LOGIC GATES

As discussed earlier, everything in the digital world is based on the binary number system. Numerically, this involves only two symbols — 0 and 1. According to the need, we can use these symbols or we can equate them with others. Thus, when dealing with digital logic, we can specify that:

0 = False = No
1 = True = Yes

Using this two-valued logic system, either every statement or condition must be 'true' or 'false'. It cannot be partly true and partly false.

Thus, a *logic gate* is an elementary building block of a digital circuit. A simple logic gate has two inputs and one output. At any given moment, every terminal is in one of the two binary conditions, low (0) or high (1), represented 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). In other words, a logic gate is an electronic circuit, which operates on one or more input signals to produce the desired output signals. Besides, the properties of a logic gate, or of a combination of gates, may be defined and presented in the form of a diagram called a **truth table**, which lists the output that will be triggered by each of the possible combinations of input signals.

2.8.1 Basic Logic Gates

Information is fed to a gate in the form of binary coded input signals (logic value 0 stands for 'OFF' or 'low-voltage pulse' and logic 1 for 'ON' or 'high-voltage'). By using different combination of the input signals and the output signals, different types of logic gates are made and are used to perform the necessary operation of the computer circuits. Essentially, there are three basic logic gates:

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

AND gate The AND gate is composed of two or more inputs and a single output and performs logical multiplication. The standard symbol for the AND gate is shown in Figure 2.1 and its truth table is listed in Table 2.10. The logical operation of the AND gate is such that the output is HIGH (1) when all the inputs are HIGH, otherwise it is LOW (0). A dot (.) is used to show the AND operation.

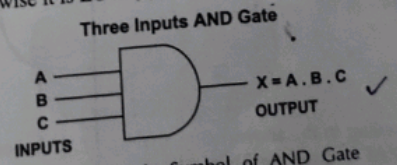


Figure 2.1 Logic Symbol of AND Gate

The term logic gate is used to describe the set of basic electronic components which when combined with each other, are able to perform complex logical and arithmetic operations.

Table 2.10 Truth table of AND Gate

A	B	C	X = A . B . C
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

The Venn diagram shown in Figure 2.2 provides an insight into the AND function. The highlighted area represents the function X = ABC.

Note: An AND gate will yield a logic 1 output only if it receives a logic 1 signal through all its inputs.

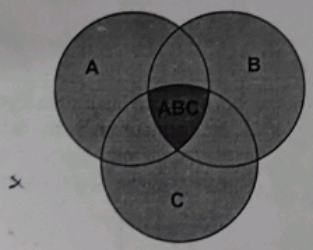
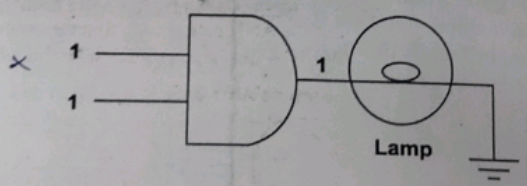


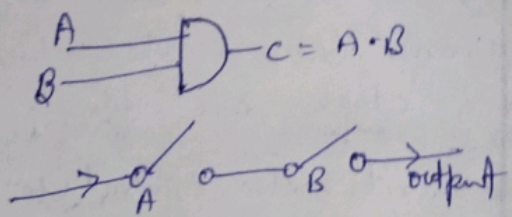
Figure 2.2 Venn Diagram illustrating AND Gate

We can explain the function of the AND gate with the help of an example.



According to the above figure, a value of '1' is needed at all AND gate inputs to produce an output value of '1' from the AND gate.

AND



Input		output
A	B	C = A · B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate The OR gate is composed of two or more inputs and a single output and performs logical addition. The standard symbol for the OR gate is shown in Figure 2.3 and its truth table is listed in Table 2.11. The logical operation of the OR gate is such that the output is HIGH (1) when any of the inputs are HIGH; otherwise it is LOW (0). A plus (+) is used to show the OR operation. In other words, the OR gate is an electronic circuit that gives a high output if one or more of its inputs are high.

