## DEPARTMENT OF COMPUTER SCIENCE AND MATHEMTICS

## COLLEGE OF BASIC AND APPLIED SCIENCES

# COURSE OUTLINE

**CSC 205: Introduction to Digital Logic & Computer Hardware (3 Units: LH 30, PH 45)**

**Semester:** First Semester 2024/2025 Session, October, 2024 – February, 2025

**Lecturer:** Olumoye Mosud, PhD (Associate Professor)

**Office:** Department of Computer Science,
College of Basic and Applied Sciences (CBAS),
E:mail: yomosud@mtu.edu.ng & myolumoye@yahoo.com

**Course Description**
**CSC 205: Introduction to Digital Logic & Computer Hardware (3 Units: LH 30, PH 45)**
Binary arithmetic operations: addition, subtraction, multiplication, division. Binary point, floating point. Basic logic gates, symbols, truth tables. Boolean Algebra; Theorems, Minimization methods, Karnaugh maps (up to to six variables) etc. Computer circuits; diode arrays, PIAs etc, Integrated circuits fabrication process. Use of MSI, LSI and VLSI IC' hardware Design. Primary and Secondary memories; core memory, etc. Magnetic devices; disks, tapes, video disks etc. Peripheral devices; printers, CRT's, keyboards, character recognition.

**Recommended Texts:**

**Course Performance Criteria:**
***Continuous Assessment:***

| | |
|---|---|
| a. (i) Mid-semester Test | % |
| (ii) Assignments | % |
| b. Final Examinations | % |
| Total | 100% |

**Course Outline: Nov, 2023 – February, 2024**

| Week | Day | Topic | Assignment | Reference |
|---|---|---|---|---|
| 1 | | | | Listed above |
| 2 | | | No 1 | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | No 2 | |
| 10 | | | | |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |

# CHAPTER ONE
# BINARY ARITHMETIC OPERATIONS

## 1.1 INTRODUCTION

The decimal number system is the common number system we use in our daily transactions and other forms of record keeping. This number system is also referred to as base ten and it has ten digits of numbers ranging from 0 to 9 (i.e. 0,1,2,3,4,5,6,7,8 & 9).

Since computers are electronics devices, they cannot store numbers and letters directly. They can only store digital electrical signal which are either charged or discharged that is, in one of the two possible states. And the storage locations in the computer system are divided into cells with each cell capable of storing one ***BI***nary digi***T*** (BIT) which can either be 0 or 1. Because of this convenience of representing data in 0 (off) and 1 (on) state any character that has to be stored must be identified in the patterns of 0's and 1's.

Other base numbers are ternary (base 3), quaternary (base 4), quinary (base 5), octal (base 8), duodecimal (base 12) and hexadecimal (base 16). But the relevant base numbers to the concept of data representation in computer are octal, decimal (denary) and hexadecimal.

## 1.2 BINARY NUMBER SYSTEM

Binary numbers are fundamentals to computer since storage in it is based on the two states concept. The base or radix of the binary number system is 2 and is composed of two digits 0 and 1.

### 1.2.1 Conversion form Binary to Decimal

The procedure for converting binary to decimal can be termed positional. Firstly, the first digit from the right-hand side is given a zero position and the next given 1, 2, 3, respectively depending on the numbers of digits involved. Secondly, the base number which is 2 is raised to the power of the position multiplied by the digit of the binary number. Thirdly, the products in the stage are now added together to get the decimal equivalent of the binary number.

