DEPARTMENT OF ELECTRICAL ENGINEERING

DIGITAL CIRCUITS AND DESIGN (4 credit)

Course Code: BEC1405

(5^{TH} SEMESTER)

SYLLABUS

DIGITAL CIRCUITS AND DESIGN (3-1-0)

MODULE-I

Number system& codes: Binary Number base conversion, Octal &hexadecimalnumbers, complements, signed binary numbers, binary codes-BCD codes, gray codes, ASCIICharacter Code, Codes for serial data transmission &storage.

Boolean Algebra & Logic gates: Axiomatic definition of boolean Algebra .Property of BooleanAlgebra, boolean functions, Canonical & standard form; min terms & max terms, standard forms; Digital Logic Gates, Multiple inputs.

MODULE-II

Gate level Minimization: The Map Method, K Map up to five variables, Product of Sum simplification, Sum of Product simplification, Don't care conditions. NAND and NORImplementation, AND-OR inverter, OR-AND inverter implementation, Ex-OR Function, parity generation& checking, Hardware Description Language (HDL).

Combinational Logic: Combinational Circuits, Analysis &Design procedure; Binary Adder-subtractor, Decimal Adder, Binary Multiplier, Magnitude comparator, Multiplexers and demultiplexers, Decoders, Encoders, Multipliers, Combinational Circuits design

MODULE-III

Synchronous Sequential logic: Sequential Circuit, latches, Flip-flop, Analysis of Clocked Sequential circuits, HDL for Sequential Circuits, State Reduction &Assignment, Design procedure.Register &Counters: Shift Register, Ripple Counters, Synchronous Counter, Asynchronous Counter,Ring Counters, Module-n Counters, HDL for Register &Counters .

MODULE-IV

Memory & Programmable logic: Random Access Memory (RAM), Memory , Decoding, Errordetection & correction, Read only Memory, Programmable logic array,Sequential ProgrammableDevices.

Register Transfer levels: Register transfer level notion, Register transfer level in HDL, Algorithm,State machine, Design Example,. HDL Description of Design, Examples, Binary Multiplier, HDL Description,

Digital Integrated logic Circuits: RTL, DTL, TTL, ECL, MOS & C-MOS Logic circuits,. Switchlevelmodeling with HDL

BOOKS

[1]. Digital Design,3rd edition by M. Morris Mano, Pearson Education

[2]. Digital Design-Principle& practice, 3rd edition by John F. Wakerley, Pears

Disclaimer

This document does not claim any originality and cannot be used as a substitute for prescribed textbooks. The information presented here is merely a collection by the committee members for their respective teaching assignments. Various sources as mentioned at the end of the document as well as freely available material from internet were consulted for preparing this document. The ownership of the information lies with the respective authors or institutions. Further, this document is not intended to be used for commercial purpose and the committee members are not accountable for any issues, legal or otherwise, arising out of use of this document. The committee members make no representations or warranties with respect to the accuracy or completeness of the contents of this document and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. The committee members shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

MODULE-I

NUMBER SYSTEMS

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 usbecause it is a tool that we use every day. Examining some of its characteristics will help us tobetter understand the other systems. In the next few pages we shall introduce four numericalrepresentation 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

3.14_{10}
52_{10}
1024_{10}
64000_{10}

Binary System

In the binary system, there are only two symbols or possible digit values, 0 and 1. This base-2 system canbe 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. Binaryquantities 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.

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 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.

Hexadecimal to Decimal Conversion

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

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.

$11011_2 = 2_4 + 2_3 + 0_1 + 2_1 + 2_0 = 16 + 8 + 0 + 2 + 1 = 27_{10}$

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.**

Binary Codes

Binary codes are codes which are represented in binary system with modification from the original ones. Below we will be seeingthe following:

- **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.

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

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.

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.

Reflective Code

A code is said to be reflective when code for 9 is complement for the code for 0, and so is for 8 and 1 codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211, and excess-3 are reflective, whereas the 8421 code is not.

Excess-3 Code

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).

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 are 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 Graycode has got a special place.

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. Normally in networking chips (ASIC), we have 2 Error detection bits and 1 Error correction bit.

Hamming Code

Hamming code adds a minimum number of bits to the data transmitted in a noisy channel, to be able to correct every possible one-bit error. It can detect (not correct) two bits errors and cannot distinguish between 1-bit and 2-bits inconsistencies. It can't – in general – detect 3(or more)-bits errors The idea is that the failed bit position in an n-bit string (which we'll call X) can be represented in binary with log2(n) bits, hence we'll try to get it adding just log2(n) bits.

ASCII Code

ASCII stands for American Standard Code for Information Interchange. It has become a world standard alphanumeric code for microcomputers and computers. It is a 7-bit code representing 27 = 128 different characters. These characters represent 26 upper case letters (A to Z), 26 lowercase letters (a to z), 10 numbers (0 to 9), 33 special characters and symbols and 33 control characters.

BOOLEAN ALGEBRA AND LOGIC GATES

The English mathematician George Boole (1815-1864) sought to give symbolic formto Aristotle‘s system of logic. Boole wrote a treatise on the subject in 1854, titled AnInvestigation of the Laws of Thought, on Which Are Founded the Mathematical Theories ofLogic and Probabilities, which codified several rules of relationship between mathematicalquantities limited to one of two possible values: true or false, 1 or 0. His mathematicalsystembecame known as Boolean algebra. All arithmetic operations performed with Booleanquantities have but one of two possibleOutcomes: either 1 or 0. There is no such thing as ‖2‖ or ‖-1‖ or ‖1/2‖ in the Boolean world.

It is a world in which all other possibilities are invalid by fiat. As one might guess, this is notthe kind of math you want to use when balancing a check book or calculating current througha resistor.

