FUNDAMENTALS OF DIGITAL DESIGN

[Read chapter 5 of the recommended text book –Digital Electronics with VHDL by William Kleitz]

After studying this module, students should be able to:

- *Design combinational logic systems using Truth Tables and Boolean expressions*
- *Use Boolean Axioms and Theorems to simplify logic circuits*

<u>Synthesis using AND, OR and NOT Gates</u>

Synthesis is the task of designing a new logic network that implements a desired functional behavior.

The logic design procedure, is as follows:

- Start with the problem statement.
- Determine the number of inputs variables and the required number of output variables.
- Derive a truth table that defines the required relationship between input and output.
    - For small networks
- From the Truth Table write out the Boolean expression.
- Simplify each output function (Using Boolean algebra or Karnaugh maps).
- The logic circuit is then drawn from the simplified Boolean expression

Let us consider the example below:

<u>Example 1</u>

Design a circuit to perform the following function. Two power lines are to be monitored for the occurrence of HIGH voltage signals. The function of the circuit is to monitor the state of two sensors attached to the power lines and output a high voltage to trigger an alarm system whenever there is a HIGH (LOGIC 1) on either one or both power lines. The function of the circuit is to continuously monitor the state of the sensors and produce an output logic value 1, whenever the sensors are in the following state: (1, 0), (0, 1) and

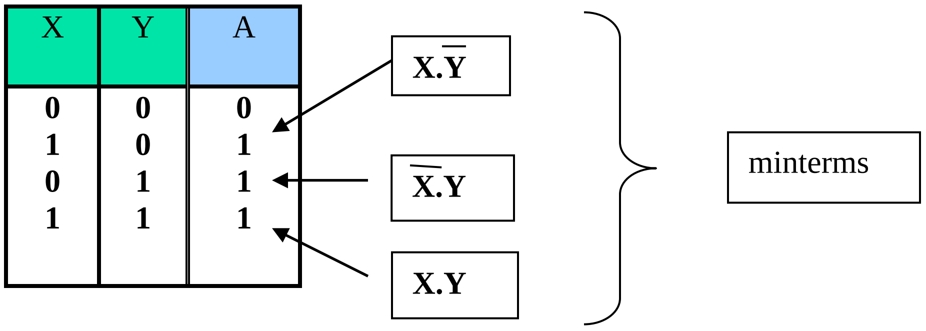
(1, 1). In other words the required functional behavior of the circuit is that the output must be equal to 1 whenever either or both of the sensors are at logic 1.

The above behavior can be summarized in a truth table as shown below:

X	Y	A
0	**0**	**0**
1	**0**	**1**
0	**1**	**1**
1	**1**	**1**

A possible procedure for designing a logic circuit that implements the truth table shown above is to create a product term for each valuation for which the output function A has a value of 1.

Step 2 (Sum Of Products (SoP) realization.)



The desired function is then the sum of these product (AND) terms hence, the name **Sum of Products (SoP) realization.**

$$X.\overline{Y} + \overline{X}.Y + X.Y = A$$

These expressions come in two forms: the Sum of products form is the type described above and is normally referred to as the minterm form in engineering. The above logical expression can also be achieved using the product of sum form which is called maxterm form. However the examples described here employ the minterm form.

Constructing a logic circuit from a Boolean expression

To construct the logic circuit, we start at the output and work towards the input. The equation above shows that the output should be formed by a three input **OR** gate.

$$X.\overline{Y} + \overline{X}.Y + X.Y = A$$

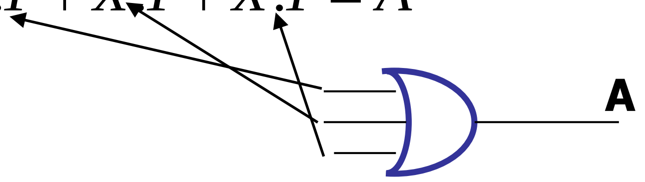


Fig 1

In the second step, an **AND** gate has been added to feed the **X.$\overline{\mathbf{Y}}$** input to the **OR** gate