**Example 1**

Convert $(10010110)_{two}$ to decimal

**Solution:**

| | | |
|---|---|---|
| Step 1: | Position: | 7 6 5 4 3 2 1 0 |
| | Digits: | 1 0 0 1 0 1 1 0 |

Step 2: $(1 * 2^7) + (0 * 2^6) + (0 * 2^5) + (1 * 2^4) + (0 * 2^3) + (1 * 2^2) + (1 * 2^1) + (0 * 2^0)$

Step 3: $= 128 + 0 + 0 + 16 + 0 + 4 + 2 + 0 = 150_{ten}$

So, $(100100110)_{two} = 150_{ten}$

Another method is the one that follows:
$(10010110)_{two}$

```
=          1     0        0        1        0          1      1       0
          *2
           2 + 0 = 2
                  * 2
                   4 +   0 = 4
                            *2
                             8 + 1 = 9
                                     * 2
                                    18 + 0 = 18
                                              *2
                                              36 + 1 = 37
                                                         *2
                                                         74 + 1 = 75
                                                                    *2
                                                                 150 + 0  = 150 ten
```

So, $(10010110)_{two} = 150_{ten}$

**Example 2**
Convert $(1111001)_2$ to decimal

**Solution:**

| | | |
|---|---|---|
| Step 1: | Position: | 6 5 4 3 2 1 0 |
| | Digits: | 1 1 1 1 0 0 1 |

Step 2: $(1 * 2^6) + (1 * 2^5) + (1 * 2^4) + (1 * 2^3) + (0 * 2^2) + (0 * 2^1) + (1 * 2^0)$

Step 3: $64 + 32 + 16 + 8 + 0 + 0 + 1 = (121)_{ten}$

### 1.2.2 Conversion from Decimal to Binary

This conversion is done by continuously dividing the given number by two (binary) and noting the remainder. The division continues until the quotient is zero. The binary equivalent of the decimal number will be the list of the remainder starting from the last to the top as shown in the example.

**Example 3**
Convert $(150)_{ten}$ to a number in binary

**Solution:**

| Target Base | Quotient | Remainder R |
|---|---|---|
| 2 | 150 | |
| 2 | 75 | 0 |
| 2 | 37 | 1 |
| 2 | 18 | 1 |
| 2 | 9 | 0 |
| 2 | 4 | 1 |
| 2 | 2 | 0 |
| 2 | 1 | 0 |
| 2 | 0 | 1 |

From the last to the top we have 10010110
So, $150_{ten}$ = $1001\ 0110_{two}$

**Example 4**
Convert $874_{10}$ to binary

**Solution:**

| Target Base | Quotient | Remainder R |
|---|---|---|
| 2 | 874 | |
| 2 | 437 | 1 |
| 2 | 218 | 1 |
| 2 | 109 | 0 |
| 2 | 54 | 1 |
| 2 | 27 | 0 |
| 2 | 13 | 1 |
| 2 | 6 | 1 |
| 2 | 3 | 1 |
| 2 | 1 | 1 |
| | 0 | 1 |

From the last to the top 1111101011
So, $874_{10}$ = $1111101011_{two}$

### 1.2.3 Binary Addition

The addition of numbers in the binary system is similar to decimal addition and are made up in bit patterns of 1's and 0's. After addition the result will still be 1's and 0'.

**Example 5**
Add $111101_2$ and $001101_2$ together

**Solution:**

| Binary | Decimal |
|---|---|
| 1 1 1 1 0 1 | 6 1 |
| + 0 0 1 1 0 1 | +1 3 |
| $1\ 0\ 0\ 1\ 0\ 1\ 0_2$ | $7\ 4_{10}$ |

**Example 6**
Add 1 0 1 1 0 0 and 1 0 0 1 1 using binary system.
**Solution:**

| Binary | Decimal |
|---|---|
| 1 0 1 1 0 0 | 4 4 |
| + 1 0 0 1 1 | 1 9 |
| $1\ 1\ 1\ 1\ 1\ 1_2$ | $6\ 3\ _{10}$ |

### 1.2.4 Binary Subtraction
The subtraction in binary is similar to that of decimal number system.

**Example 7**
Subtract 1111 base two from 101101 base two.

**Solution:**

| Binary | Decimal |
|---|---|
| 101101 | 59 |
| - 1111 | - 14 |
| 11111 | 31 |

**Example 8**
Subtract 10101 from 111011 all in binary number.

**Solution:**

| Binary | Decimal |
|---|---|
| 111011 | 59 |
| - 10101 | - 21 |
| 100110 | 38 |

But when you are subtracting a larger number from a smaller number, the result you will get will be the 2's complement. This implies that instead of getting a negative number, we get the 2's complement of the negative number.

