DIGITAL LOGIC DESIGN

3RD sem CSE
Faculty Name: MITALI PANDA

Content

Module-I
Introduction: Logic Design, Transistors as switches, CMOS Gates, Sequential circuits, Some Examples.

Digital Systems: Representation of numbers, binary codes, Gray code, Error- detecting and error-correcting codes, registers, binary logic, basic logic gates.

Boolean Algebra: Boolean operations, Boolean functions, Algebraic manipulation, Minterms and Maxterms, Sum-of-products and product-of-sum representations, two-input logic gates, functional completeness.

Chapter-1

Introduction

All of us are familiar with the impact of digital calculators, watches, modern communication systems and computers in everyday life. All persons working in various fields related to electronics must understand the performance of Digital Electronic Circuits. All sizes of computers, as we know, perform complicated task with fantastic speed and accuracy. At stores, the cash register read out digital display digital clock and watches flash the time in all city shops and restaurants. Most automobiles use microprocessors to control engine functions. Aircraft's defence sectors, factory machines and modern diagnostic in medical science are controlled by digital circuits.
This revolution took place with the advent of integrated circuits (IC) which is an offspring of semiconductor technology. The inexpensive fabrication of ICs has made the subject Digital Electronics easy to study. One small IC can perform the task of thousands of Transistors Diodes and Resistors. Many ICs are used to construct Digital Circuits. This is an exciting and rapidly growing field, which uses several principles for the working of computers, Communication systems, Digital machinery's etc. The basic idea is to let the beginners understand the operation of the Digital system and many other systems based on the principles of Digital Techniques. Any device working under Digital Techniques are called Digital Systems and the Electronic Network used to make them operational are called Digital Circuits. The subject as a whole is often referred as **Modern Digital Electronics.**

Electronic circuits use two kinds of signals. They are **Analog Signals**(continuous supply of voltages and currents) and **Digital Signals** (discrete voltages and current). For example, Circuits (electronic network) using Analog signals are known as **Linear or Analog Circuits**. Similarly, the electronic network of an electronic calculator or digital watch that uses Digital signals are called **Digital Circuits**.

An analog device, then, is one that has a signal, which varies continuously in time with the input, whereas, a digital device operates with a digital signal that varies discontinuously. As a result, the Digital Electronics is the world of **ZEROS'** (OFF/LOW/DOWN/FALSE) and '**ONES**' (ON/HIGH/UP/TRUE). Figure1.1 shows the behavior of Analog and Digital signal and the possibility of conversion from Analog to Digital. Figure 1.2 represents a symbol of some devices using Analog and Digital world. This example is set to explain how real life problem like movement of a pivot on a water tank that indicates the level of water can be translated to analog/digital form.

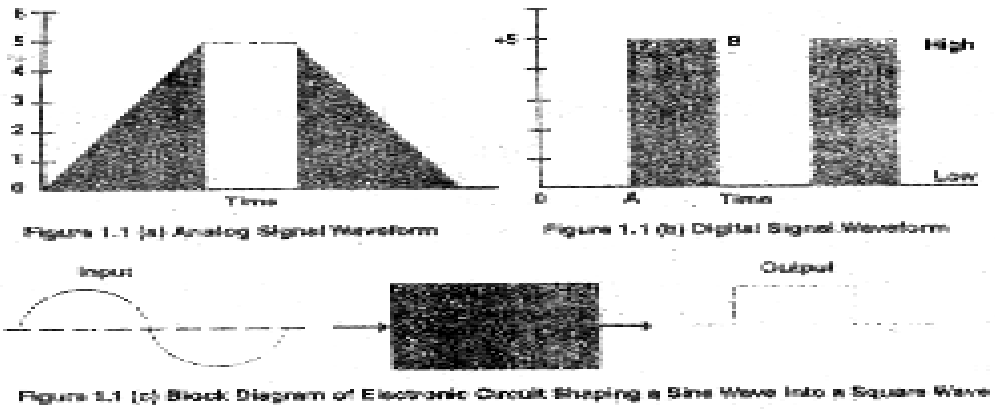


Figure 1.1 (a) Analog Signal Waveform

Figure 1.1 (b) Digital Signal Waveform

Figure 1.1 (c) Block Diagram of Electronic Circuit Shaping a Sine Wave Into a Square Wave

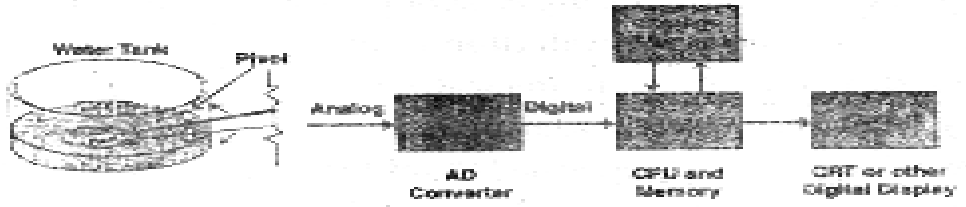


Figure 1.2 Digital System used to Interpret Float Level in Water Tank

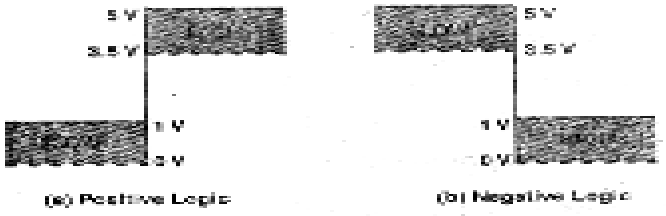


Figure 1.3 Digital Signal Representation

Characteristics of Digital systems

• Digital systems manipulate discrete elements of information.

• Discrete elements are nothing but the digits such as 10 decimal digits or 26 letters of alphabets and so on.

• Digital systems use physical quantities called signals to represent discrete elements.

• In digital systems, the signals have two discrete values and are therefore said to be binary.

• A signal in digital system represents one binary digit called a bit. The bit has a value either 0 or 1.

ANALOG Vs DIGITAL:

To learn and understand about the digital logic design, the initial knowledge we require is to differentiate between analog and digital. The following are fews that differentiate between analog and digital.

• Analog information is made up of a continuum of values within a given range.

• At its most basic, digital information can assume only one of two possible values: one/zero, on/off, high/low, true/false, etc.

• Digital Information is less susceptible to noise than analog information

• Exact voltage values are not important, only their class (1 or 0)

• The complexity of operations is reduced, thus it is easier to implement them with high accuracy in digital form.

Advantages of Digital system over Analog system

1. Ease of programmability
The digital systems can be used for different applications by simply changing the program without additional changes in hardware.

2. Reduction in cost of hardware
The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.

3.High speed
Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.

4. High Reliability Digital systems are highly reliable one of the reasons for that is use of error correction codes.

5. Design is easy
The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.

6. Result can be reproduced easily
Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

Disadvantages of Digital Systems

• Use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well.

• Digital circuits are often fragile, in that if a single piece of digital data is lost or misinterpreted the meaning of large blocks of related data can completely change.

• Digital computer manipulates discrete elements of information by means of a binary code.