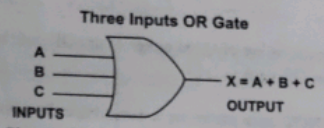


Figure 2.3 Logic Symbol of OR Gate

Table 2.11 Truth Table of OR Gate

A	B	C	X = A + B + C
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

The venn diagram shown in Figure 2.4 provides function of the OR gate. The highlighted area represents the function X = A + B + C.

Note: An OR gate will give a logic 1 output if one or more of its inputs receives a logic 1 signal.

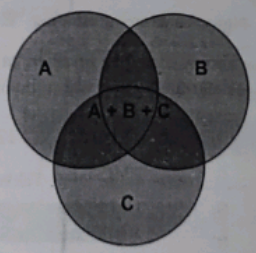
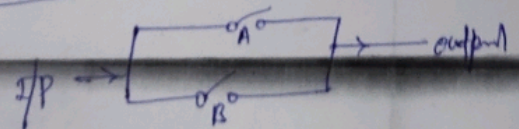
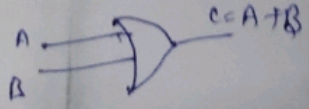


Figure 2.4 Venn Diagram illustrating OR Gate

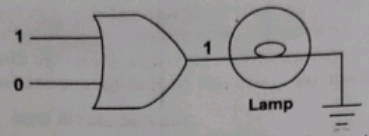
OR Gate



Input		output
A	B	C = A+B
0	0	0
0	1	1
1	0	1



We can explain the function of the OR gate with the help of an example that is discussed below:



According to this example, an input value of '1' at either of the OR gate inputs will result in an output value of '1' from the OR gate.

NOT gate The NOT gate performs a basic logic function called *inversion* or *complementation*. The purpose of this gate is to change one logic level (HIGH / LOW) to the opposite logic level. In terms of bits, it changes '1' to '0' and vice versa. The standard logic symbol for the NOT gate and a Venn diagram illustrating the relationship between the variables and the logic gate operation are shown in Figure 2.5 and Figure 2.6, respectively.

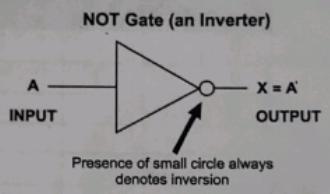


Figure 2.5 Logic Symbol of NOT Gate

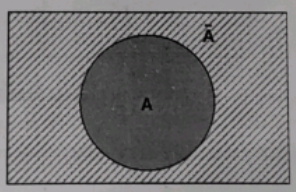


Figure 2.6 Venn Diagram Illustrating NOT Gate

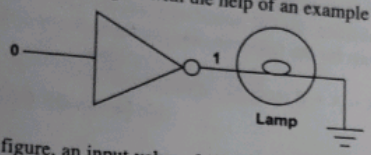
In electronics a NOT gate is more commonly called an *inverter*. The circle on the symbol is called a *bubble*, and is generally used in circuit diagrams to indicate an inverted input or output.

The truth table for the NOT gate is shown in Table 2.12.

Table 2.12 Truth Table of NOT Gate

A	X = A'
0	1
1	0

We can explain the function of the NOT gate with the help of an example that is discussed below:



With reference to the above figure, an input value of '0' at the NOT gate produces an output value of '1' from the NOT gate. In addition, an input value of '1' at the NOT gate produces an output value of '0' from the NOT gate.

2.8.2 Logic Operations

Generally, information is fed to a gate in the form of binary coded input signals, that is, either '0' or '1' and after performing certain logical operations; each combination of input signals yields a specific output (logic 0 or 1). Probably, these logical operations are based on making decision between two logics. There are basically three logical operations:

1. AND Operation
2. OR Operation
3. NOT Operation

AND operation In the AND operation, a result of 1 occurs when all of the input variables are 1. Furthermore, the output of this operation is 0 for any case where one or more inputs are 0. The multiplication sign stands for the AND operation, same for ordinary multiplication of 1s and 0s.