**Example 9**
Subtract 101101 from 1111 all in binary

**Solution:**

```
                Binary              Decimal
                001111                15
               -101101               -45
Ignore Carry  ⟶ 1                    -30
                 00010
                    10
```

In this binary subtraction, instead of getting -11110 (-30) we get the 2's complement of 11110, which is 10.

**Complements**

The term complement is used by computers to perform subtraction sand to represent negative numbers. There are two types of complements that are used in binary subtraction: The 1's complement and 2's complement. Also, there are two types of complements in decimal or denary systems which are 9's complement and 10's complement. The reason why complement is used is it is convenient and efficient with two state devices.

**1. One's Complement**

The 1's complement of a binary number is referred to 9's complement in the decimal system. In order to obtain the 1's complement of a number, each bit is subtracted from 1. For example, the 1's complement of 10011 is 0110 while that of 010110 is 101001. One's complement is also called **radix-minus-one complement** (or diminished radix complement).

One of the easiest methods of subtracting binary numbers from each other is the **One's Complemental Subtraction**: The rules for using this method is as follows:

(i) First of all, compute the **subtrahend** by subtracting each bit from 1.
(ii) Add this complement to the **minuend.**
(iii) Perform **end-around carry** of the last bit 0 or 1.
(iv) If there is no end-around carry that is the bit is 0, then the answer must be recoplemented and a negative sign is attached to it.
(v) And if the end-around carries is 1, then no need for recomplementing. You only add the end-around carry to the normal digits.

**Example 10**

Using the 9's complement subtracts 30964 from 999999 and find its 10's complement.

**Solution:**

```
   999999
 - 030964
   969035          9' complement
 +      1          Add 1
   969036          10's complement
```

**Example 11**
Using 1′s complements subtract the binary number 110001 from 011011. Also find the two′s complement.

**Solution**:
The question can be rephrased to 010111 – $110000)_2$

$$\begin{array}{rl}
011011 & \text{Minuend} \\
+\underline{001110} & \text{1′s compliment of subtrahend 110001} \\
\underline{100101} &
\end{array}$$

As seen, there is no end-around carry, so following rule (iv), the answer must be recomplemented to obtain 011010 and a negative sign attached.
The two′s complement can be obtained by adding 1 to the 1′s complement.

**Example 12**
Using one′s complements subtract 1011 from 1110 all in binary.

**Solution:**

$$\begin{array}{rl}
1\,1\,1\,0 & \text{Minuend} \\
\underline{+\,0\,1\,0\,0} & \text{1′s complement of subtrahend 1011} \\
\underline{\mathbf{1}\,0\,0\,1\,0} & \\
\underline{+\ \mathbf{1}} & \\
\underline{0\,0\,1\,1} &
\end{array}$$

## 2. Two′s Complement
The 2′s complement in binary system is called the 10′s complement in the decimal system. In order to get 2′s complement you add 1 to 1′s complement. i.e. **2′s complement = 1′s complement + 1**. Two′s complement is also referred to as radix complement (true or noughts complement).

Two′s complement subtraction can be used to solve the problem of two′s complement and the procedures involved are as follows:
(i) First of all find the 2′s complement of the subtrahend.
(ii) Add this complement to the minuend.
(iii) Drop the final carry if it is 1, then the answer is positive and does not need to be recomplemented.
(iv) If there is no carry, you first of all subtract 1 from the answer then recomplement the result obtained and attach minus sign to obtain the final answer.