However, Claude Shannon of MIT fame recognized how Boolean algebra could be applied toon-and-off circuits, where all signals are characterized as either ‖high‖ (1) or ‖low‖ (0).His1938 thesis, titled A Symbolic Analysis of Relay and Switching Circuits, put Boole‘stheoretical work to use in a way Boole never could have imagined, giving us a powerfulmathematical tool for designing and analyzing digital circuits.

Like ‖normal‖ algebra, Boolean algebra uses alphabetical letters to denote variables.Unlike ‖normal‖ algebra, though, Boolean variables are always CAPITAL letters, neverlowercase.

Because they are allowed to possess only one of two possible values, either 1 or 0, each andevery variable has a complement: the opposite of its value. For example, if variable ‖A‖ has avalue of 0, then the complement of A has a value of 1. Boolean notation uses a bar above thevariable character to denote complementation, like this:

If: $A=0$

Then: $\overline{A}=1$

If: $A=1$

Then: $\overline{A}=0$

In written form, the complement of ‖A‖ denoted as ‖A-not‖ or ‖A-bar‖. Sometimes a ‖prime‖symbol is used to represent complementation. For example, A‘ would be the complement ofA, much the same as using a prime symbol to denote differentiation in calculus rather thanthe fractional notation dot. Usually, though, the ‖bar‖ symbol finds more wide spread use thanthe ‖prime‖ symbol, for reasons that will become more apparent laterin this chapter.

Boolean Arithmetic:

Let us begin our exploration of Boolean algebra by adding numbers together:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 1

The first three sums make perfect sense to anyone familiar with elementary addition. TheLast sum, though, is quite possibly responsible for more confusion than any other singlestatement in digital electronics, because it seems to run contrary to the basic principles ofmathematics.Well, it does contradict principles of addition for real numbers, but not for Boolean numbers.Remember that in the world of Boolean algebra, there are only two possible values for anyquantity and for any arithmetic operation: 1 or 0. There is no such thing as ‖2‖ within thescope of Boolean values. Since the sum ‖1 + 1‖ certainly isn‘t 0, it must be 1 by process ofelimination.

Principle of Duality:

It states that every algebraic expression is deducible from the postulates of Booleanalgebra,and it remains valid if the operators & identity elements are interchanged. If theinputs of a NOR gate are inverted we get a AND equivalent circuit. Similarly when the inputsof a NAND gate are inverted, we get a OR equivalent circuit.This property is called duality.

Theorems of Boolean algebra

The theorems of Boolean algebra can be used to simplify many a complex Booleanexpression and also to transform the given expression into a more useful and meaningful

equivalent expression. The theorems are presented as pairs, with the two theorems in a given pair being the dual of each other. These theorems can be very easily verified by the method of 'perfect induction'. According to this method, the validity of the expression is tested for all possible combinations of values of the variables involved. Also, since the validity of thetheorem is based on its being true for all possible combinations of values of variables, there is no reason why a variable cannot be replaced with its complement, or vice versa, without disturbing the validity. Another important point is that, if a given expression is valid, its dual will also be valid.

Theorem 1 (Operations with '0' and '1')

(a) 0.X = 0 and (b) 1+X= 1

Where X is not necessarily a single variable – it could be a term or even a large expression.

Theorem 1(a) can be proved by substituting all possible values of X, that is, 0 and 1, into the given expression and checking whether the LHS equals the RHS:

• For X = 0, LHS = 0.X = 0.0 = 0 = RHS.

• For X= 1, LHS = 0.1 = 0 = RHS.

Thus, 0.X =0 irrespective of the value of X, and hence the proof.

Theorem 1(b) can be proved in a similar manner. In general, according to theorem 1,

0. **(Boolean expression) = 0 and 1+ (Boolean expression) =1.**
1. **For example: 0. (A.B+B.C +C.D) = 0 and 1+ (A.B+B.C +C.D) = 1, where A, B and C are**

Boolean variables.

Theorem 2 (Operations with ‘0‘ and ‘1‘)

(a) 1.X = X and (b) 0+X = X

where X could be a variable, a term or even a large expression. According to this theorem,ANDing a Boolean expression to ‘1‘ orORing ‘0‘ to it makes no difference to the expression:

• For X = 0, LHS = 1.0 = 0 = RHS.

• For X = 1, LHS = 1.1 = 1 = RHS.

Also,

1. (Boolean expression) = Boolean expression and 0 + (Boolean expression) = Boolean expression.

For example,

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

Theorem 3 (Idempotent or Identity Laws)

(a) X.X.X……X = X and (b) X+X+X +⋯+X = X

Theorems 3(a) and (b) are known by the name of idempotent laws, also known as identity laws.

Theorem 3(a) is a direct outcome of an AND gate operation, whereas theorem 3(b) representsan OR gate operation when all the inputs of the gate have been tied together. The scope ofidempotent laws can be expanded further by considering X to be a term or an expression. Forexample, let us apply idempotent laws to simplify the following Boolean expression:

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

Theorem 4 (Complementation Law)

(a) X_X = 0 and (b) X+X = 1

According to this theorem, in general, any Boolean expression when ANDed to its complement yields a '0' and when ORed to its complement yields a '1', irrespective of the complexity of the expression:

- For $X=0$, $\overline{X}=1$. Therefore, $X.\overline{X}=0.1=0$.
- For $X=1$, $\overline{X}=0$. Therefore, $X.\overline{X}=1.0=0$.

Hence, theorem 4(a) is proved. Since theorem 4(b) is the dual of theorem 4(a), its proof is implied.

The example below further illustrates the application of complementation laws:

$$(A+B.C)(\overline{A+B.C})=0 \quad \text{and} \quad (A+B.C)+(\overline{A+B.C})=1$$

Theorem 5 (Commutative property)

Mathematical identity, called a ‖property‖ or a ‖law,‖ describes how differingvariables relate to each other in a system of numbers. One of these properties is known as the commutative property, and it applies equally to addition and multiplication.