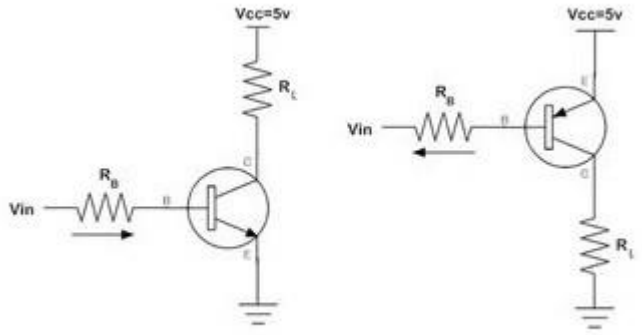
• Quantization error during analog signal sampling.

Transistor as a Switch

Basically as per the generations of the electronic circuits gets revolutionized and improved for better and comfortable living the transistors played a prominent role by replacing themselves with vacuum tubes.

This leads to an improvement in efficiency and compression in size. The main functionality of the transistor can be observed either by making it be used for amplification or to the basic application in the digital circuitry of switching.

The main reason behind using this transistor for the purpose of the switch is that the current at the base controls the current present at the collector directly. If the current at the base exceeds the minimum cut-off value of voltage then the behaviour of the transistor is like a close switch otherwise it will remain in open switch condition.



By the application of bias to the base of the transistor both the types in the bipolar junction transistor can be used as switches. The areas at which the operation of the switch is preferred is either it should be completely in the region called saturation or the cut-off operating region. The main theme behind using these regions is that switch mode should be completely ON or OFF.

N-P-N Transistor as a Switch

Once the voltage has been applied to the region of base based on it the operation of the switching is performed. Similar to the condition of the diode there exist the cut-in voltage. Between the region of the emitter and the base, the voltage applied must reach the cut-in voltage. If its crosses it the transistor is said to be ON otherwise OFF.

Once the transistor is in ON condition the current generated tends to flow from source to the load. The load can be either the led or the resistor, load depends based on the requirement.

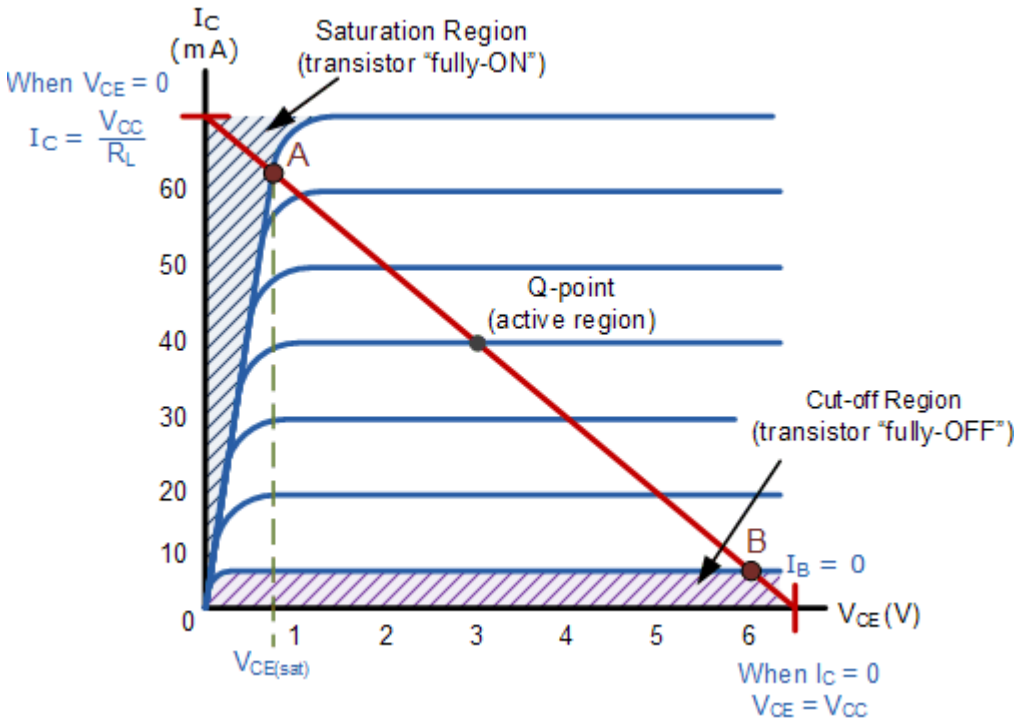
P-N-P Transistor as a Switch

The operating conditions of the P-N-P and N-P-N transistor differ in the application positive or the negative voltages. But the criteria of the operation remain the same. If it is in ON condition the flow of current is observed otherwise it is OFF.

Applications

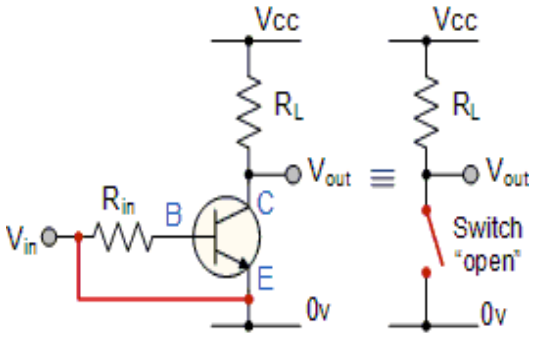
The applications of the transistor used as a switch are as follows:

1. The most frequently used practical application that is used for the transistor as a switch is functioning of LED.
2. The relay operation can be controlled by making the necessary changes in the circuit so that any external device is connected with respect to the relay and gets controlled.
3. The dc motors can be controlled and monitored by using this concept of transistors. In order to turn ON the motor and OFF it this application is used. By varying the values of the frequencies of transistor the speed of the motor can be varied.
4. One of the examples of these switches is light-bulb. It facilitates to switch on the light provided a bright environment and switches off based on the dark environment. It is done by using a light-dependent resistor (LDR).
5. A component called thermistor that senses the surrounding temperature can be monitored by using this switching technique. The thermistor is referred to as a resistor. This resistance tends to increase when the sensed temperature is low and a decrease in the resistance is observed when the sensed temperature is high.



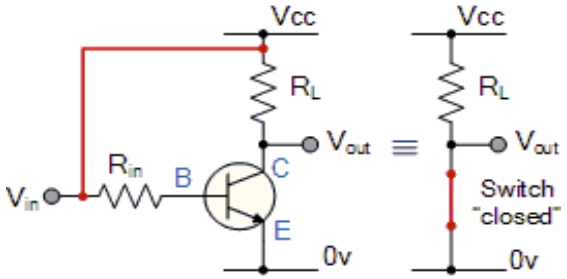
Cut-off Region

Here the operating conditions of the transistor are zero input base current (I_B), zero output collector current (I_C) and maximum collector voltage (V_{CE}) which results in a large depletion layer and no current flowing through the device. Therefore the transistor is switched "Fully-OFF".



Saturation Region

Here the transistor will be biased so that the maximum amount of base current is applied, resulting in maximum collector current resulting in the minimum collector emitter voltage drop which results in the depletion layer being as small as possible and maximum current flowing through the transistor. Therefore the transistor is switched "Fully-ON".