The expression X = A . B reads as 'X equals A AND B'. An example of two inputs AND gate and its truth table is as follows:

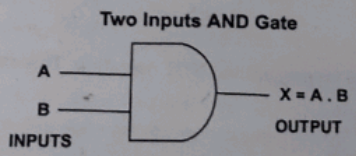


Figure 2.7 AND Gate

Table 2.13 Truth Table

A	B	X = A . B
0	0	0
0	1	0
1	0	0
1	1	1

With the AND operation, the outputs $1 \times 1 = 1$, $1 \times 1 \times 1 = 1$, and so on can be obtained.

OR operation According to the OR operation, a result of 1 is obtained when any of the input variable is 1. In addition, the OR operation produces a result of 0 only when all the input variables are 0. The + sign stands for the OR operation, not for ordinary addition.

The expression X = A + B reads as 'X equals A OR B'.

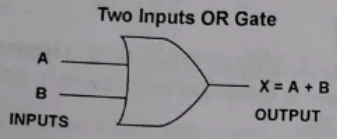


Figure 2.8 OR Gate

Table 2.14 Truth Table

A	B	X = A + B
0	0	0
0	1	1
1	0	1
1	1	1

With the OR operation, the outputs 1 + 1 = 1, 1+ 1 + 1 = 1, and so on can be obtained.

NOT operation The NOT operation is unlike the OR and AND operations. This operation can be performed on a single input variable. For example, if the variable A is subjected to the NOT operation, the result x can be expressed as:

$x = A'$ or $\bar{A}$

where, the prime (') represents the NOT operation. This expression is read as:
x equals NOT A
x equals the inverse of A
x equals the complement of A.

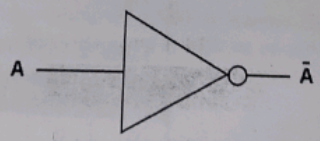


Figure 2.9 NOT Gate

Each of these is in common usage and all indicate that the logic value of x = A' is opposite to the logic value of A. The NOT operation is also referred to as inversion or complementation, and these terms are used interchangeably.

The truth table of the NOT operation is as follows:

Table 2.15 Truth Table

A	X = A'
0	1
1	0

With reference to the above table,
$1' = 0$ because NOT 1 is 0
$0' = 1$ because NOT 0 is 1.

2.9 BOOLEAN ALGEBRA

In the mid-eighteenth century, Boolean algebra was developed by the English mathematician, George Boole. It became known as Boolean algebra, after its developer's name. *Boolean algebra* is a mathematical system, which is used for formulation of the logical statements with symbols so that problems can be solved in a definite manner of ordinary algebra. In short, Boolean algebra is the mathematics of digital systems.

Since an ordinary algebraic expression may be simplified by means of the basic theorems, the expressions that describe a given digital network can also be reduced or simplified by using the Boolean algebra. Nowadays Boolean algebra is extensively used in designing the circuitry that is used in computers.

Since Boolean algebra deals with the binary number system, the variables used in the Boolean equations have only two possible values (0 or 1). Thus, for performing the logical algebraic operations, that is, 'addition' and 'multiplication', Boolean algebra follows certain rules. These rules are listed below in Table 2.16.

Table 2.16 Boolean Addition and Multiplication

Addition Rules	Multiplication Rules
0 + 0 = 0	0 · 0 = 0
0 + 1 = 1	0 · 1 = 0
1 + 0 = 1	1 · 0 = 0
1 + 1 = 1	1 · 1 = 1

2.9.1 Laws of Boolean Algebra

1. **Commutative Law**
   The commutative law of addition for two variables is algebraically expressed as:
   $A + B = B + A$

The commutative law of multiplication for two variables is expressed as:
$AB = BA$

In summary, the orders in which the variables are ORed or ANDed make no difference.

2. **Associative Law**
The associative law of addition of three variables is expressed as:
$A + (B + C) = (A + B) + C$
The associative law of multiplication of three variables is expressed as:
$A\,(BC) = (AB)\,C$
In summary, ORing or ANDing a group of variables produces the same result regardless of the group of the variables.

3. **Distributive Law**
The distributive law of three variables is expressed as follows:
$A\,(B + C) = AB + AC$
$A + (BC) = (A + B)\,(A + C)$
This law states that ORing several variables and ANDing the result is equivalent of ANDing the single variable with each of the variables in the grouping, then ORing the result.