**Example 13**
Using two′s complement subtract the following binary numbers from each other:
(a) 110101 from 011001
(b) 1011 from 1110

**Solution**:

(a) The question can be rephrased to 011001 - 110101

The 1's complement of subtrahend 110101 is 001010

1st rule: The 2's complement will be = 001010 + 1 = 001011

2nd rule: Add 011001 to 001011 as shown:

$$\begin{array}{rl} 0\,1\,1\,0\,0\,1 & \text{Minuend}\\ +\underline{0\,0\,1\,0\,1\,1} & \text{2's complement of subtrahend of 110101}\\ \underline{1\,1\,0\,1\,0\,0} & \end{array}$$

3rd rule: This is not applicable since there is no final carry.

4th rule: Since there is no carry, first of all subtract 1 from 110100 which gives you 110011. Then recomplement it and attach minus sign to give the final answer which is equal to **-001100**.

(b) The 1's complement of subtrahend 1011 is 0100

1st rule: the 2's complement = 0100 + 1 = 0101

2nd rule: Add 1110 + 0101 as shown:

$$\begin{array}{rl} 1\,1\,1\,0 & \text{Minuend}\\ \underline{+\,0\,1\,0\,1} & \text{2's complement of subtrahend of 1011}\\ \underline{\mathbf{1}\,1\,0\,1\,1} & \end{array}$$

3rd rule: From the third rule you drop the final carry if it is 1, then the answer is positive and does not need to be recomplemented

So, the final answer is $1011_{two}$

### 1.2.5 Binary Multiplication

The multiplication in binary is similar to that of the denary. Two products can be multiplied by partial products and the sum of the partial products gives the final answer.

**Example 14**

Find the product of these two binary numbers 101 and 110.

**Solution:**

$$\begin{array}{r} 1\;0\;1\\ \underline{*\,1\;1\;0}\\ 0\;0\;0\\ +\,1\;0\;1\phantom{\;0}\\ \underline{1\;0\;1\phantom{\;0\;0}}\\ \underline{1\;1\;1\;1\;0} \end{array}$$

Answer = 11110

### 1.2.6 Binary Division

Binary division is also of the same kind to the division in decimal numbers.

**Example 15**

Divide 11001 by 101 all in binary numbers.

**Solution**:
The divisor is $111_2$ while the dividend is $11001_2$. The procedure for doing this is the same as that of the decimal division.

```
          1 1 0
        ____________
  101  | 1 1 0 0 1
       | 1 0 1
         -----
         0 1 1 0
           1 0 1
           -----
            0 1 1
```

The quotient of 11001 divide by 110 is $110_2$ while the remainder is $11_2$.

## 1.4 CODING SYSTEM

The manufacturers of computers and data processing equipment have different views about the ideal number system to use. In our daily business transaction, the decimal number system is preferred, and input and output of data are typically decimal. On the other hand, the computer designer prefers a binary system because it is much easier to design and build a machine with components that can maintain two stable states than ten. The method used in representing data in computer is known as a **code** or **coding system**. The most commonly used codes are:

**1. Binary Coded Decimal (BCD):** This is a 4-bit code used for coding numeric values only and this 4-bit is used to represent a decimal number. For example, 3 is represented by 0011. And if a decimal number has more than one digit such as 25, each of the digits is represented individually by its equivalent 4 bits binary code. The BCD equivalent of 25 will now be 0010 0101.

**2. Standard BCD 6-Bit Code**: In other to allow the non-numeric characters to be coded on the BCD 4-bit code, it is extended by two bits.

**3. Extended Binary Coded Decimal Interchange Code (EBCDIC**): This code is often pronounced as **"Eb-See Dick**" and sometimes called "8-bit ASCII". It is an 8-bit code without parity, 9th bit can be used for parity. With 8 bits up to 256 characters can be coded. EBCDIC has code that can represent every number, alphabetic characters or special characters.

**4. American Standard Code for Information Interchange (ASCII)**: ASCII is a 7-bit code with the 8th bit as the parity bit. The code was developed by the American National Standards Institute (ANSI) to provide a standard code that could be used by many different computer manufacturers in order to make them compatible.