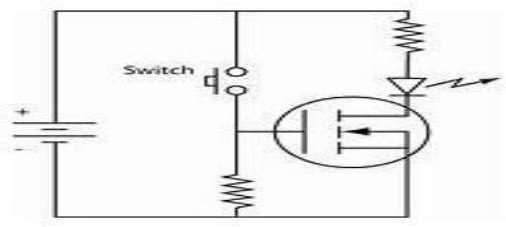
MOSFET as a Switch

When the voltage value of the gate and the source is greater than the threshold then it is referred to as the region of saturation where the device tends to turn ON. But it is dependent on the voltage across the terminals of the drain as well as the source. The resistance is maintained to be constant during this stage.

When the value of the voltage at the terminals of the gate and the source is less than the threshold the device turns to be OFF this region of operation is known as the cut-off region. As the device turning ON and OFF is dependent on the value of the voltages applied this device is known to be voltage-driven.

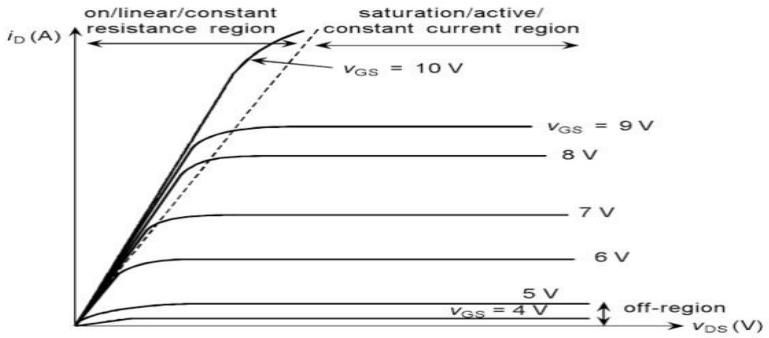
N-Channel MOSFET Switch Circuit

In this switching circuit where n-channel is used and the gate-source terminals are supplied by the positive polarity of the voltage. This positive polarity makes the device that is connected to the transistor makes it turn ON. The arrow marks in the symbol indicates the direction of flow of charge carriers.

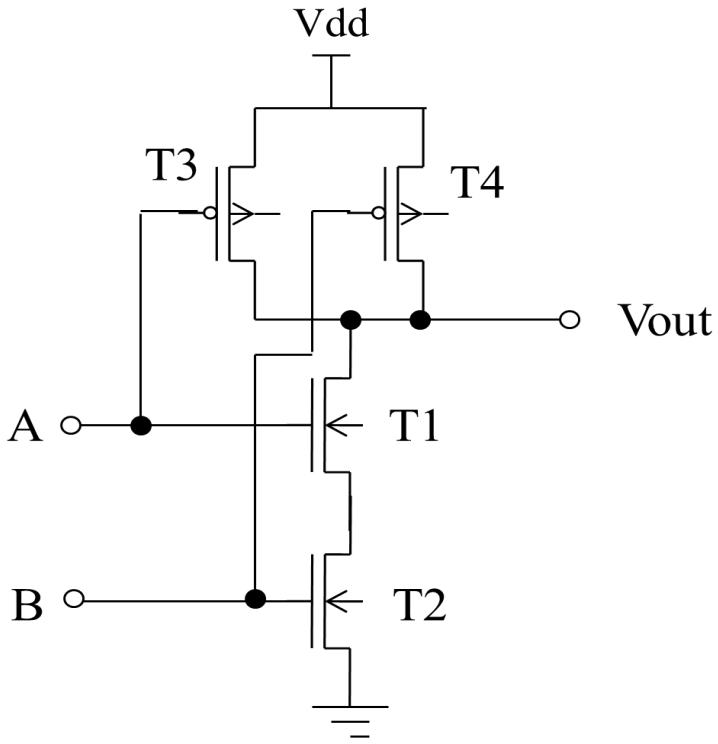


P-Channel MOSFET Switch Circuit

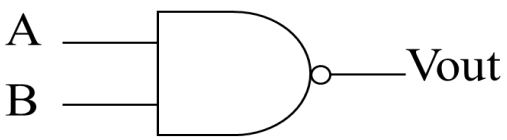
As the channel considered here is made up of p-type then, in this case, the negative voltage must be applied to the gate so that the device gets turned ON.



NAND Gate using CMOS

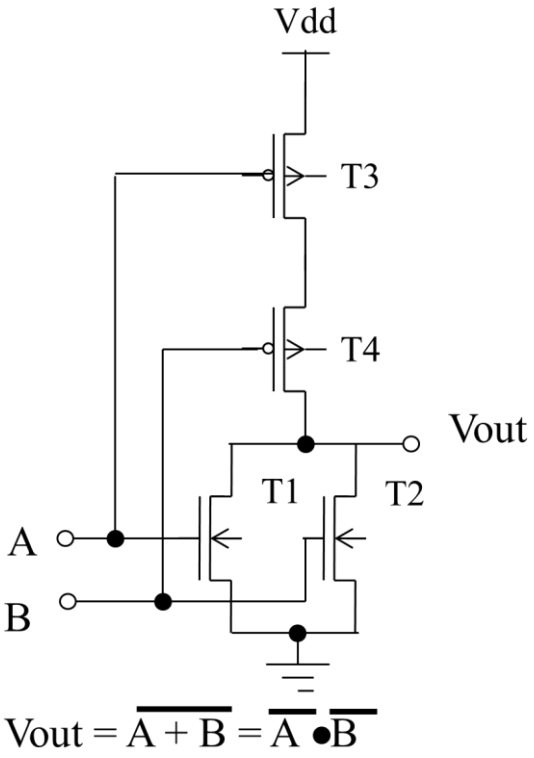


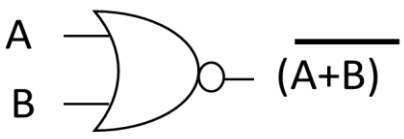
$$Vout = \overline{A \cdot B} = \overline{A} + \overline{B}$$



- Circuit Topology:
  - T1 and T2 are N-FET devices connected in series; T3 and T4 are P-FET devices connected in parallel with their sources at Vdd and their drains at Vout.
  - Inputs A and B are connected to the gates of T1 & T3 and T2 & T4, respectively.
  - T2, T3, & T4 operate as "grounded source" devices, but T1 has its source generally above Gnd potential.
- Operation:
  - If both A and B are high (at Vdd), both T1 and T2 are ON hard, therefore pulling Vout low (to zero volts). Both T3 and T4 are OFF due to their gate-to-source voltages (Vgs) being at 0 volts, thus preventing any dc current.
  - If either A or B (or both) are low (at 0 volts), either T1 or T2 (or both) are OFF; T3 or T4 (or both) are ON hard, thus pulling Vout high to Vdd (a "1" output).