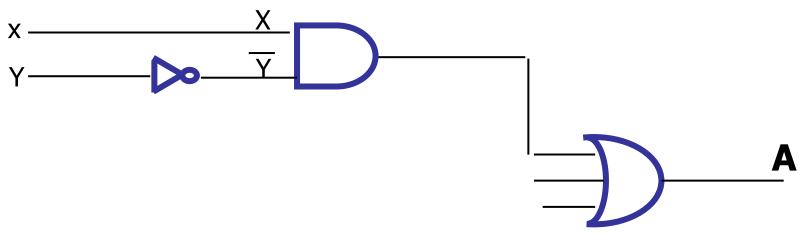


Fig 2

Figure 3, below adds an **AND** gate to form the **$\overline{\mathbf{X}}$.Y** input to the **OR** gate.

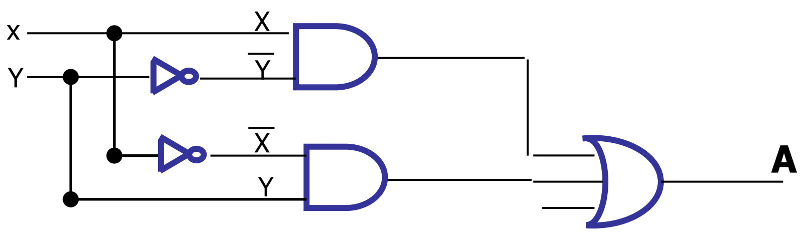


Fig 3

Finally fig 4 adds another **AND** gate to form the **X.Y** input

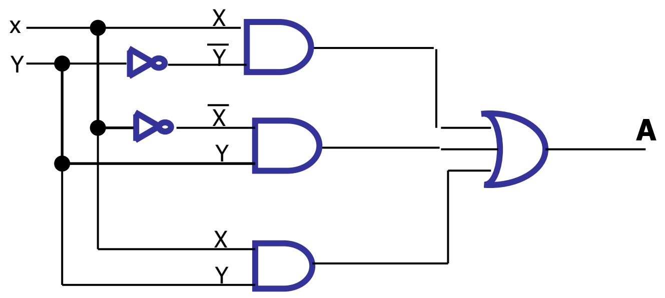


Fig 4.

Sum of Product and Product of Sum methods always produces a logical expression, but usually is not the simplest or minimum expression.

- By simplification or minimization we actually mean, optimizing some engineering criteria such as minimizing total number of gates, the delay, maximizing the reliability, etc.

Secondly the complexity of a given network has a direct impact on its cost. Because it is always desirable to reduce the cost of any manufactured product, it is important to find ways for implementing logic circuits as inexpensively as possible. It should be noted that a given function can be implemented with a number of different networks. Some of these networks are simpler than others, hence, searching for solutions that entail minimum cost is prudent.

Boolean Algebra

Boolean algebra provides is a powerful tool that can be used for designing and analyzing logic circuits. It provides the foundation for much of our modern digital technology. Boolean algebra is based on a set of rules that are derived from a small number of basic assumptions as shown below.

Single-Variable Theorems

1a. $x \cdot 0 = 0$
1b. $x + 1 = 1$
2a. $x \cdot 1 = x$
2b. $x + 0 = x$
3a. $x \cdot x = x$
3b. $x + x = x$
4a. $x \cdot \bar{x} = x$
4b. $x + \bar{x} = 1$
5. $\bar{\bar{x}} = x$