**Revision Questions**
1. (a) What do you understand by the term coding system?
(b) Briefly explain the following coding systems:
(i) ASCII Code (ii) Binary Coded Decimal (iii) EBCDIC

(c) Differentiate between binary and floating-point systems,

2. Using two's complement, subtract the following binary numbers from each other:
(a) 110101 from 011001 (b) 1011 from 1110

3. Briefly discuss the following the number systems:
(a) Binary Number System (b) Octal Number System (c) Hexadecimal Number System

4. Briefly describe the usefulness of the following terms in modern day computers:
(a) Check digit (b) Parity bit (C) Differentiate between a computer structure and its function.

# CHAPTER TWO
# LOGIC GATES

## 2.1 INTRODUCTION

This chapter introduces some basic concepts that the reader needs to be familiar with in order to understand the simplest form of electronic logic. The chapter further describes the meaning of logic gates, the various types and construction of simple truth tables.

## 2.2 HISTORY AND DEVELOPMENT OF LOGIC GATES

The initial logic gates were made mechanically. Charles Babbage, around 1837, devised the Analytical Engine. The logic gates which he used for his engine relied on mechanical gearing for performing its operations. Electromagnetic relays were later used for logic gates. In 1891, Almon Strowger patented a device containing a logic gate switch circuit (U.S. Patent ), which became widely used in 1920s. In 1898, Nikola Tesla filed for patents of devices containing logic gate circuits.

Eventually, vacuum tubes replaced relays for logic operations. In 1907, Lee De Forest's modified the vacuum tube to Fleming valve which can be used as AND logic gate. Ludwig Wittgenstein introduced a version of the 16-row truth table in logic gates and hardware, as a proposal of Tractatus Logico-Philosophicus in 1921. Claude E. Shannon introduced the use of Boolean algebra in the analysis and design of switching circuits in 1937. Walther Bothe, inventor of the coincidence circuit, got part of the 1954 Nobel Prize in physics, for the first modern electronic AND gate in 1924. Active research is taking place in molecular logic gates.

## 2.3 MEANING OF LOGIC GATES

**Digital logic gate**, which is also referred to as **combinational logic gate** or simply **logic gate** is elementary building block of a digital integrated circuit (IC) whose output at any time is determined by the states of its inputs at that time. Also, logic gate is a type of circuit (or collection of transistors and resistors) that regulates the flow of electricity or optical signals in fiber optic computing system which the Boolean logic computer uses to make complex logical decisions.

A logic gate takes one or more logic-level inputs (binary form) and produces a single logic-level output of 1 or 0. Because the output is also a logic level an output of one logic gate can connect to the input of one or more other logic gates. With some types of logic gates such as those with open-collector outputs, two outputs may be wired to produce what is known as a **wired OR**.

*The primary application of logic gates is to implement* ***logic*** *in the flow of digital signals in a digital circuit. The term logic in its ordinary sense is defined as a branch of philosophy that deals with what is true and false, based on whether other things are true and false. The fundamental function of logic gates in digital circuits is to determine which outputs will be true or false, given a set of inputs that can either be true (****logic 1****) or false (****logic 0****).*

The earliest logic gates were mechanically made. But primarily logic gates are electronically implemented using diodes or transistors and can also be constructed using electromagnetic relays, fluidics, optics, or even mechanical elements. The simplest form of electronic logic is diode logic. This allows AND and OR gates to be built, but not inverters, and so is an incomplete form of logic. In order to build a complete logic system, valves or transistors can be used. The simplest family of logic gates using bipolar transistors is called **resistor-transistor logic** or **RTL**. For higher speed, the resistors used in RTL were replaced by diodes leading to **diode-transistor logic**, or **DTL**. Later, it was discovered that one transistor could do the job of one diode, so **transistor-transistor** logic or **TTL** was created. To further reduce the size and power consumption in some types of chips, the bipolar were replaced with complimentary field-effect transistors (**MOSFETs**), thereby resulting in complimentary metal-oxide-semiconductor (**CMOS**) logic.

## 2.4 LOGIC GATE SYMBOLS

There are two series of symbols for logic gates:

1. The **traditional symbols** have distinctive shapes which make them to be easily recognized. So they are widely used in industry and education.