4. **De Morgan's Law**
For N variables, De Morgan's theorems are expressed by the following formulas:
$\overline{A + B} = \overline{A} \cdot \overline{B}$
$\overline{A \cdot B} = \overline{A} + \overline{B}$

According to above formulas, the complement of the product is equivalent to the sum of the complements. Similarly, the complement of the sum is equivalent to the product of the complements.

2.9.2 Rules of Boolean Algebra

By combining the laws of Boolean algebra and our knowledge of logic gates, we form several useful rules that may be used in manipulating and simplifying Boolean algebra expressions. Rules 1-8, as listed in Table 2.17, are the core precepts from which rules 9-10 are derived. Note that in each case, A, B, or C can represent a single variable for a combination of variables.

Table 2.17 Rules of Boolean Algebra

Rule Number	Boolean Expression
1	$A + 0 = A$
2	$A + 1 = 1$
3	$A \cdot 0 = 0$
4	$A \cdot 1 = A$

5	$A + A = A$
6	$A + \overline{A} = 1$
7	$A \cdot A = A$
8	$A \cdot \overline{A} = 0$
9	$A + AB = A$
10	$A + \overline{A}B = A + B$

2.9.3 Boolean Functions

A Boolean function is an expression formed with binary variables and logical operators (OR, AND, NOT, and equal sign). In essence, a truth table is a list, which defines a Boolean function. For example, $X = A \cdot B + A \cdot C$. Let us consider the truth table of the given value as shown in Table 2.18. Note that the Function (X) is equal to 1 at three conditions, that is, A = 1, B = 0, C = 1; A = 1, B = 1, C = 0 and A = 1, B = 1, C = 1 otherwise X = 0. The algebraic expression representing this function is therefore expressed as:

$$X = A\overline{B}C + AB\overline{C} + ABC$$

Table 2.18 Truth Table for Boolean Functions

Input			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Accordingly, the logic circuit of the given example is shown in Figure 2.10.

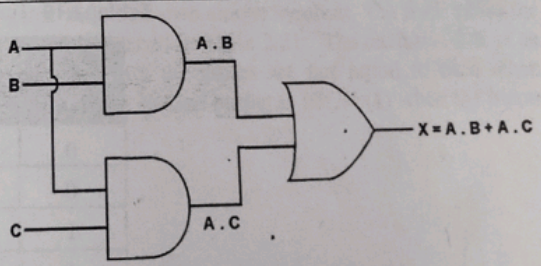


Figure 2.10 Logic Circuit

2.10 COMBINATION OF LOGIC GATES

When we look at logic circuit diagrams for digital equipment, we are not going to see just a single gate, but many combinations of gates. Using combinations of logic gates, complex operations can be performed. Although there is no limit to the number of gates that can be arrayed together in a single device, nevertheless, in practice, there is a limit to the number of gates that can be packed into a given physical space. In this section, we will analyse some basic combinations of gates. These gates can be listed as:

- NAND gate
- NOR gate
- Exclusive-OR (XOR) and Exclusive-NOR (XNOR) gate

NAND gate The NAND, which is composed of two or more inputs and a single output, is a very popular logic element because it may be used as a *universal function*. That is, it may be employed to construct an inverter, an AND gate, an OR gate, or any combination of these functions. The term NAND is formed by the combination of NOT-AND and implies an AND function with an inverted output. The standard symbol for the NAND gate is shown in Figure 2.11 and its truth table is listed in Table 2.19. The logical operation of the NAND gate is such that the output is LOW (0) only when all the inputs are HIGH (1).