NOR Gate using CMOS





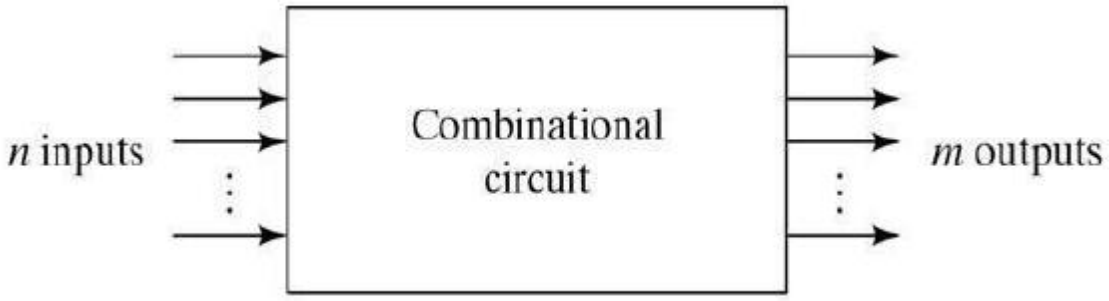
- Circuit Topology:
  - T1 and T2 are N-FET devices connected in parallel with their sources at Gnd and drains at Vout; T3 and T4 are P-FET devices connected in series.

  - Inputs A and B are connected to the gates of T1 & T3 and T2 & T4, respectively.
- Operation:
  - If either A or B is high, T1 and/or T2 are ON hard and either T3 or T4 (or both) are OFF, pulling Vout to gnd. No dc current flows.
  - If both A and B are low (at gnd), both T1 and T2 are OFF and both T3 and T4 are ON hard, thus pulling Vout to Vdd (a "1" output).
  - T1, T2, and T3 operate as common source, but T4's source potential will drop below Vdd.

Combinational Logic

• Logic circuits for digital systems may be combinational or sequential.

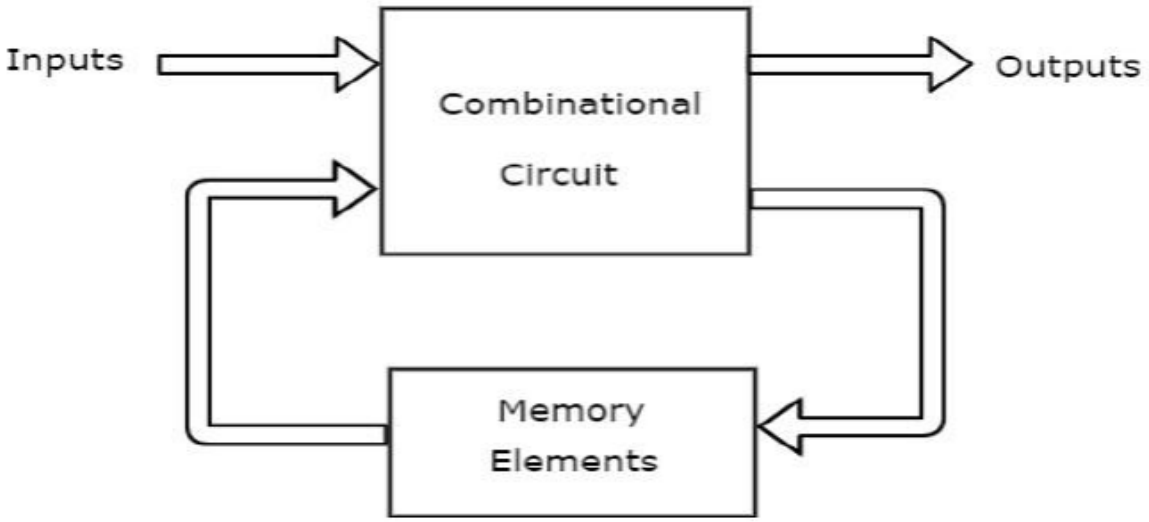
• A combinational circuit consists of input variables, logic gates, and output variables.



For n input variables, there are 2n possible combinations of binary input variables .For each possible input Combination, there is one and only one possible output combination. A combinational circuit can be described by m Boolean functions one for each output variables. Some examples are Adders, Subtractor, MUX etc.

Sequential Circuits

The following figure shows the **block diagram** of sequential circuit.



This sequential circuit contains a set of inputs and output(s). The output(s) of sequential circuit depends not only on the combination of present inputs but also on the previous output(s). Previous output is nothing but the **present state**. Therefore, sequential circuits contain combinational circuits along with memory (storage) elements. Some sequential circuits may not contain combinational circuits, but only memory elements. Some examples are Latches, Flip-flops.

Following table shows the **differences** between combinational circuits and sequential circuits.

Combinational Circuits	**Sequential Circuits**
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

Chapter-2

Digital Systems:
Representation of numbers

Many number systems are in use in digital technology. The most common are the decimal, binary, octal, and hexadecimal systems. The decimal system is clearly the most familiar to us because it is a tool that we use every day. Examining some of its characteristics will help us to better understand the other systems. In the next few pages we shall introduce four numerical representation systems that are used in the digital system. There are other systems, which we will look at briefly.

- Decimal
- Binary
- Octal
- Hexadecimal

Decimal System

The decimal system is composed of 10 numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Using these symbols as digits of a number, we can express any quantity. The decimal system is also called the base-10 system because it has 10 digits.

Decimal Examples

3.1410
5210
102410
6400010

Binary System

In the binary system, there are only two symbols or possible digit values, 0 and 1. This base-2 system can be used to represent any quantity that can be represented in decimal or other base system. In digital systems the information that is being processed is usually presented in binary form. Binary quantities can be represented by any device that has only two operating states or possible conditions.

E.g.. a switch is only open or closed. We arbitrarily (as we define them) let an open switch represent binary 0 and a closed switch represent binary 1. Thus we can represent any binary number by using series of switches.

Octal System

The octal number system has a base of eight, meaning that it has eight possible digits:
0,1,2,3,4,5,6,7.

Hexadecimal System

The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols.

Code Conversion

Converting from one code form to another code form is called code conversion, like converting from binary to decimal or converting from hexadecimal to decimal.

Binary-To-Decimal Conversion

Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1.e.g.

$1001.01_2 = [(1) \times 2^3] + [(0) \times 2^2] + [(0) \times 2^1] + [(1) \times 2^0] + [(0) \times 2^{-1}] + [(1) \times 2^{-2}]$

$1001.01_2 = [1 \times 8] + [0 \times 4] + [0 \times 2] + [1 \times 1] + [0 \times 0.5] + [1 \times 0.25]$

$1001.01_2 = 9.25_{10}$

Octal to Decimal Conversion
$237_8 = 2 \times (8^2) + 3 \times (8^1) + 7 \times (8^0) = 159_{10}$
$24.6_8 = 2 \times (8^1) + 4 \times (8^0) + 6 \times (8^{-1}) = 20.75_{10}$
$11.1_8 = 1 \times (8^1) + 1 \times (8^0) + 1 \times (8^{-1}) = 9.125_{10}$
$12.3_8 = 1 \times (8^1) + 2 \times (8^0) + 3 \times (8^{-1}) = 10.375_{10}$

Hexadecimal to Decimal Conversion
$24.6_{16} = 2 \times (16^1) + 4 \times (16^0) + 6 \times (16^{-1}) = 36.375_{10}$
$11.1_{16} = 1 \times (16^1) + 1 \times (16^0) + 1 \times (16^{-1}) = 17.0625_{10}$
$12.3_{16} = 1 \times (16^1) + 2 \times (16^0) + 3 \times (16^{-1}) = 18.1875_{10}$

Octal-To-Binary Conversion
Each Octal digit is represented by three binary digits.
Example:
$4\ 7\ 2_8 = (100)\ (111)\ (010)_2 = 100\ 111\ 010_2$

Octal-To-Hexadecimal Hexadecimal-To-Octal Conversion

- Convert Octal (Hexadecimal) to Binary first.
- Regroup the binary number by three bits per group starting from LSB if Octal is required.
- Regroup the binary number by four bits per group starting from LSB if Hexadecimal is required.