In essence,the commutative property tells us we can reverse the order of variables that are either addedtogether or multiplied together without changing the truth of the expression:

Commutative property of addition

A + B = B + A

Commutative property of multiplication

AB = BA

Theorem 6 (Associative Property)

The Associative Property,again applying equally well to addition and multiplication.

This property tells us we can associate groups of added or multiplied variables together with

parentheses without altering the truth of the equations.

Associative property of addition

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

Associative property of multiplication

A (BC) = (AB) C

Theorem 7 (Distributive Property)

The Distributive Property, illustrating how to expand a Boolean expression formed by

the product of a sum, and in reverse shows us how terms may be factored out of Boolean

sums-of-products:

Distributive property

A (B + C) = AB + AC

Theorem 8 (Absorption Law or Redundancy Law)

(a) X+X.Y = X and (b) X.(X+Y) = X

The proof of absorption law is straightforward:

X+X.Y = X. (1+Y) = X.1 = X

Theorem 8(b) is the dual of theorem 8(a) and hence stands proved.

The crux of this simplification theorem is that, if a smaller term appears in a larger term, then

the larger term is redundant. The following examples further illustrate the underlying

concept:

$$A + A.\overline{B} + A.\overline{B}.\overline{C} + A.\overline{B}.C + \overline{C}.B.A = A$$

and

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

De-Morgan‘s First Theorem

It States that —The complement of the sum of the variables is equal to the product of the complement of each variable . This theorem may be expressed by the following Boolean expression.

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

De-Morgan‘s Second Theorem

It states that the —Complement of the product of variables is equal to the sum of complementsof each individual variables‖. Boolean expression for this theorem is

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

Boolean Function

Boolean functions are represented in various forms. The two popular forms are *truth tables* and *Venn diagrams*. Truth tables represent functions in a tabular form, while Venn diagrams provide a graphic representation. In addition, there are two algebraic representations known as the *standard* (or *normal*) *form* and the *canonical form*.

Draw the truth table for $Z = AB' + A'C + A'B'C$.

There are three component variables: A, B, and C. Hence, there will be $2^3 = 8$ combinations of values. These eight combinations are shown in the first three columns of Table 2.2. These combinations correspond to binary numbers 000 through 111, or $(0)_{10}$ through $(7)_{10}$.

To evaluate Z in the example function, knowing the values for A, B, and C at each row of the truth table (Table 2.2), we would first generate values for A' and B' and then generate values for AB', $A'C$ and $A'B'C$ by ANDing the values in the appropriate columns for each row. Finally, we would derive the value of Z by ORing the values in the last three columns for each row. Note that evaluating $A'B'C$ corresponds to ANDing A' and B' values, followed by ANDing the value of C. Similarly, if more than two values are to be ORed, they are ORed

Z=AB'+A'C+A'B'C'

A	B	C	A'	B'	AB'	$A'C$	$A'B'C$	Z
0	0	0	1	1	0	0	0	0
0	0	1	1	1	0	1	1	1
0	1	0	1	0	0	0	0	0
0	1	1	1	0	0	1	0	1
1	0	0	0	1	1	0	0	1
1	0	1	0	1	1	0	0	1
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

Canonical Form of Boolean Expressions

An expanded form of Boolean expression, where each term contains all Boolean variables in their true or complemented form, is also known as the canonical form of the expression. As an illustration, $f(A.B, C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.B.C$ is a Boolean function of three variables expressed in canonical form. This function after simplification reduces to $\overline{A}.\overline{B} + A.B.C$ and loses its canonical form.

Minterms and Maxterms

A minterm is the product of N distinct literals where each literal occurs exactly Anyboolean expression may be expressed in terms of either minterms or maxterms. To do this we must first define the concept of a literal. A literal is a single variable within a term which may or may not be complemented. For an expression with N variables, minterms and maxterms are defined as follows :

- **once.**
- **A maxterm is the sum of N distinct literals where each literal occurs exactly once.**

Product-of-Sums Expressions

A product-of-sums expression contains the product of different terms, with each term

being either a single literal or a sum of more than one literal. It can be obtained from the truthtable by considering those input combinations that produce a logic ‘0‘ at the output. Eachsuch input combination gives a term, and the product of all such terms gives the expression.

Different terms are obtained by taking the sum of the corresponding literals. Here ‘0‘ and‘1‘respectively mean the uncomplemented and complemented variables, unlike sum-of products expressions where ‘0‘ and ‘1‘ respectively mean complemented and uncomplementedvariables.

Since each term in the case of the product-of-sums expression is going to be the sum ofliterals, this implies that it is going to be implemented using an OR operation. Now, an ORgate produces a logic ‘0‘ only when all its inputs are in the logic ‘0‘ state, which means thatthe first term corresponding to the second row of the truth table will be A+B+C. The productof-sums Boolean expression for this truth table is given by Transforming the given productof-sums expression into an equivalent sum-of-products expression is a straightforwardprocess. Multiplying out the given expression and carrying out the obvious simplificationprovides the equivalent sum-of-products expression:

A given sum-of-products expression can be transformed into an equivalent product-of sumsexpression by (a) taking the dual of the given expression, (b) multiplying out differenttermsto get the sum-of products form, (c) removing redundancy and (d) taking a dual to get the

equivalent product-of-sums expression. As an illustration, let us find the equivalent productof sums expression of the sum-of products expression

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

The dual of the given expression = $(A+B).(\overline{A}+\overline{B})$:

$$(A+B).(\overline{A}+\overline{B}) = A.\overline{A}+A.\overline{B}+B.\overline{A}+B.\overline{B} = 0+A.\overline{B}+B.\overline{A}+0 = A.\overline{B}+\overline{A}.B$$

The dual of $(A.\overline{B}+\overline{A}.B) = (A+\overline{B}).(\overline{A}+B)$. Therefore

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

Digital Logic Gates

The basic logic gates are AND, OR, NAND, NOR, XOR, INV, and BUF. The last two are notstandard terms; they stand for 'inverter' and 'buffer', respectively. The symbols for thesegates and their corresponding Boolean expressions are given in Fig. 2.