Two-and Three-Variable Properties

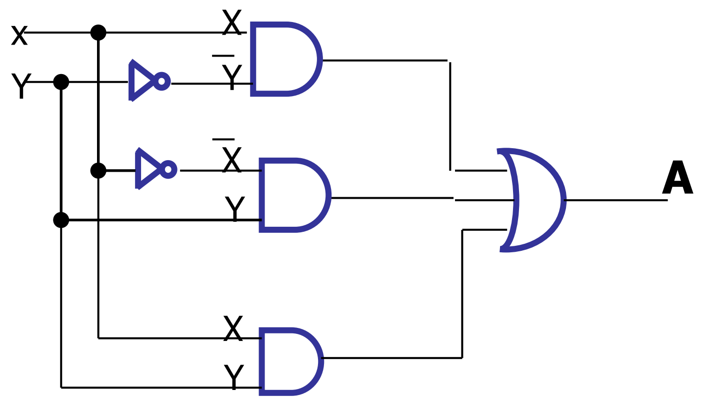
1a.	$x \cdot y = y \cdot x$	**Cumulative**
1b.	$x + y = y + x$	
2a.	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	**Associative**
2b.	$x + (y + z) = (x + y) + z$	
3a.	$x \cdot (y + z) = x \cdot y + x \cdot z$	**Distributive**
3b.	$x + y \cdot z = (x + y) \cdot (x + z)$	
4a.	$x + x \cdot y = x$	**Absorption**
4b.	$x \cdot (x + y) = x$	
5a.	$x \cdot y + x \cdot \bar{y} = x$	**Combining**
5b.	$(x + y) \cdot (x + \bar{y}) = x$	
6a.	$\overline{x \cdot y} = \bar{x} + \bar{y}$	**DeMorgan's Theorem**
6b.	$\overline{x + y} = \bar{x} \cdot \bar{y}$	
7a.	$x + \bar{x} \cdot y = x + y$	
7b.	$x \cdot (\bar{x} + y) = x \cdot y$	
8a.	$x \cdot y + y \cdot z + \bar{x} \cdot z = x \cdot y + \bar{x} \cdot z$	**Consensus**
8b.	$(x + y) \cdot (y + z) \cdot (\bar{x} + z) = (x + y) \cdot (\bar{x} + z)$	

CORRECTION
LINE 4a should read $\bar{X}.X = 0$

Boolean rules for simplification

Boolean algebra finds its most practical use in the simplification of logic circuits. If we translate a logic circuit's function into symbolic (Boolean) form, and apply certain algebraic rules to the resulting equation to reduce the number of terms and/or arithmetic operations, the simplified equation may be translated back into circuit form for a logic circuit performing the same function with fewer components. If equivalent function may be achieved with fewer components, the result will be increased reliability and decreased cost of manufacture.

Let us consider our derived circuit



fig

The circuit above is described by the function:

$$A = X.\overline{Y} + \overline{X}.Y + X.Y ------(1)$$

Applying the commutative law to the 2nd and 3rd term we can write the above as:

$$A = X.\overline{Y} + X.Y + \overline{X}.Y ------(2)$$

Using the distributive term,

$$A = X(\overline{Y} + Y) + \overline{X}.Y ------(3)$$

Theorem 4b,

$$A = X.1 + \overline{X}.Y ------(4)$$

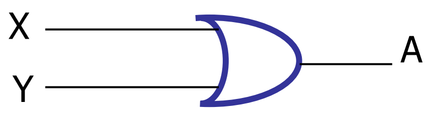
Theorem 2a

$$A = X + \overline{X}.Y ------(5)$$

DeMorgan,s theorem 7a

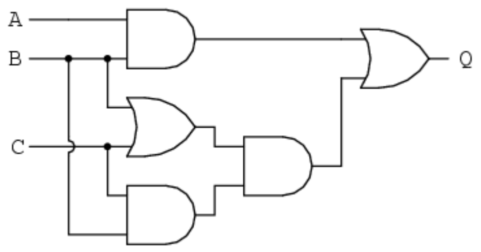
$$A = X + Y ------(6)$$

From equation 6, the problem stated in example 1 can be implemented with a simplified circuit shown below. This is much simpler than the original circuit and yet performs the same function.