Complements:
In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.
i) The radix complement or r's complement
ii) The diminished radix complement or (r-1)'s complement

Representation of signed no.s binary arithmetic in computers:
• Two ways of rep signed no.s
1. Sign Magnitude form
2. Complemented form
    (i) 1's compliment form
    (ii) 2's compliment form

Advantage of performing subtraction by the compliment method is reduction in the hardware.(instead of addition & subtraction only adding ckt's are needed.) i. e, subtraction is also performed by adders only.

Instead of subtracting one no. from other the compliment of the subtrahend is added to minuend. In sign magnitude form, an additional bit called the sign bit is placed in front of the no. If the sign bit is 0, the no. is +ve, If it is a 1, the no is -ve.

Representation of signed no.s using 2's or 1's complement method: If the no. is +ve, the magnitude is written in its true binary form & a sign bit 0 is placed in front of the MSB. If the no

is -ve , the magnitude is written in its 2‘s or 1‘s compliment form &a sign bit 1 is placed in front of the MSB.

Special case in 2’s comp representation:

Whenever a signed no. has a 1 in the sign bit & all 0‘s for the magnitude bits, the decimal equivalent is -2^n , where n is the no of bits in the magnitude .

Ex: 1000= -8 & 10000=-16

Signed binary numbers:

Decimal	Sign 2‘s comp form	Sign 1‘s comp form	Sign mag form
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0011	0011	0011
+0	0000	0000	0000
-0	--	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
8	1000	--	--

Binary Codes

Binary codes are codes which are represented in binary system with modification from the original ones. It may be classified as follows:

- Weighted Binary Systems
- Non Weighted Codes

Weighted Binary Systems

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. The binary counting sequence is an example.

(i)8421 Code/BCD Code

The BCD (Binary Coded Decimal) is a straight assignment of the binary equivalent. It is possible to assign weights to the binary bits according to their positions. The weights in the BCD code are 8,4,2,1.

Example: The bit assignment 1001, can be seen by its weights to represent the decimal 9 because:

1x8+0x4+0x2+1x1 = 9

(ii)2421 Code

This is a weighted code, its weights are 2, 4, 2 and 1. A decimal number is represented in 4-bit form and the total four bits weight is 2 + 4 + 2 + 1 = 9. Hence the 2421 code represents the decimal numbers from 0 to 9.

(iii)5211 Code

This is a weighted code, its weights are 5, 2, 1 and 1. A decimal number is represented in 4-bit form and the total four bits weight is 5 + 2 + 1 + 1 = 9. Hence the 5211 code represents the decimal numbers from 0 to 9.

Non Weighted Codes

Non Weighted binary codes are those which do not obey the positional weighting principles, each position of the number does not represent a specific weight.
Examples

(i)Excess-3
Excess-3 is a non weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

(ii)Gray Code
The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code. In digital Gray code has got a special place.

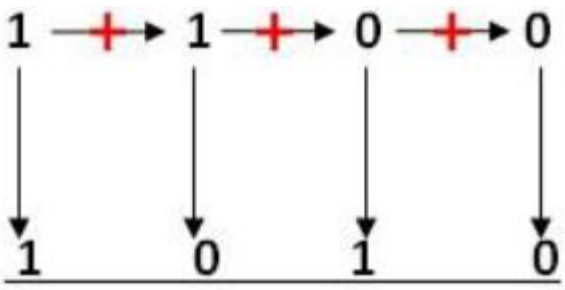
Binary to gray code Converter

1. First, write the binary code and copy down MSB. The MSB of gray and binary code are same.

2. Then, add MSB and next lower significant bit and write down the addition of them.

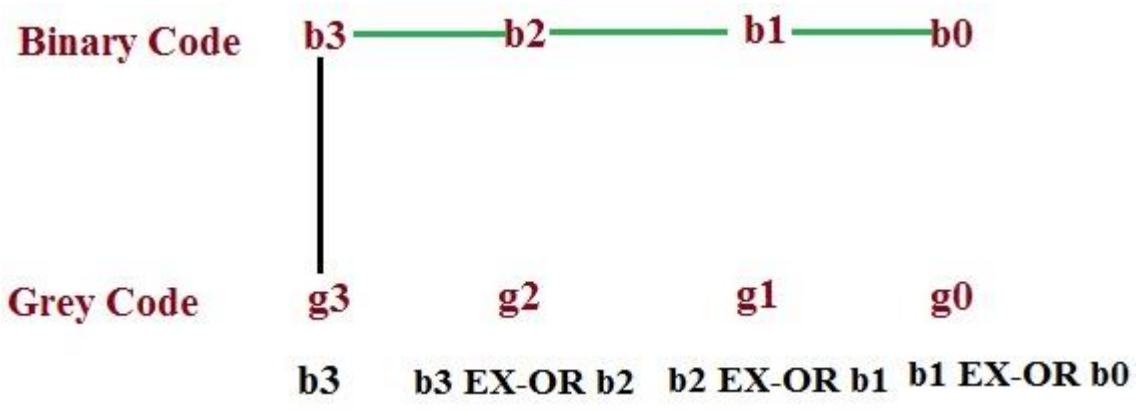
3. Continue the same process for all bits.

E.g. Find gray code for (1100)2



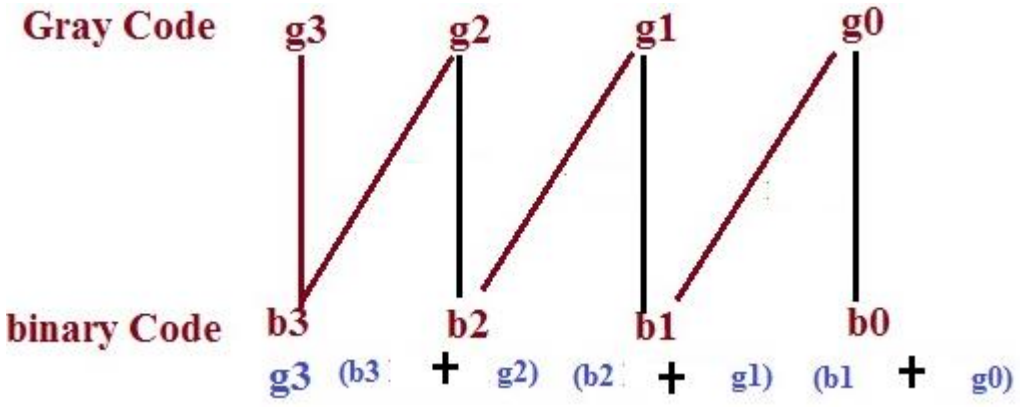
Binary to Gray Code

Let assume the Binary code digits be bo, b1, b2, b3 whereas the particular Gray Code can be attained based on the following concept.