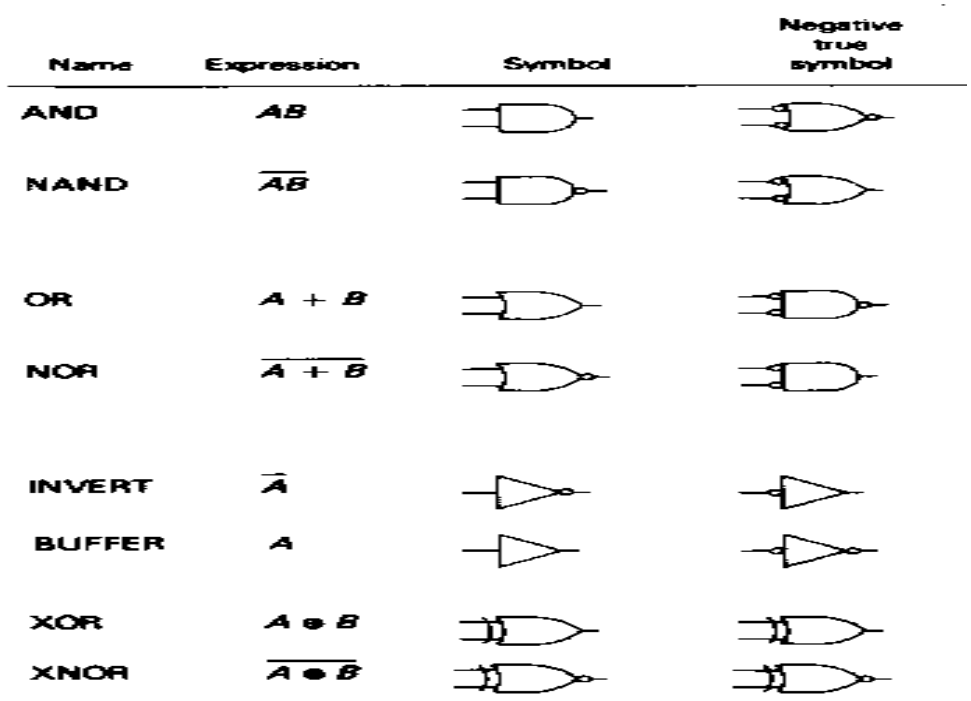


Figure 2:

All of the logical gate functions, as well as the Boolean relations discussed in the next section, follow from the truth tables for the AND and OR gates. We reproduce these below. We also show the XOR truth table, because it comes up quite often, although, as we shall see,it is not elemental.

MODULE-II

Gate level Minimization

The primary objective of all simplification procedures is to obtain an expression thathas the minimum number of terms. Obtaining an expression with the minimum number ofliterals is usually the secondary objective. If there is more than one possible solution with thesame number of terms, the one having the minimum number of literals is the choice.There are several methods for simplification of Boolean logic expressions. The process isusually called logic minimization‖ and the goal is to form a result which is efficient. Twomethods we will discuss are algebraic minimization and Karnaugh maps. For verycomplicated problems the former method can be done using special software analysisprograms. Karnaugh maps are also limited to problems with up to 4 binary inputs. TheQuine–McCluskey tabular method is used for more than 4 binary inputs.

The Map Method

Maurice Karnaugh, a telecommunications engineer, developed the Karnaugh map atBell Labs in 1953 while designing digital logic based telephone switching circuits.Karnaughmapsreduce logic functions more quickly and easily compared to Boolean algebra. Byreduce we mean simplify, reducing the number of gates and inputs. We like to simplify logicto a lowest cost form to save costs by elimination of components. We define lowest cost asbeing the lowest number of gates with the lowest number of inputs per gate.A Karnaugh map is a graphical representation of the logic system. It can be drawn directlyfrom either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions.Drawing a Karnaugh map from the truth table involves an additional step of writing theminterm or maxterm expression depending upon whether it is desired to have a minimizedsum-of-products or a minimized product of-sums expression

Construction of a Karnaugh Map

An n-variable Karnaugh map has 2n squares, and each possible input is allotted asquare. In the case of a mintermKarnaugh map, '1' is placed in all those squares for which the output is '1', and '0' is placed in all those squares for which the output is '0'. 0s are omitted for simplicity. An 'X' is placed in squares corresponding to 'don't care' conditions.In the case of a maxtermKarnaugh map, a '1' is placed in all those squares for which the output is '0', and a '0' is placed for input entries corresponding to a '1' output. Again, 0s are omitted for simplicity, and an 'X' is placed in squares corresponding to 'don't care'

conditions. The choice of terms identifying different rows and columns of a Karnaugh map isnot unique for a given number of variables. The only condition to be satisfied is that thedesignation of adjacent rows and adjacent columns should be the same except for one of theliterals beingcomplemented. Also, the extreme rows and extreme columns are consideredadjacent.Some of the possible designation styles for two-, three- and four-variable mintermKarnaughmaps are shown in the figure below.