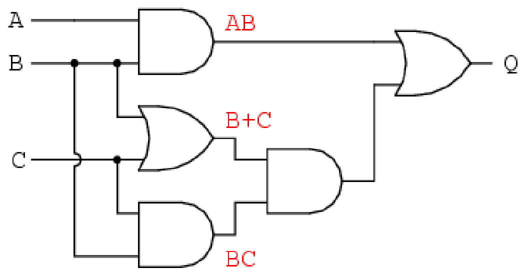


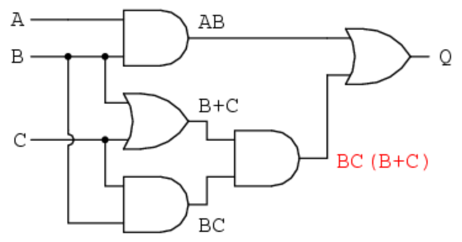
Example 2

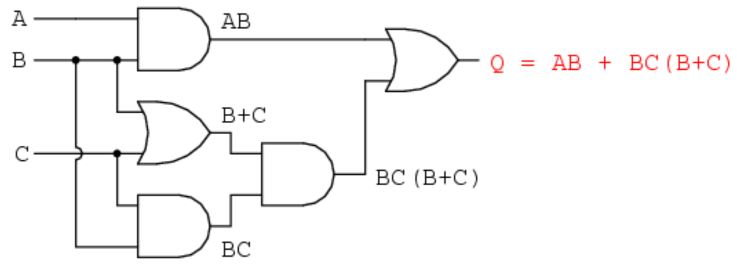
Use Boolean expressions to derive a simplified form of the circuit shown below:



Our first step in simplification must be to write a Boolean expression for this circuit. This task is easily performed step by step if we start by writing sub-expressions at the output of each gate, corresponding to the respective input signals for each gate. Remember that OR gates are equivalent to Boolean addition, while AND gates are equivalent to Boolean multiplication.







$$AB + BC(B + C)$$

↓ Distributing terms

$$AB + BBC + BCC$$

↓ Applying identity **AA = A** to 2nd and 3rd terms

$$AB + BC + BC$$

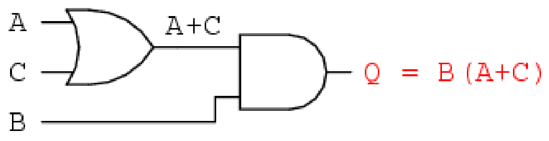
↓ Applying identity **A + A = A** to 2nd and 3rd terms

$$AB + BC$$

↓ Factoring **B** out of terms

$$B(A + C)$$

The simplified circuit is thus shown below:



The final expression, B(A + C), is much simpler than the original, yet performs the same function. If you would like to verify this, you may generate a truth table for both expressions and determine Q's status (the circuits' output) for all eight logic-state combinations of A, B, and C, for both circuits. The two truth tables should be identical.

SUMMARY

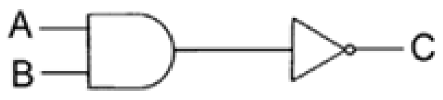
- A straight forward implementation of a function can be obtained by using a product term (AND gate) for each row of the truth table for which the function is equal to 1
- The sum of these products realizes the desired function.
- There are many different networks that can realize a given function. Some networks are simpler than others.
- Algebraic manipulation can be used to derive simplified logic expressions and thus lower cost networks
- Logic circuits found in complex systems, such as computers cannot be designed manually – they are designed using sophisticated CAD tools that automatically implement the synthesis technique

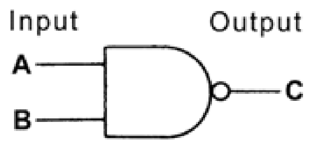
OTHER GATES DERIVED FROM AND, OR, & NOT

A circuit that utilizes more than 1 logic function has Combinational Logic. As an example, if a circuit has an AND gate connected to an Inverter gate, this circuit has combinational logic.

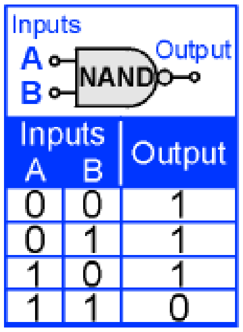
NAND Gate

A NAND gate is the combination of both an AND gate and a NOT gate. It operates on the same as an AND gate but the output will be the opposite. Remember the NOT gate does not always have to be on the output leg; it could be used to invert an Input signal also.