Gray to Binary Code Converter

Let assume the Gray Code digits g3, g2, g1, g0 whereas the particular Binary code digits are bo, b1, b2, b3 can be attained based on the following concept.



Example − Convert Gray code 100111 into Binary number.

So, according above algorithm,

$b_5=g_5=1=1$

$b_4=g_5+g_4=1+0=1$

$b_3=b_4+g_3=1+0=1$

$b_2=b_3+g_2=1+1=0$ (discard carry 1)

$b_1=b_2+g_1=0+1=1$

$b_0=b_1+g_0=1+1=0$ (discard carry 1)

So, Binary number will be 111010.

Error Detecting and Correction Codes

For reliable transmission and storage of digital data, error detection and correction is required. Below are a few examples of codes which permit error detection and error correction after detection.

Error Detecting Codes

When data is transmitted from one point to another, like in wireless transmission, or it is just stored, like in hard disks and memories, there are chances that data may get corrupted. To detect these data errors, we use special codes, which are error detection codes.

Parity

In parity codes, every data byte, or nibble (according to how user wants to use it) is checked if they have even number of ones or even number of zeros. Based on this information an additional bit is appended to the original data. Thus if we consider 8-bit data, adding the parity bit will make it 9 bit long.
At the receiver side, once again parity is calculated and matched with the received parity (bit 9), and if they match, data is ok, otherwise data is corrupt.
There are two types of parity:

- Even parity: Checks if there is an even number of ones; if so, parity bit is zero. When the number of ones is odd then parity bit is set to 1.
- Odd Parity: Checks if there is an odd number of ones; if so, parity bit is zero. When number of ones is even then parity bit is set to 1.

Error-Correcting Codes

Error correcting codes not only detect errors, but also correct them. This is used normally in Satellite communication, where turn-around delay is very high as is the probability of data getting corrupt.
ECC (Error correcting codes) are used also in memories, networking, Hard disk, CDROM, DVD etc.

Hamming Code

The hamming code technique, which is an **error-detection and error-correction technique**, was proposed by R.W. Hamming. Whenever a data packet is transmitted over a network, there are possibilities that the data bits may get lost or damaged during transmission.

Let's understand the Hamming code concept with an example:

Let's say you have received a 7-bit Hamming code which is 1011011.

First, let us talk about the redundant bits.

The **redundant bits** are some extra binary bits that are not part of the original data, but they are generated & added to the original data bit. All this is done to ensure that the data bits don't get damaged and if they do, we can recover them.

Now the question arises, how do we determine the number of redundant bits to be added?

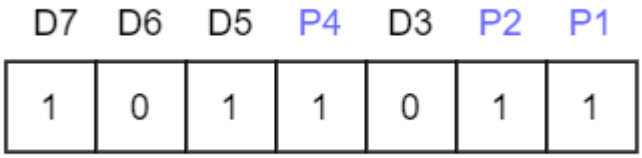
We use the formula, 2r >= m+r+1; where r = redundant bit **&** m = data bit.

From the formula we can make out that there are 4 data bits and 3 redundancy bits**,** referring to the received 7-bit hamming code**.**

To proceed further we need to know about parity bit, which is a bit appended to the data bits which ensures that the total number of 1's are even (even parity) or odd (odd parity).

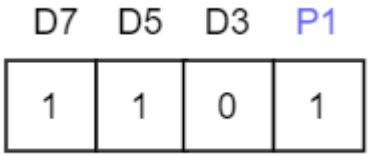
While checking the parity, if the total number of 1's are odd then write the value of parity bit P1**(or** P2 etc.) as 1 (which means the error is there) and if it is even then the value of parity bit is 0 (which means no error).

As we go through the example, the first step is to identify the bit position of the data & all the bit positions which are powers of 2 are marked as parity bits (e.g. 1, 2, 4, 8, etc.). The following image will help in visualizing the received hamming code of 7 bits.



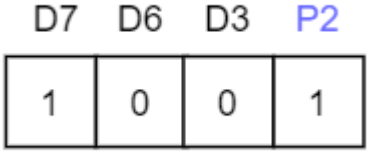
First, we need to detect whether there are any errors in this received hamming code.

Step 1: For checking parity bit **P1**, use check one and skip one method, which means, starting from P1 and then skip P2, take D3 then skip P4 then take D5, and then skip D6 and take D7, this way we will have the following bits,



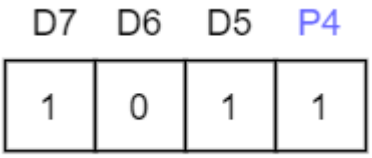
As we can observe the total number of bits are odd so we will write the value of parity bit as **P1 = 1**. This means **error is there**.

Step 2: Check for P2 but while checking for P2, it will give us the following data bits. But remember since we are checking for P2, so we have to start our count from P2 (P1 should not be considered).



As we can observe that the number of 1's are even, then we will write the value of **P2 = 0**. This means **there is no error**.

Step 3: Check for P4 but while checking for P4, it will give us the following data bits. But remember since we are checking for P4, so we have started our count from P4(P1 & P2 should not be considered).

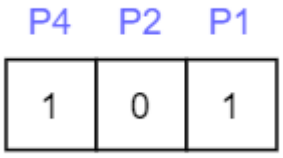


As we can observe that the number of 1's are odd, then we will write the value of **P4 = 1**. This means the error is there.

So, from the above parity analysis, P1 & P4 are not equal to 0, so we can clearly say that the received hamming code has errors.

Since we found that received code has an error, so now we must correct them. To correct the errors, use the following steps:

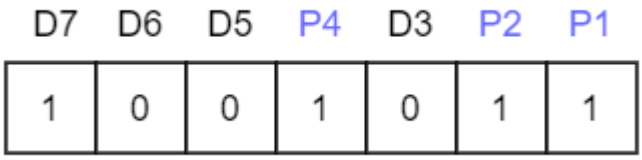
Now the error word E will be:



Now we have to determine the decimal value of this error word **101** which is **5 (2^2 *1 + 2^1 * 0 + 2^0 *1 = 5)**.

We get **E = 5**, which states that the error is in the fifth data bit. To correct it, just invert the fifth data bit.

So the correct data will be:



Logic Level

Put simply, a logic level is a specific voltage or a state in which a signal can exist. We often refer to the two states in a digital circuit to be ON or OFF. Represented in binary, an ON translates to a binary 1, and an OFF translates to a binary 0. In Arduino, we call these signals HIGH or LOW, respectively. There are several different technologies that have evolved over the past 30 years in electronics to define the various voltage levels.

Logic 0 or Logic 1

Digital electronics rely on binary logic to store, process, and transmit data or information. Binary Logic refers to one of two states -- ON or OFF. This is commonly translated as a binary 1 or binary 0. A binary 1 is also referred to as a HIGH signal and a binary 0 is referred to as a LOW signal.

