), and output variable (Z). A combinational circuit produces an output based on input variables only, but a **sequential circuit** produces an output based on **current input and previous output variables**. That means sequential circuits include memory elements that are capable of storing binary information. That binary information defines the state of the sequential circuit at that time. A latch capable of storing one bit of information.

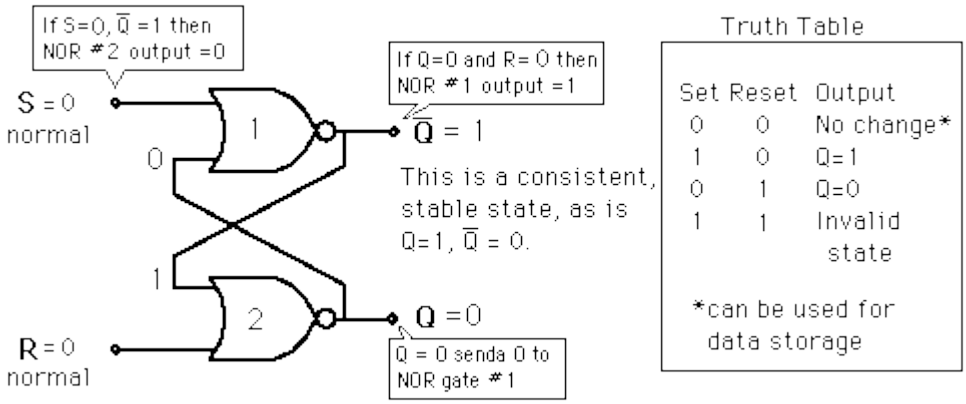


Flip-flops and latches are used as data storage elements. A flip-flop is a device which stores a single bit (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic in electronics.

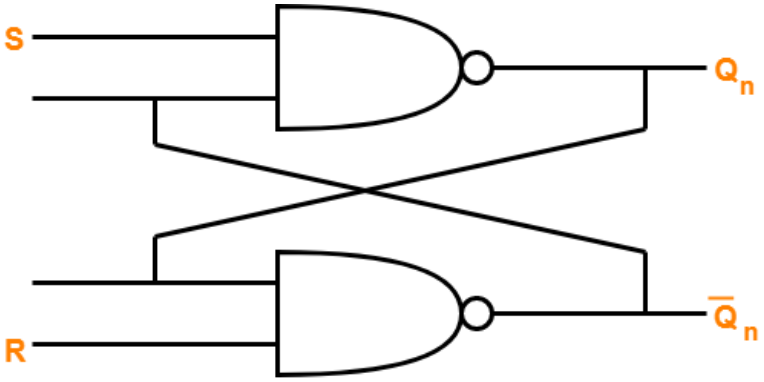
The logic circuit for a **latch constructed using NOR gates** is as shown below.



Latch Construction Using NOR Gates



The logic circuit for a **latch constructed using NAND** gates is as shown below.



Latch Construction Using NAND Gates