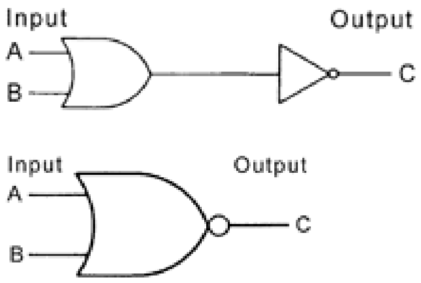
Truth Table for the NAND Gate


Inputs
A
B
NAND
Output

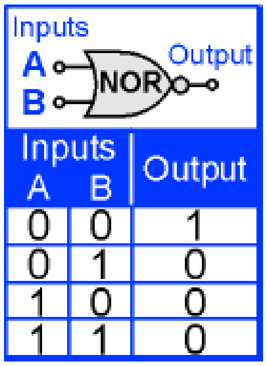
Inputs A	Inputs B	Output
0	0	1
0	1	1
1	0	1
1	1	0

The NOR Gate

A NOR gate is the combination of both an OR gate and a NOT gate. It operates the same as an OR gate, but the output will be the opposite.



The truth table for a NOR gate

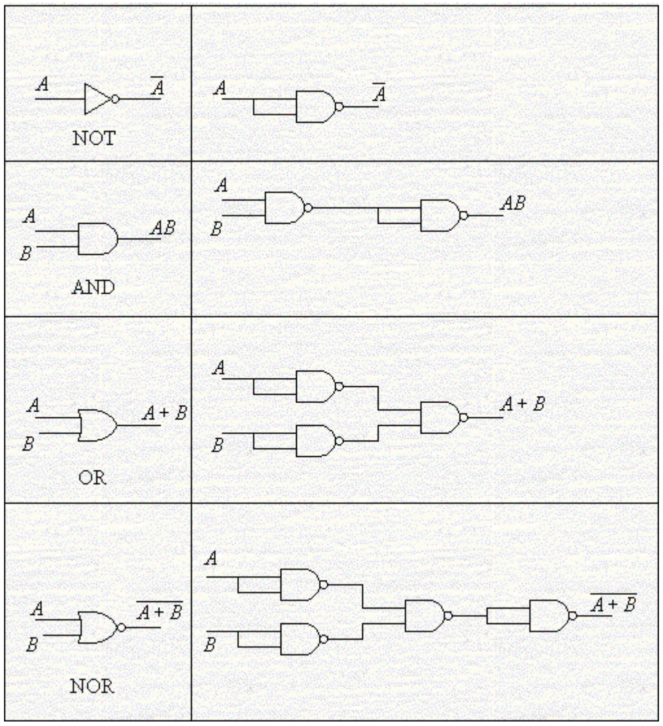

Inputs
A
B
NOR
Output

Inputs A	Inputs B	Output
0	0	1
0	1	0
1	0	0
1	1	0

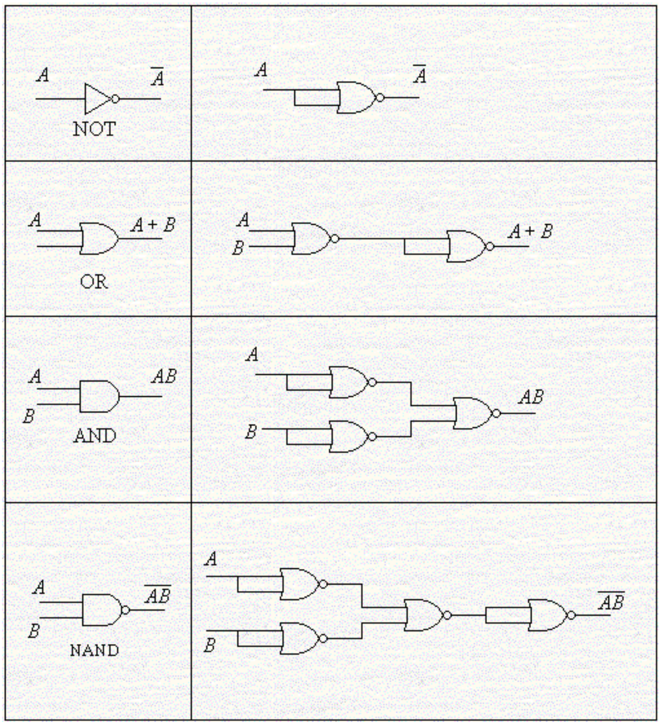
Universal Capability NAND and NOR

NAND and NOR gates are universal logic gates. The AND, OR, NOR and Inverter functions can all be performed using only NAND gates. Also, the AND, OR, NAND and Inverter functions can all be performed using only NOR gates. An inverter can be made from a NAND or a NOR by connecting all inputs of the gate together. If the output of a NAND gate is inverted, it becomes an AND function.