The strength of a signal is typically described by its voltage level. How is a logic 0 (LOW) or a logic 1 (HIGH) defined? Manufacturers of chips generally define these in their spec sheets. The most common standard is TTL or Transistor-Transistor Logic.

Active-Low and Active-High

When working with ICs and microcontrollers, you'll likely encounter pins that are active-low and pins that are active-high. Simply put, this just describes how the pin is activated. If it's an active-low pin, you must "pull" that pin LOW by connecting it to ground. For an active high pin, you connect it to your HIGH voltage (usually 3.3V/5V).

For example, let's say you have a shift register that has a chip enable pin, CE. If you see the CE pin anywhere in the datasheet with a line over it like this, $\overline{CE}$, then that pin is active-low. The CE pin would need to be pulled to GND in order for the chip to become enabled. If, however, the CE pin doesn't have a line over it, then it is active high, and it needs to be pulled HIGH in order to enable the pin.

Many ICs will have both active-low and active-high pins intermingled. Just be sure to double check for pin names that have a line over them. The line is used to represent NOT (also known as bar). When something is NOTTED, it changes to the opposite state. So if an active-high input is NOTTED, then it is now active-low.

Chapter-3

BOOLEAN ALGEBRA AND LOGIC GATES

The English mathematician George Boole (1815-1864) sought to give symbolic form to Aristotle‘s system of logic. Boole wrote a treatise on the subject in 1854, titled An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities, which codified several rules of relationship between mathematical quantities limited to one of two possible values: true or false, 1 or 0. His mathematical system became known as Boolean algebra.

Boolean Algebra

Boolean Algebra is Algebra of logic. This is an algebra that deals with logical propositions, which are either true or false. This algebra is suitable for binary number system and is very useful in designing digital circuits, which operates under logic.

For example,

A+A =A, not 2A

Also 1+1 = 1 not 2 as it is logical expression.

One can visualize this as,

TRUE+TRUE=TRUE

FALSE+FALSE=FALSE

A Boolean algebraic expression is composed of variables, constants and operators. The **variables are** generally represented by the letters of the Alphabet (say A) which can have two possible values 1 or 0. The interpretation of 1 may be that the variable is presented input signal is ON, is TRUE, and is a positive voltage. If A is 0, then it mean that the variable is absent, input signal if OFF, is FALSE, and is a negative voltage. Similarly, the **Boolean Constant** can have any two values, either 1 or 0.

Boolean Operators are used in Boolean Algebra where a mathematical function called **Boolean Function** is constructed.

These operators are the:

(i) Symbols PLUS (+) meaning an OR operation
(ii) DOT (.) meaning and AND operation
(iii) BAR ‘A read as COMPLEMENT meaning a NOT operation.

Postulates of Boolean Algebra

A set of Boolean postulates are the following

(a) 0.0=0
(b) 1+1=1
(c) 0+0=0
(d) 1.1=1
(e) 1.0=0.1=0
(f) 1+0=0+1=1

The realization of any Boolean Expression can be obtained with the help of a table called **TRUTH TABLE**. To simplify a Boolean Expression, one requires certain laws of Boolean Algebra.

Laws of Boolean Algebra and their Truth Table

a) Commutative Law

i) A+B=B+A
ii) A.B=B.A

Truth Table

A	*B*	*A+B*	*A.B*	*B+A*	*B.A*
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	0	1	0
1	1	1	1	1	1

b) Associative Law

i) A+(B+C)=(A+B)+C

ii) A.(B.C)=(A.B).C

Truth Table

A	*B*	*C*	*B+C*	*A+B*	*A+(B+C)*	*(A+B)+C*	*B.C*	*A.B*	*A.(B.C)*	*(A.B).C*
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	0	0	0	0
0	1	0	1	1	1	1	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0
1	0	0	0	1	1	1	0	0	0	0
1	0	1	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1

C) Distributive Law

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

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

Truth Table

A	B	C	B+C	A.B	A.C	A.(B+C)	(A.B+A.C)	B.C	A+B	A+C	A+(B.C)	(A+B).(A+C)
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	1	1	1
1	0	1	1	0	1	1	1	0	1	1	1	1
1	1	0	1	1	0	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1

De Morgan's Theorem

This is a useful theorem in Boolean Algebra which states how to complement a Boolean expression. They allow us to convert back and forth from minterm to maxterm forms of Boolean expression. It helps to eliminate long over-bars that cover several variables.

a) First Theorem

(A + B)' = A'. B' (For two variables)

In general (A + B+ C+……)' = A' . B' . C'…. (For many variables)

b) Second Theorem

(A . B)' = A' + B'

In general (A . B . C)' = A' + B' + C' +…….

Truth Table

A	B	$\overline{A}$	$\overline{B}$	AB	A+B	$\overline{A+B}$	$\overline{A.B}$
0	0	1	1	0	0	1	1
0	1	1	0	0	1	0	0
1	0	0	1	0	1	0	0
1	1	0	0	1	1	0	0

Representation of Boolean Algebra & De Morgan's Theorem through Logic Circuits

There are nine basic identities commonly used in converting complex Boolean expression to their simple forms. A Boolean identity usually consists of one variable and one constant equates two expressions, which are equal for all possible combinations of the variables. The equivalence of two expressions is presented through Truth Table and representative circuits symbol.

Double Complementation

A double complementation is the complementation of a single complement. The single complement is called **NOT** operation.

Expression : (A')' = A

Truth Table

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

AND Function Identities

The identity states that A.1 = A

Corresponding **Truth Table** and circuit are given below

A	1	Y
0	1	0
1	1	1

The Boolean expression for the output is A.1, which is read as 'A and 1". In general this is A.B. Similarly, there are other identities using AND operator.

A.0=0

A	0	Y
0	0	0
1	0	0

A.A=A

A	A	Y
0	0	0
1	1	1

$A.\overline{A}=0$

A	$\overline{A}$	Y
0	1	0
1	0	0

OR Function Identities

This identity can be realized with one input permanently tied to logic 1 and the other is varying. A+1=1

A	1	Y
0	1	1
1	1	1

Other three identities using OR operator are

A+0 = A

A	0	Y
0	0	0
1	0	1

$A+A=A$

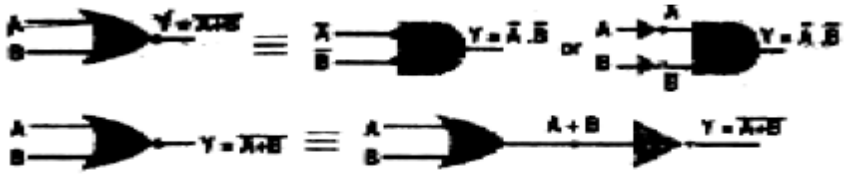
A	0	Y
0	0	0
1	0	1

$A+\overline{A}=1$

A	0	Y
0	0	0
1	0	1

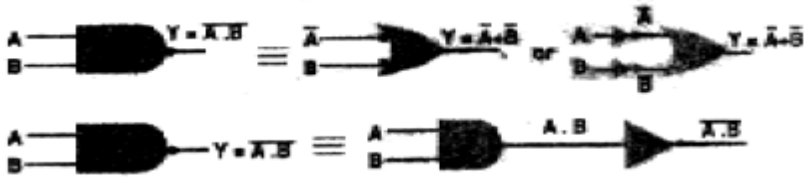
The Circuit Representation of De Morgan's Theorem

1. A+B = A.B known as NOR circuit, which is equivalent to Bubbled input AND circuit.



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

2. $\overline{A.B}=\overline{A}+\overline{B}$ known as NAND circuit, which is equivalent to Bubbled input OR circuit.