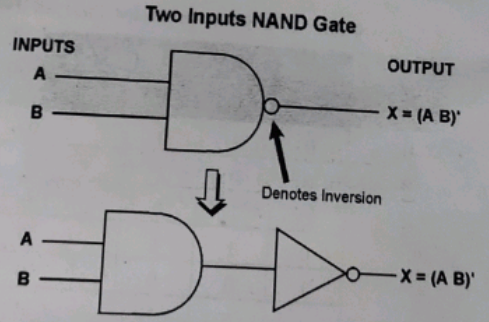
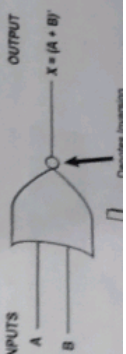


Figure 2.11 Standard Logic Symbol of NAND Gate

Table 2.19 Truth Table for NAND Gate

INPUT		OUTPUT
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate The NOR gate, which is composed of two or more inputs and a single output, also has a universal property. The term NOR is formed by the concatenation NOT-OR and implies an OR function with an inverted output. The standard symbol for the NOR gate is shown in Figure 2.12 and its truth table is listed in Table 2.20. The logical operation of the NOR gate is such that the output is HIGH (1) only when all the inputs are LOW.



NOR gate The NOR gate, which is composed of two or more inputs and a single output, also has a universal property. The term NOR is formed by the concatenation NOT-OR and implies an OR function with an inverted output. The standard symbol for the NOR gate is shown in Figure 2.12 and its truth table is listed in Table 2.20. The logical operation of the NOR gate is such that the output is HIGH (1) only when all the inputs are LOW.

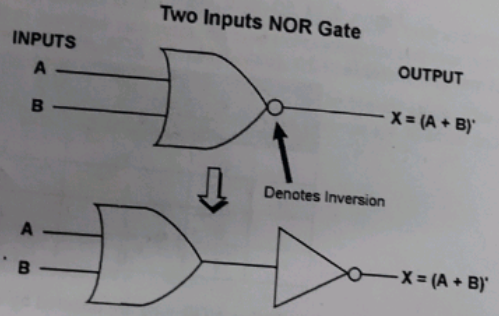


Figure 2.12 Standard Logic Symbol of NOR Gate

Table 2.20 Truth Table for NOR Gate

INPUT		OR	NOR
A	B	$X = A + B$	$X = \overline{A} \cdot \overline{B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Exclusive-OR (XOR) and exclusive-NOR (XNOR) gate These gates are usually formed from the combination of the other logic gates already discussed. However, because of their functional importance, these gates are treated as basic gates with their own unique symbols. The truth tables for the XOR and XNOR gates, shown in Figure 2.13, are listed in Table 2.21. The exclusive-OR is an 'inequality' function and the output is HIGH (1), when the inputs are **not equal** to each other. Conversely, the exclusive-NOR is an 'equality' function and the output is HIGH (1) when the inputs are equal to each other.

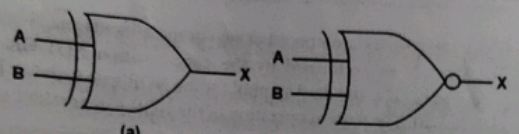


Figure 2.13 Standard Logic Symbols for (a) XOR (b) XNOR

Table 2.21 Truth Table for XOR and XNOR Logic Gates

INPUTS		XOR	XNOR
A	B	X	X
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

The exclusive-OR gate and exclusive-NOR gate are denoted by the $\oplus$ and $\odot$, respectively.

In addition, these gates perform the following Boolean functions:

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

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

Let Us Summarise

1. Number systems have been around for thousands of years. It defines a set of values used to represent the quantity, and other special characters.
2. Number systems basically are of two types: non-positional and positional number systems.
3. In a non-positional number system, special symbols or characters are used to indicate the values. It is very difficult to perform arithmetic with such a number system, as it has no symbol for zero.
4. In a positional number system, the value of each digit in a number is defined by the symbols but also by the symbol's position. These symbols are called as *digits*.
5. The positional number system, which is being used nowadays, is called as the *decimal number system*. Apart from this number system, there are some other positional number systems such as binary number system, octal number system, and hexadecimal number system.
6. The base or radix of the number system tells the number of symbols or digits used in the system. The base of the decimal number system is 10, of binary number system is 2, of octal number system is 8, and of hexadecimal number system is 16.
7. The primary number system used in our day-to-day life is the decimal number system. This number system includes ten digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).