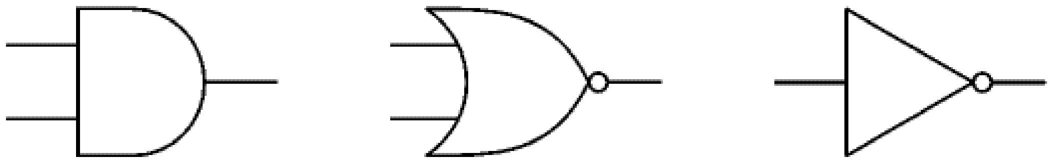


**Figure 2.1 Traditional Symbols of Logic Gates**

2. The **IEC (International Electro-technical Commission) symbols** are rectangles with a symbol inside to show the gate function. They are rarely used despite their official status, but one may need to know them for examination purpose.

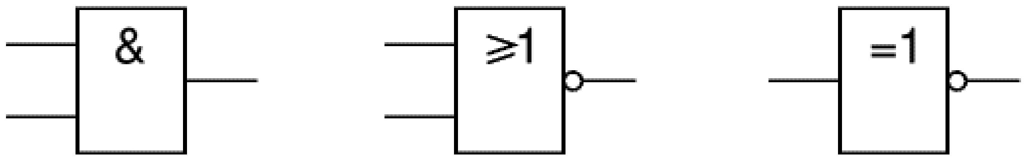


**Figure 2.2 IEC Symbols of Logic Gates**

## 2.5 TYPES OF LOGIC GATES

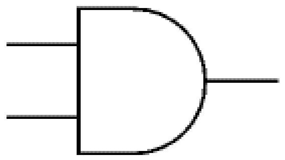
There are seven types of logic gates namely: **AND**, **OR**, **NOT**, **XOR** (exclusive -OR), **NAND** (NOT-AND), **NOR** (NOT-OR), and **XNOR** (exclusive NOR). Computer uses these gates to make complex decision making process.

Three of these gates which are **AND**, **OR** and **NOT** are referred to as **simple gates**, because they combine to perform complex decision-making process. The **complex logic gates** are **XOR**, **NAND**, **NOR** and **XNOR**. Among all these gates NAND and NOR are the pillars of logic in that all other types of Boolean logic gates (i.e. AND, OR, NOT, XOR and XNOR) can be created from a suitable network of just NAND or NOR gate. Hence, these gates NAND and NOR gates are called ***Universal Gates***. The term Boolean as used in this context is like the common algebra, which deals with manipulation of expressions to solve or simplify equations, while the expressions used in Boolean algebra are called Boolean expressions. The details about the seven types of logic gates are as follows:

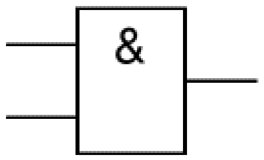
## 1. AND Gate

The AND gate is a logic gate which will give a high output if and only if all its inputs are high. It symbolizes logical multiplication. The AND gate is so called because if **0** is **false** and **1** is **true**, the gate acts in the same way as the logical **'and'** operator. This can further be described as a logic gate that gives an output of '**1**' only when all of its inputs are '1'. Thus, its output is '0' whenever at least one of its inputs is '0'. Mathematically, $Q = A \cdot B$.

Also, the output Q is true if input A **AND** input B are both true. An AND gate can have two or more inputs, its output is true if all inputs are true. AND gate can be represented with either the traditional or IEC symbol with its truth table shown in Figure 3.



**Traditional symbol**



**IEC symbol**

| Input A | Input B | Output, Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

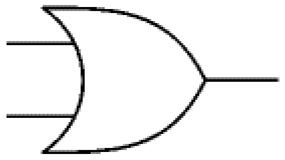
**Truth Table**

**Figure 2.3: AND Gate Symbols**

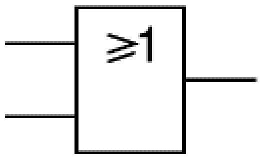
## 2. OR Gate

The OR gate is a gate which produces a high output of any or both (all) if its inputs are high. It symbolizes logic addition. The ***OR gate*** got its name from the fact that it behaves after the fashion of the logical inclusive "or." The OR gate gives an output which is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false". Thus, its output is '1' whenever at least one of its inputs is '1'. Mathematically, Q = A + B (or Q = A OR B).