The style of row identification need not be the same as that of column identification as longas it meets the basic requirement with respect to adjacent terms. It is, however, acceptedpractice to adopt a uniform style of row and column identification. Also, the style shown inthe figure below is more commonly used. A similar discussion applies for maxtermKarnaughmaps. Having drawn the Karnaugh map, the next step is to form groups of 1s as per thefollowing guidelines:

1. Each square containing a '1' must be considered at least once, although it can beconsidered as often as desired.

2. The objective should be to account for all the marked squares in the minimum numberof groups.

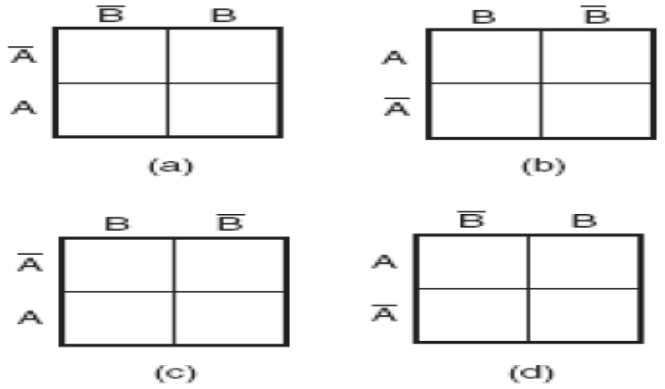
3. The number of squares in a group must always be a power of 2, i.e. groups can have 1,2, 4,8, 16, squares.

4. Each group should be as large as possible, which means that a square should not beaccounted for by itself if it can be accounted for by a group of two squares; a group oftwo squares should not be made if the involved squares can be included in a group offour squares and so on.

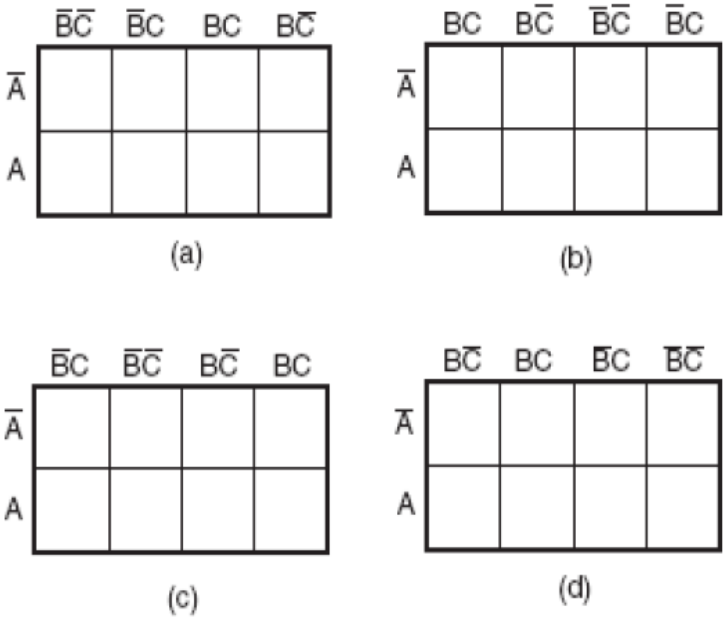
5. 'Don't care'entries can be used in accounting for all of 1-squares to make optimum groups. They are marked 'X' in the corresponding squares. It is, however, notnecessary to

account for all ‘don‘t care‘ entries. Only such entries that can be used toadvantage should be used.

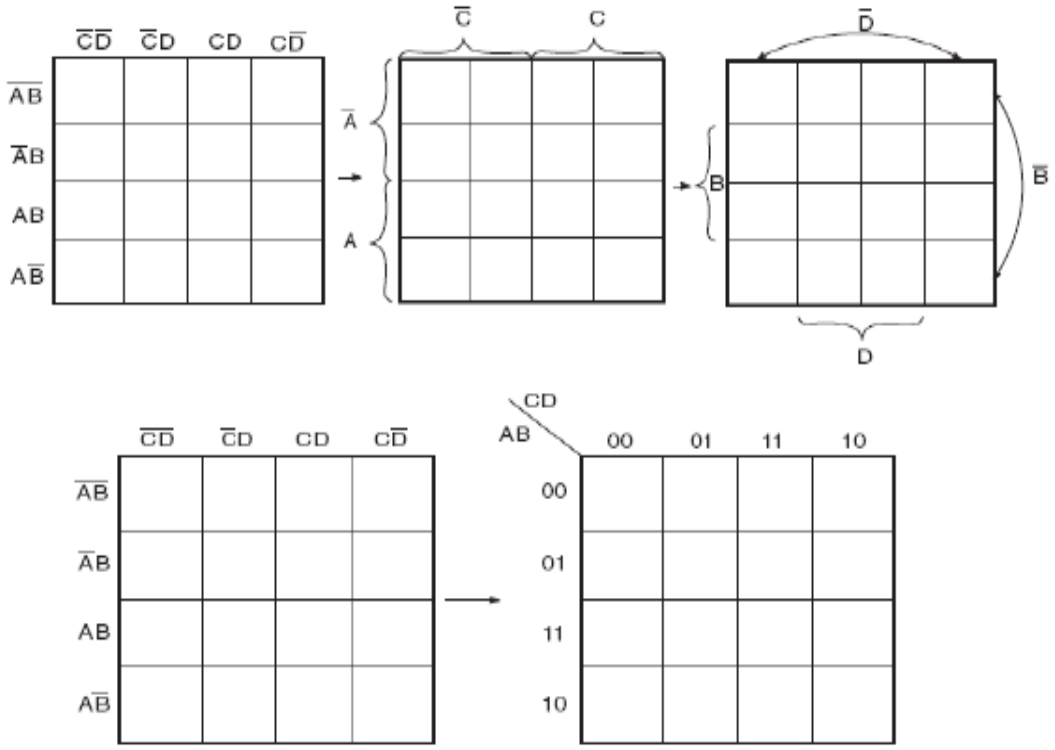
TWO VARIABLE K-MAP



THREE VARIABLE K-MAP



Four variable K Map



Different Styles of row and column identification

Having accounted for groups with all 1s, the minimum ‘sum-of-products‘ or ‘product-ofsums‘expressions can be written directly from the Karnaugh map. MintermKarnaugh mapand MaxtermKarnaugh map of the Boolean function of a two-input OR gate. The Mintermand Maxterm Boolean expressions for the two-input OR gate are as follows:

$$Y = A + B \text{ (maxterm or product-of-sums)}$$

$$Y = \overline{A}.B + A.\overline{B} + A.B \text{ (minterm or sum-of-products)}$$