8. The modern computer systems are operating by using the binary number system. This system is based on the number two and deals with only two numbers: 0 and 1.
9. In the hexadecimal number system, each hexadecimal number represents a power of 16. To represent the decimal numbers, this system uses 0 to 9 numbers and A to F characters to represent 10 to 15 numbers, respectively.
10. Every number system can be converted into another number system such as decimal to binary and vice versa, decimal to octal and vice versa, decimal to hexadecimal and vice versa, binary to octal and vice versa, and so on. However, the method of each conversion is different from one another.
11. All the computers perform the arithmetic operations in the binary mode. The basic arithmetic operations that have been performed by all the number systems are addition and subtraction.
12. The rules of binary addition are as follows:
    $0 + 0 = 0$
    $0 + 1 = 1$
    $1 + 0 = 1$
    $1 + 1 = 0$ plus a carry of 1 to next higher column
13. The rules of binary subtraction are as follows:
    $0 - 0 = 0$
    $1 - 0 = 1$
    $1 - 1 = 0$
    $0 - 1 = 1$ with a borrow from the next column
14. The *complement* of a number is the number which when added to the original will make it equal to a multiple of the base number system. The complement of a number can be used to represent a number as a *negative* and a *positive* number.
15. The addition and subtraction of the signed numbers is dependent on the 2s complement of the numbers and whenever the 2s complement of a number is added to any other binary number, it will be equivalent to its subtraction from that number.
16. When we add the like-signed numbers and subtract the unlike-signed numbers, the condition of overflow or underflow can occur.
17. The binary coding schemes are used to represent the internal storage area of the computers. In binary coding, every character is represented by a combination of bits.
18. The most commonly used computer coding systems are BCD, ASCII, and EBCDIC.
19. BCD (Binary Coded Decimal) is a method that represents the decimal digits with the help of binary digits. It is a 6-bit code, which can represent a maximum of 64 different characters.
20. ASCII is a 8-bit code and is exclusively used to represent the data internally in the microcomputers. It can represent 128 different characters.
21. EBCDIC or Extended Binary Coded Decimal Interchange Code uses 8 bits for each character and can represent 256 different characters. It provides a unique code for each decimal value 0 through 9 and for a variety of special characters.
22. A logic gate is a building block of a digital circuit that operates on one or more input signals to obtain the standard output signals.

23. An AND gate is an electronic circuit, which performs the logical multiplication operation. T... circuit generates an output signal of 1, only if all input signals are also 1.
24. An OR gate is an electronic circuit, which performs the logical addition operation. This gate generates an output signal of 1, if any of the input signals is 1.
25. The NOT gate performs a basic logic function called *inversion* or *complementation*. In electronics, a NOT gate is more commonly called an *inverter*. The purpose of this gate is to change one logic level (HIGH / LOW) to the opposite logic level. In terms of bits, it changes a '1' to a '0' and vice versa.
26. Information is fed to a gate in the form of binary coded input signals, that is, either '0' or '1'and after performing certain logical operations; each combination of input signals yields a specific output (logic 0 or 1). There are basically, three logical operations: AND operation, OR operation, NOT operation.
27. *Boolean Algebra* is a mathematical system, which is used for the formulation of the logical statements with symbols so that problems can be solved in a definite manner to ordinary algebra. In short, Boolean algebra is the mathematics of digital systems.
28. Boolean algebra deals with the binary number system, the variables used in the Boolean equations have only two possible values (0 or 1). Boolean algebra is extensively used in designing the circuitry that is used in computers.
29. For manipulating and simplifying Boolean algebra expressions, we follow certain rules and laws of Boolean algebra.
30. The laws of Boolean algebra include:
    - Commutative Law:
      $A + B = B + A$ (of addition)
      $AB = BA$ (of multiplication)
    - Associative Law:
      $A + (B + C) = (A + B) + C$ (of addition)
      $A (BC) = (AB) C$ (of multiplication)
    - Distributive Law:
      $A (B + C) = AB + AC$
      $A + (BC) = (A + B) (A + C)$
    - De Morgan's Law:
      $\overline{A + B} = \overline{A} \cdot \overline{B}$
      $\overline{A \cdot B} = \overline{A} + \overline{B}$
31. A Boolean function is an expression formed with binary variables and logical operators (OR, AND, NOT, and equal sign). In essence, a truth table is a list, which defines a Boolean function.
32. The combinational logic gates are the interconnection of the logic gates for forming other logic gates or networks. Using combinations of logic gates, complex operations can be performed. Some of these combinational logic gates are: NAND gate, NOR gate, and Exclusive-OR (XOR) and Exclusive-NOR (XNOR) gate.