An OR gate can have two or more inputs; its output is true if at least one input is true. An OR gate can be represented with either a traditional or IEC symbol as shown in figure 3.4 with the truth table.



**Traditional Symbol**



**IEC Symbol**

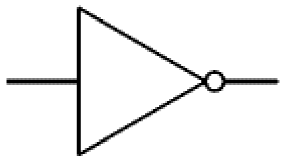
| Input A | Input B | Output, Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Truth Table**

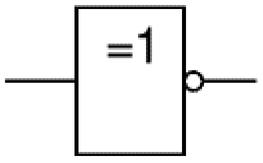
**Figure 2.4: OR Gate Symbols**

**3. NOT Gate (Inverter)**

The NOT gate which is also called an **inverter** is a logic gate that gives an output that is opposite to the state of its input. A NOT gate can only have one input while the output will be the opposite of the input. In electronics a NOT gate is more commonly called an **inverter**. The circle on the symbol is called a **bubble**, and is generally used in circuit diagrams to indicate an inverted input or output. Mathematically**, Q = A**.



**Traditional Symbol**



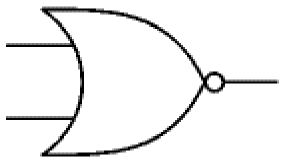
**IEC Symbol**

| Input A | Output, Q |
|---|---|
| 0 | 1 |
| 1 | 0 |

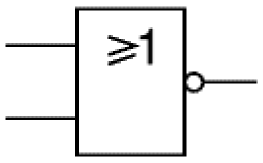
**Truth Table**
**Figure 2.5: NOT Gate Symbols**

**4. NOR Gate (NOR = N**ot **OR)**

The NOR gate is an **OR** with a **NOT** gate at its end. Consequently, for the same combination of inputs, the output of a NOR gate will be the opposite of that of an OR gate. Also, a NOR gate can be described as an OR gate with inverted output. A NOR gate can have two or more inputs and its output is true if no inputs are true. Mathematically, **Q = NOT (A OR B)**.



**Traditional Symbol**



**IEC Symbol**

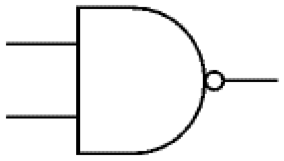
| Input A | Input B | Output, Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Truth Table**

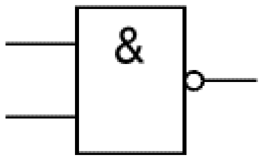
**Figure 2.6: NOR Gate Symbols**

5. **NAND Gate (NAND = N**ot **AND)**

The NAND gate is an AND gate with the output inverted or NOT gate at its end as shown by the '**o**' on the output. The output is true if input A AND input B are NOT both true. A NAND gate can have two or more inputs; its output is if NOT all inputs are true. Mathematically, **Q = NOT (A AND B)**



**Traditional Symbol**



**IEC Symbol**

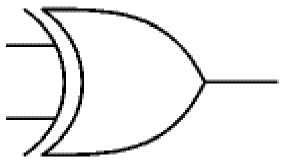
| Input A | Input B | Output, Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Truth Table**

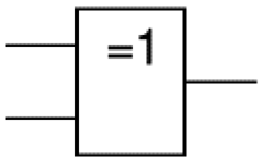
**Figure 2.7: NAND Gate Symbols**

**6. EX-OR Gate (EX**clusive**-OR)**

The EXOR gate is a logic gate that gives an output of 1 when only one of its inputs is 1. It is like an OR gate but excludes both inputs being true. That is to say the output is true if inputs A and B are different. EX-OR gate scan only have two inputs. Mathematically, **Q = (A AND NOT B) OR (B AND NOT A)**