Truth table

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

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

Sum-of-products K-map

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

Product-of-sums K-map

MintermKarnaugh map and MaxtermKarnaugh map of the three variable Boolean function

$$Y = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C}$$

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

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

	$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
$\overline{A}$	1			1
A	1			1

Sum-of-products K-map

	$\overline{B}+\overline{C}$	$\overline{B}+C$	B+C	$B+\overline{C}$
$\overline{A}$	1			1
A	1			1

Product-of-sums K-map

The truth table, MintermKarnaugh map and MaxtermKarnaugh map of the four variable

Boolean function

$$Y=\overline{A}.\overline{B}.\overline{C}.\overline{D}+\overline{A}.\overline{B}.\overline{C}.D+\overline{A}.B.\overline{C}.\overline{D}+\overline{A}.B.\overline{C}.D+A.\overline{B}.\overline{C}.\overline{D}+A.\overline{B}.\overline{C}.D+A.B.\overline{C}.\overline{D}+A.B.\overline{C}.D$$

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

To illustrate the process of forming groups and then writing the corresponding minimized .Boolean expression, The below figures respectively show minterm and maxtermKarnaugh maps for the Boolean functions expressed by the below equations. The minimizedexpressions as deduced from Karnaugh maps in the two cases are given by Equation in thecase of the mintermKarnaugh map and Equation in the case of the maxtermKarnaugh map:

$$Y=\overline{A}.\overline{B}.\overline{C}.\overline{D}+\overline{A}.\overline{B}.C.\overline{D}+\overline{A}.B.\overline{C}.D+\overline{A}.B.C.D+A.\overline{B}.\overline{C}.\overline{D}+A.\overline{B}.C.\overline{D}+A.B.\overline{C}.D+A.B.C.D$$

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

$$Y=\overline{B}.\overline{D}+B.D$$

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

Truth table

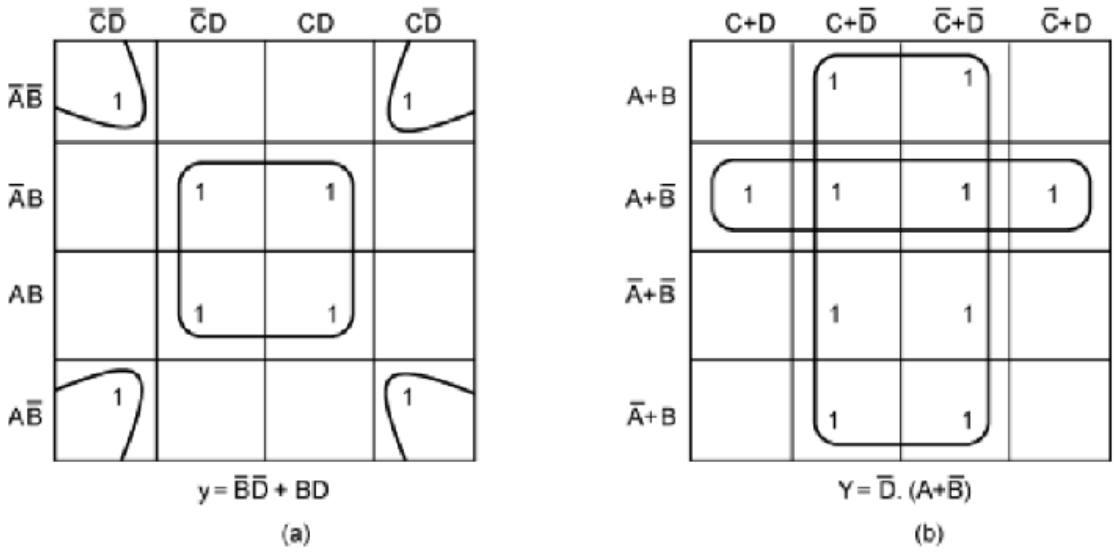
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	1		
$\overline{A}B$	1	1		
AB	1	1		
$A\overline{B}$	1	1		

Sum-of-products K-map

	$\overline{C}+\overline{D}$	$\overline{C}+D$	$C+D$	$C+\overline{D}$
$\overline{A}+\overline{B}$	1	1		
$\overline{A}+B$	1	1		
$A+B$	1	1		
$A+\overline{B}$	1	1		

Product-of-sums K-map



Group formation in minterm and maxterm Karnaugh maps.

Combinational Logic

Combinational Circuits

The term ‖combinational‖ comes to us from mathematics. In mathematics acombination is an unordered set, which is a formal way to say that nobody cares which orderthe items came in. Most games work this way, if you rolled dice one at a time and get a 2followed by a 3 it is the same as if you had rolled a 3 followed by a 2. With combinationallogic, the circuit producesthe same output regardless of the order the inputs are changed.There are circuits which depend on the when the inputs change, these circuits are calledsequential logic. Even though you will not find the term ‖sequential logic‖ in the chaptertitles, the next several chapters will discuss sequential logic. Practical circuits will have a mixof combinational and sequential logic, with sequential logic making sure everything happensin order and combinational logic performing functions like arithmetic, logic, or conversion.

Design Using Gates

A combinational circuit is one where the output at any time depends only on thepresent combination of inputs at that point of time with total disregard to the past state of theinputs. The logic gate is the most basic building block of combinational logic. The

logicalfunction performed by a combinational circuit is fully defined by a set of Boolean expressions. The other category of logic circuits, called sequential logic circuits, comprisesboth logic gates and memory elements such as flip-flops. Owing to the presence of memoryelements, the output in a sequential circuit depends upon not only the present but also the paststate of inputs.