BCD Counter Summary

In this tutorial we have seen that a ***BCD Counter*** *is a devices that goes through a sequence of ten states when it is clocked and returns to 0 after the count of 9. In our simple example above, the input clock pulses are from a push button switch but counters can be used to count many real-world events such as counting moving objects.*

However, suitable circuitry may be required to generate the electrical pulses for each event to be counted as these events may occur at discrete time intervals or they may be completely random.

In many digital electronic circuits and applications, digital counters are counters that go through a sequence of Toggle flip-flops or with any other type of flip-flop switching function, or with the use of flip-flops that go through a sequence of "n" number of flip-flop states.

BCD Counter Summary

In this tutorial we have seen that a **BCD Counter** is a devices that goes through a sequence of ten states when it is clocked and returns to 0 after the count of 9. In our simple example above, the input clock pulses are from a push button switch but counters can be used to count many real-world events such as counting moving objects.

However, suitable circuitry may be required to generate the electrical pulses for each event to be counted as these events may occur at discrete time intervals or they may be completely random.

In many digital electronic circuits and applications, digital counters are implemented using Toggle flip-flops or with any other type of flip-flop that can be connected to give the required switching function, or with the use of dedicated counting IC's such as the 74LS90. Binary counters are counters that go through a binary sequence and an n-bit binary counter is made of "n" number of flip-flops counting from 0 to 2n-1.

BCD counters follow a sequence of ten states and count using BCD numbers from 0000 to 1001 and then returns to 0000 and repeats. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits giving a MOD-10 count.

We have also seen that the BCD coded output can be displayed using four LED's or with a digital display. But to display each number from 0 to 9 requires a decoder circuit, which translates a binary coded number representation into the appropriate logic levels on each of the display segments.

Display decoder circuits can be constructed from combinational logic elements and there are many dedicated integrated circuits on the market to perform this function such as the 74LS47 BCD to 7-segment decoder/driver IC.

Most 7-segment displays are usually used in multi-digit counting applications so by cascading together more BCD counters, 4-digit counters giving displays with a maximum reading of 9999 can be constructed. The 74LS90 BCD Counter is a very flexible counting circuit and can be used as a frequency divider or made to divide any whole number count from 2 to 9 by feeding the appropriate outputs back to the IC's Reset and Set inputs

BCD Counter

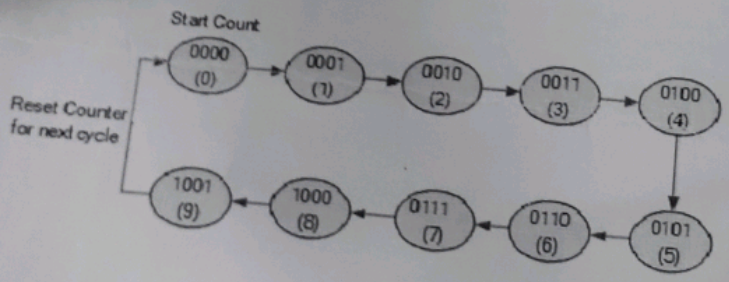
A BCD counter is a special type of a digital counter which can count to ten on the application of a clock signal. We saw previously that toggle T-type flip flops can be used as individual divide-by-two counters. If we connect together several toggle flip-flops in a series chain we can produce a digital counter which stores or display the number of times a particular count sequence has occurred.

Clocked T-type flip-flops act as a binary divide-by-two counter and in asynchronous counters, the output of one counting stage provides the clock pulse for the next stage. Then a flip-flop counter has two possible output states and by adding more flip-flop stages, we can make a divide-by-2^N counter. But the problem with 4-bit binary counters is that they count from 0000 to 1111. That is from 0 to 15 in decimal.