**Traditional Symbol**



**IEC Symbol**

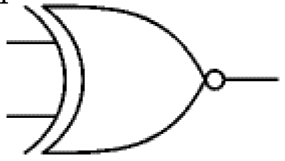
| Input A | Input B | Output, Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Truth Table**

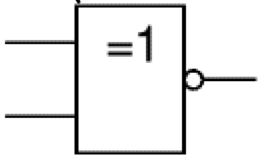
**Figure 2.8: EX-OR Gate Symbols**

**7. XNOR Gate EX-NOR (EXclusive-NOR) gate**

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



**Traditional Symbol**



**IEC Symbol**

| Input A | Input B | Output, Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Truth Table**

**Figure 2.9: XNOR Gate Symbols**

## 2.6 INPUTS AND OUTPUTS OF A LOGIC GATE

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

Most books do not distinguish between an $AND_2$ gate and an $AND_3$ gate. They claim an AND gate is an AND gate, regardless of the number of inputs. While this is actually true, I subscript it but an $AND_2$ and an $AND_3$ do not have the same truth table. In particular, an $AND_2$ truth table has 4 rows while an $AND_3$ has 8. While the two truth tables are related, they still define different functions.

Also, some gate symbols have a circle on their output which means that their functions include inverting the output. This is equivalent to feeding the output through a NOT gate. For example, the NAND (Not AND) gate symbol shown on the right is the same as an AND gate symbol but with the addition of an inverting circle on the output.

## 2.7 TRUTH TABLES

A truth table is a table that describes the behaviours or functions of a logic gate. It lists the value of the output for every possible combination of the inputs states. The symbols 0 (false) and 1 (true) are usually used in truth tables. The figure that follows is an example of a truth table that shows the inputs and output of an AND gate.

| **Input A** | **Input B** | **Output, Q** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*The figure below summarizes the output states for all types of 2-input and 3-input gates, which can be useful if one is trying to select a suitable gate for two or three inputs.*

| ummary for all 2-input gates | | | | | | | |
|---|---|---|---|---|---|---|---|
| Inputs | | Output of each gate | | | | | |
| A | B | AND | NAND | OR | NOR | EX-OR | EX-NOR |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

Note that EX-OR and EX-NOR gates can only have 2 inputs.

| **Summary for all 3-input gates** | | | | | | |
|---|---|---|---|---|---|---|
| Inputs | | | Output of each gate | | | |
| A | B | C | AND | NAND | OR | NOR |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

However, because logic gates are physical, they have some characteristics that are not expressed in the truth tables. In particular, **gate delay**; this describes the amount of time it takes for the output to change when the input changes. This time is not zero, thus, one must wait a while for the output to take effect. Real gates have delay. For example, if one changes the values of the inputs, say from 0 and 0 to 0 and 1, then the output takes some small amount of time before it changes. This delay is called **gate delay**. The delay is due to the fact that information can travel at the speed of light, but in reality, the time taken for computation is not considerably fast. This delay limits the speed at which the inputs can change and still give the output a meaningful value.

2.8 Boolean Algebra; Theorems, Minimization methods, Karnaugh maps

**REVIEW QUESTIONS**

1 (a) What do you understand by a logic gate?
(b) List and explain the seven types of logic gates you know

2. (a) Briefly describe the term digital logic gates
(b) What is a truth table?
(c) Draw a truth table for the output states of 2-input and 3-input of an AND gates.

3. Briefly explain the following logic gates and draw their truth table:
(i) AND gates (ii) OR gates (iii) NOR gates
(iv) NAND gates (v) XOR gates

4. (a). Briefly describe the term digital logic gates
(b). Identify all the universal logic gates. State the reason why they are called universal gates.
(c). i). Concisely explain the following logic gates: AND, OR, and NOR
(ii). Draw a truth table for the output states of 2-input of an AND, OR and NOR gates.