The Fig shows the block schematic representation of a generalized combinational circuithaving n input variables and m output variables or simply outputs. Since the number of input
variables is

Generalized Combinational Circuit

n, there are 2n possible combinations of bits at the input. Each output can be expressed interms of input variables by a Boolean expression, with the result that the generalized systemof above fig can be expressed by m Boolean expressions. As an illustration, Booleanexpressions describing the function of a four-input OR/NOR gate are given as

$$Y_1 \text{ (OR output)} = A + B + C + D \quad \text{and} \quad Y_2 \text{ (NOR output)} = \overline{A + B + C + D} \quad \ldots\ldots \text{Eq} - 1$$

Binary Adder

Half-Adder

A half-adder is an arithmetic circuit block that can be used to add two bits. Such a circuit thushas two inputs that represent the two bits to be added and two outputs, with one producing theSUM output and the other producing the CARRY. Figure shows the truth table of a half-adder, showing all possible input combinations and the corresponding outputs.

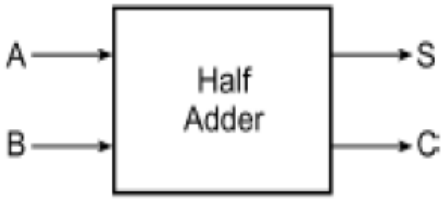
The Boolean expressions for the SUM and CARRY outputs are given by the equations below

SUM $S = A.\overline{B} + \overline{A}.B$

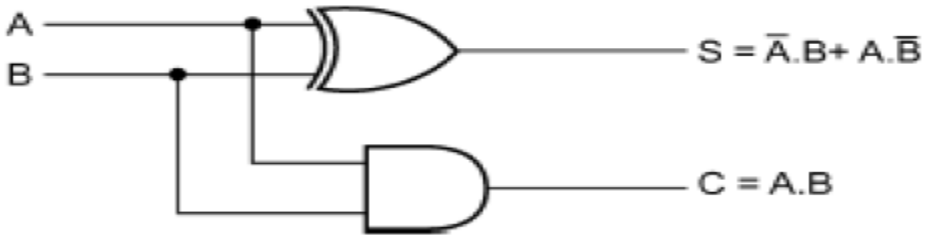
CARRY $C = A.B$

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

Truth Table of Half Adder



An examination of the two expressions tells that there is no scope for further simplification.While the first one representing the SUM output is that of an EX-OR gate, the second onerepresenting the CARRY output is that of an AND gate. However, these two expressions cancertainly be represented in different forms using various laws and theorems of Booleanalgebra to illustrate the flexibility that the designer has in hardware-implementing as simple acombinational function as that of a half-adder.



Logic Implementation of Half Adder

Although the simplest way to hardware-implement a half-adder would be to use a two-inputEX-OR gate for the SUM output and a two-input AND gate for the CARRY output, as shown in Fig. it could also be implemented by using an appropriate arrangement of eitherNAND or NOR gates.

Full Adder

A full adder circuit is an arithmetic circuit block that can be used to add three bits to producea SUM and a CARRY output. Such a building block becomes a necessity when it comes toadding binary numbers with a large number of bits. The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. Let us recall theprocedure for adding larger binary numbers. We begin with the addition of LSBs of the twonumbers. We record the sum under the LSB column and take the carry, if any, forward to the next higher column bits. As a result, when we add the next adjacent higher column bits, wewould be required to add three bits if there were a carry from the previous addition. We havea similar situation for the other higher column bits. Also until we reach the MSB. A full adderis therefore essential for the hardware implementation of an adder circuit capable of addinglarger binary numbers. A half-adder can be used for addition of LSBs only.

A	B	C_{in}	SUM (S)	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

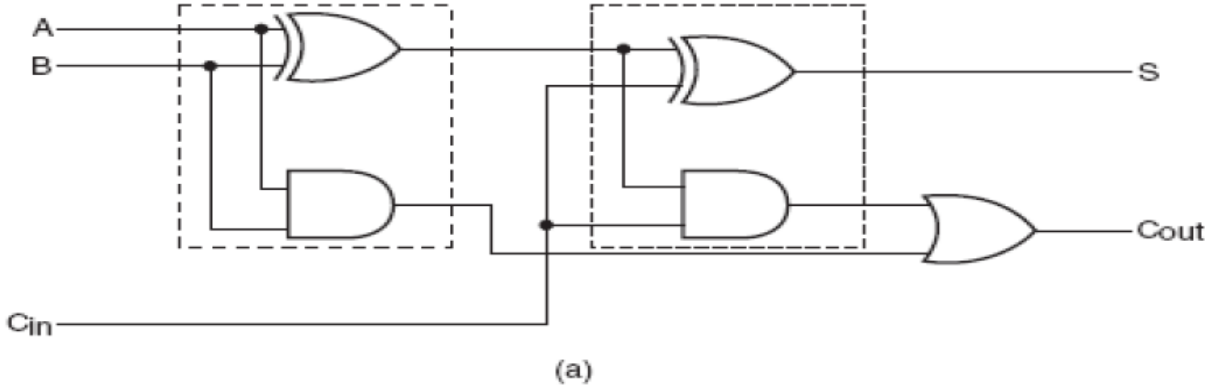
Truth Table of Full Adder

Figure shows the truth table of a full adder circuit showing all possible input combinationsand corresponding outputs. In order to arrive at the logic circuit for hardware implementationof a full adder, we will firstly write the Boolean expressions for the two output variables, thatis, the SUM and CARRY outputs, in terms of input variables. These expressions are thensimplified by using any of the simplification techniques described in the previous chapter.The Boolean expressions for the two output variables are given in Equation below for theSUM output (S) and in above Equation for the CARRY output (Cout):

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$

$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C}_{in} + A.B.C_{in}$$

Boolean expression above can be implemented with a two-input EX-OR gate provided that one of the inputs is Cin and the other input is the output of another two-input EX-OR gate with A and B as its inputs. Similarly, Boolean expression above can be implemented byORing two minterms. One of them is the AND output of A and B. The other is also theoutput of an AND gate whose inputs are Cin and the output of an EX-OR operation on A andB. The whole idea of writing the Boolean expressions in this modified form was todemonstrate the use of a half-adder circuit in building a full adder. Figure shows logicimplementation of Equations above.