To make a digital counter which counts from 1 to 10, we need to have the counter count only the binary numbers 0000 to 1001. That is from 0 to 9 in decimal and fortunately for us, counting circuits are readily available as integrated circuits with one such circuit being the **Asynchronous 74LS90 Decade Counter.**

Digital counters count upwards from zero to some pre-determined count value on the application of a clock signal. Once the count value is reached, resetting them returns the counter back to zero to start again. A decade counter counts in a sequence of ten and then returns back to zero after the count of nine. Obviously to count up to a binary value of nine, the counter must have at least four flip-flops within its chain to represent each decimal digit as shown.

BCD Counter State Diagram



Then a decade counter has four flip-flops and 16 potential states, of which only 10 are used and if we connected a series of counters together we could counter to 100 or 1,000 or what ever number we wanted.

The total number of counts that a counter can count too is called its MODULUS. A counter that returns to zero after **n** counts is called a *modulo-n counter*, for example a modulo-8 (MOD-8), or modulo-16 (MOD-16) counter, etc, and for an "n-bit counter", the full range of the count is from 0 to 2n-1.

But as we saw in the Asynchronous Counters tutorial, that a counter which resets after ten counts with a divide-by-10 count sequence from binary 0000 (decimal "0") through to 1001 (decimal "9") is called a binary-coded-decimal counter or **BCD Counter** for short and a MOD-10 counter can be constructed using a minimum of four toggle flip-flops.

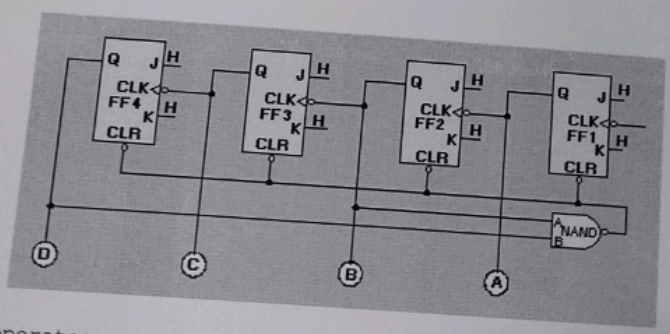
It is called a BCD counter because its ten state sequence is that of a BCD code and does not have a regular pattern, unlike a straight binary counter. Then a single stage BCD counter such as the 74LS90 counts from decimal 0 to decimal 9 and is therefore capable of counting up to a maximum of nine pulses. Note also that a digital counter may count up or count down or count up and down (bidirectional) depending on an input control signal.

Binary-coded-decimal code is an **8421** code consisting of four binary digits. The 8421 designation refers to the binary weight of the four digits or bits used. For example, $2^3 = 8$, $2^2 = 4$, $2^1 = 2$ and $2^0 = 1$. The main advantage of BCD code is that it allows for the easy conversion between decimal and binary forms of numbers.

Decade Counter

A decade counter is a binary counter that is designed to count to 10_{10}, or 1010_2. An ordinary four-stage counter can be easily modified to a decade counter by adding a NAND gate as shown in figure 3-25. Notice that FF2 and FF4 provide the inputs to the NAND gate. The NAND gate outputs are connected to the CLR input of each of the FFs.

Figure 3-25. - Decade counter.



The counter operates as a normal counter until it reaches a count of 1010_2, or 10_{10}. At that time, both inputs to the NAND gate are HIGH, and the output goes LOW. This LOW applied to the CLR input of the FFs causes them to reset to 0. Remember from the discussion of J-K FFs that CLR and PS or PR override any existing condition of the FF. Once the FFs are reset, the count may begin again. The following table shows the binary count and the inputs and outputs of the NAND gate for each count of the decade counter:

BINARY COUNT	NAND GATE INPUTS		NAND GATE OUTPUT
*******	A	B	*******
0000	0	0	1
0001	0	0	1
0010	1	0	1
0011	1	0	1
0100	0	0	1
0101	0	0	1
0110	1	0	1
0111	1	0	1
1000	0	1	1
1001	0	1	1

Changing the inputs to the NAND gate can cause the maximum count to be changed. For instance, if FF4 and FF3 were wired to the NAND gate, the counter would count to 1100_2 (12_{10}), and then reset.