- Here are the NAND equivalents of the various logic gates:



- Here are the NOR equivalents of the various logic gates:



Simplification of circuits containing NAND and NOR gates

- To simplify circuits containing NAND and NOR gates we use this theorem
- De Morgan's theorem allows large bars in a Boolean Expression to be broken up into smaller bars over individual variables.
- De Morgan's theorem says that a large bar over several variables can be broken between the variables if the sign between the variables is changed.

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

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

- In order to reduce expressions with large bars, the bars must first be broken up. This means that in some cases, the first step in reducing an expression is to use De Morgan's theorem.

<u>Example 1</u>

Simplify the following Boolean expression

$$X = \overline{AB}.\left(B + C\right)$$

Applying De Morgan's theorem produces

$$X = \left(\overline{A} + \overline{B}\right)\left(B + C\right)$$
$$X = \overline{A}B + \overline{A}C + \overline{B}B + \overline{B}C$$
$$X = \overline{A}B + \overline{A}C + \overline{B}C$$

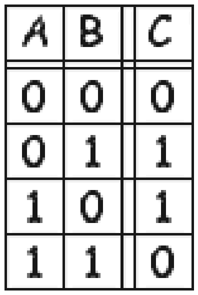
<u>Example 2</u>

$$X = \overline{\left(AB.\overline{C + D}\right).AB}$$
$$X = \overline{AB.\overline{C + D}} + \overline{AB}$$
$$X = \overline{AB} + \overline{\overline{C + D}} + \overline{AB}$$
$$X = \overline{A} + \overline{B} + C + D + \overline{A} + \overline{B}$$
$$X = \overline{A} + \overline{B} + C + D$$

The Exclusive-OR, or XOR Gate

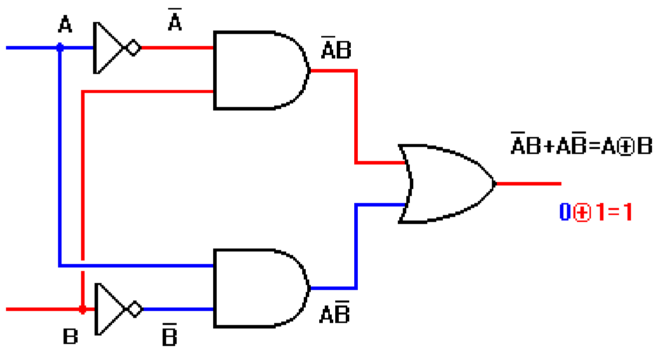
The Exclusive-OR, or XOR function is an interesting and useful variation on the basic OR function. Verbally, it can be stated as, "Either A or B, but not both." The XOR gate produces a logic 1 output only if its two inputs are *different*. If the inputs are the same, the output is a logic 0.

The truth table for the XOR

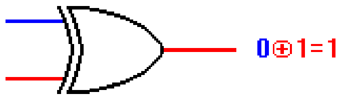


A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

A circuit which implements the above functionality is shown below:



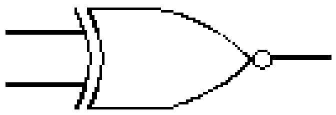
The Exclusive-OR, or XOR function can be described verbally as, "Either A or B, but not both." The circuit symbol for the XOR gate is shown below



The XOR symbol is a variation on the standard OR symbol. It consists of a plus (+) sign with a circle around it. Unlike standard OR/NOR and AND/NAND functions, the XOR function always has exactly two inputs.

The XNOR (exclusive-NOR) gate

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same, and "false" if the inputs are different.



Symbol of XNOR

Boolean expression:

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

Summary

- Circuits called logic gates are the basic building blocks of computers and almost all digital systems.
- The fundamental logic gates are called AND, OR, NAND, and NOR.