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

Absorption law

A +AB =A

Solution: A(1+B) =A

Algebraic Manipulation (Minimization of Boolean function)

• Boolean algebra is a useful tool for simplifying digital circuits.

• Why do it? Simpler can mean cheaper, smaller, faster.

• Example: Simplify F = x'yz + x'yz' + xz.

F= x'yz + x'yz' + xz

= x'y(z+z') + xz

= x'y•1 + xz

= x'y + xz

• Example: Prove x'y'z' + x'yz' + xyz' = x'z' + yz'

• Proof: x'y'z'+ x'yz'+ xyz'

= x'y'z' + x'yz' + x'yz' + xyz'

= x'z'(y'+y) + yz'(x'+x)

= x'z'•1 + yz'•1

= x'z' + yz'

Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions. Identical functions will have exactly the same canonical form.

• Minterms and Maxterms

• Sum-of-Minterms and Product-of- Maxterms

• Product and Sum terms

• Sum-of-Products (SOP) and Product-of-Sums (POS)

Definitions

Literal: A variable or its complement

Product term: literals connected by •

Sum term: literals connected by +

Minterm: a product term in which all the variables appear exactly once, either complemented or un-complemented.

Maxterm: a sum term in which all the variables appear exactly once, either complemented or un-complemented.

Canonical form: Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

Minterm

• Represents exactly one combination in the truth table.

• Denoted by mj, where j is the decimal equivalent of the minterm's corresponding binary combination (bj).
• A variable in mj is complemented if its value in bj is 0, otherwise is uncomplemented.
Example: Assume 3 variables (A, B, C), and j=3. Then, bj = 011 and its corresponding minterm is denoted by mj = A'BC

Maxterm
• Represents exactly one combination in the truth table.
• Denoted by *Mj*, where *j* is the decimal equivalent of the maxterm's corresponding binary combination *(bj*).
• A variable in *Mj* is complemented if its value in *bj* is 1, otherwise is uncomplemented.
Example: Assume 3 variables (A, B, C), and *j*=3. Then, b*j* = 011 and its corresponding maxterm is denoted by M*j* = A+B'+C'

Truth Table notation for Minterms and Maxterms
• Minterms and Maxterms are easy to denote using a truth table.
Example: Assume 3 variables x,y,z (order is fixed)

x	y	z	Minterm	Maxterm
0	0	0	$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1	$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0	$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1	$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0	$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1	$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0	$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1	$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms

• Every function F() has two canonical forms:
– Canonical Sum-Of-Products (sum of minterms)
– Canonical Product-Of-Sums (product of maxterms)

Canonical Sum-Of-Products:
The minterms included are those mj such that F( ) = 1 in row j of the truth table for F( ).

Canonical Product-Of-Sums:
The maxterms included are those Mj such that F( ) = 0 in row j of the truth table for F( ).

Example
Consider a Truth table for f1(a,b,c) shown below.

a	b	c	f_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

The canonical sum-of-products form for f1 is
f1(a,b,c) = m1 + m2 + m4 + m6
= a'b'c + a'bc' + ab'c' + abc'
The canonical product-of-sums form for f1 is f1(a,b,c) = M0 • M3 • M5 • M7
= (a+b+c)•(a+b'+c')• (a'+b+c')•(a'+b'+c').
• Observe that: mj = Mj'

Shorthand: Σ and Π

• f1(a,b,c) = Σ m(1,2,4,6), where Σ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m1, m2, m4, and m6.

• f1(a,b,c) = Π M(0,3,5,7), where Π indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M0, M3, M5, and M7.

• Since mj = Mj' for any *j*,

Σ m(1,2,4,6) = Π M(0,3,5,7) = f1(a,b,c)

Conversion between Canonical Forms

• Replace Σ with Π (or *vice versa*) and replace those *j*'s that appeared in the original form with those that do not.

• Example:
f1(a,b,c)= a'b'c + a'bc' + ab'c' + abc'
= m1 + m2 + m4 + m6
= Σ(1,2,4,6)
= Π(0,3,5,7)
= (a+b+c)•(a+b'+c')•(a'+b+c')•(a'+b'+c')

Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.
There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

F1 = y' + xy + x'yz'

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

F2 = x(y' + z)(x' + y + z')

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

Conversion of SOP from standard to canonical form

Example-1.

Express the Boolean function F = A + B'C as a sum of minterms.

Solution:

The function has three variables: A, B, and C. The first term A is missing two variables; therefore, A = A(B + B') = AB + AB'

This function is still missing one variable, so

A = AB(C + C') + AB' (C + C')

= ABC + ABC' + AB'C + AB'C'

The second term B'C is missing one variable; hence,

B'C = B'C(A + A') = AB'C + A'B'C

Combining all terms, we have

F = A + B'C

= ABC + ABC' + AB'C + AB'C'+ A'B'C

But AB'C appears twice, and according to theorem (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

F = A'B'C + AB'C + AB'C + ABC' + ABC

= m1 + m4 + m5 + m6 + m7

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

F(A, B, C) = Σm (1, 4, 5, 6, 7)

Example-2.

Express the Boolean function F = xy + x'z as a product of maxterms.

Solution:

First, convert the function into OR terms by using the distributive law:

F = xy + x'z = (xy + x')(xy + z)

= (x + x')(y + x')(x + z)(y + z)

= (x'+ y)(x + z)(y + z)

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

x'+ y = x' + y + zz' = (x' + y + z)(x' + y + z')

x + z = x + z + yy' = (x + y + z)(x + y' + z) y + z = y + z + xx' = (x + y + z)(x' + y + z)

Combining all the terms and removing those which appear more than once, we finally obtain

F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z)

F= M0M2M4M5

A convenient way to express this function is as follows: F(x, y, z) = πM(0, 2, 4, 5)

The product symbol, π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Functional completeness

In logic, a **functionally complete** set of logical connectives or Boolean operators is one which can be used to express all possible truth tables by combining members of the set into a Boolean expression. A well-known complete set of connectives is { AND, NOT }, consisting of binary conjunction and negation. Each of the singleton sets { NAND } and { NOR } is functionally complete.

A gate or set of gates which is functionally complete can also be called a universal gate / gates.

A functionally complete set of gates may utilise or generate 'garbage bits' as part of its computation which are either not part of the input or not part of the output to the system.

In a context of propositional logic, functionally complete sets of connectives are also called **(expressively) adequate**.

From the point of view of digital electronics, functional completeness means that every possible logic gate can be realized as a network of gates of the types prescribed by the set. In particular, all logic gates can be assembled from either only binary NAND gates, or only binary NOR gates.