(a)

Half-Subtractor

We will study the use of adder circuits for subtraction operations in the following pages.Before we do that, we will briefly look at the counterparts of half-adder and full addercircuits in the half-subtractor and full subtractor for direct implementation of subtractionoperations using logic gates.

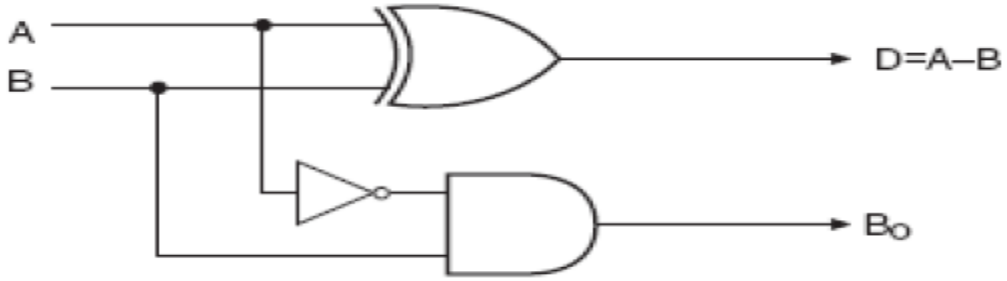
A half-subtractor is a combinational circuit that can be used to subtract one binary digit fromanother to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The truth table ofa half-subtractor, as shown in Fig. explains this further. The Boolean expressions for thetwo outputs are given by the equations

$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Subtractor



Logic Diagram of a Half Subtractor

Full Subtractor

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and
also takes into consideration whether a '1' has already been borrowed by the previousadjacent lower minuend bit or not. As a result, there are three bits to be handled at the inputof a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin .There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo.

The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next
possible higher minuend bit. Figure shows the truth table of a full subtractor.

The Boolean expressions for the two output variables are given by the equations

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in}$$

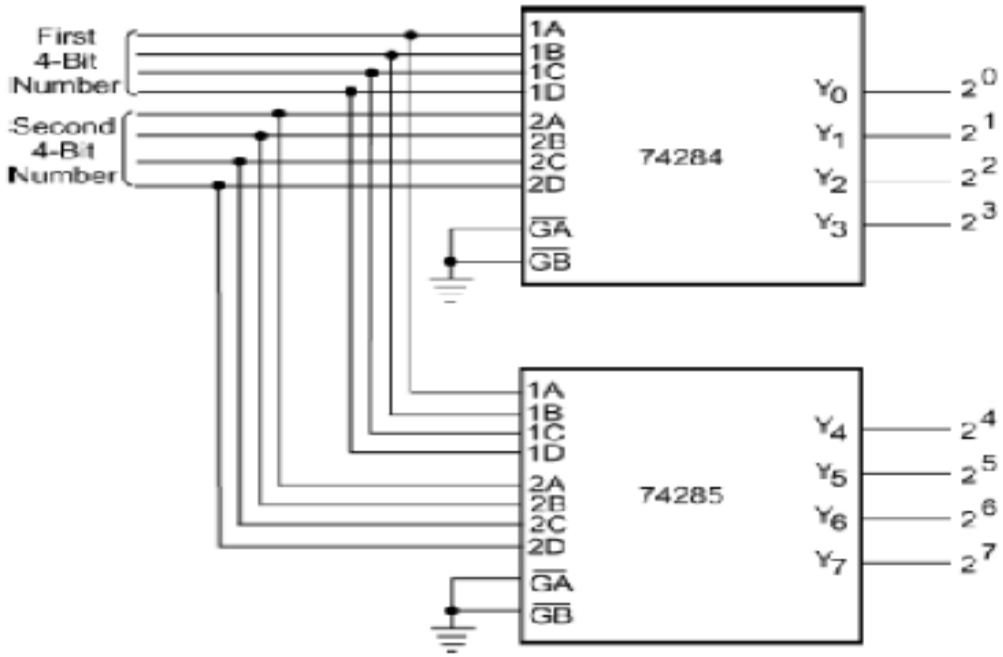
Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_O)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth Table of Full Subtractor

Binary Multiplier

Multiplication of binary numbers is usually implemented in microprocessors andmicrocomputers by using repeated addition and shift operations. Since the binary adders aredesigned to add only two binary numbers at a time, instead of adding all the partial productsat the end, they are added two at a time and their sum is accumulated in a register called theaccumulator register. Also, when the multiplier bit is '0', that very partial product is ignored, as an all '0' line does not affect the final result. The basic hardware arrangement of such abinary multiplier would comprise shift registers for the multiplicand and multiplier bits, anaccumulator register for storing partial products, a binary parallel adder and a clock pulsegenerator to time various operations.

Binary multipliers are also available in IC form. Some of the popular type numbers inthe TTL family include 74261 which is a 2 × 4 bit multiplier (a four-bit multiplicand designated asB0,B1,B2,B3 and B4, and a two-bit multiplier designated as M0, M1 and M2.The MSBs B4andM2 are used to represent signs. 74284 and 74285 are 4 × 4 bit multipliers.They can beused together to perform high-speed multiplication of two four-bit numbers. Figure shows the arrangement. The result of multiplication is often required to be stored in a register. The size ofthis register (accumulator) depends upon the number of bits in the result, which at the mostcan be equal to the sum of the number of bits in the multiplier and multiplicand. Somemultipliers ICs have an in-built register.



4 x 4 Multiplier

Magnitude comparator

A magnitude comparator is a combinational circuit that compares two given numbersand determines whether one is equal to, less than or greater than the other. The output is inthe form of three binary variables representing the conditions A = B,A>B and A<B, if A andB are the two numbers being compared. Depending upon the relative magnitude of the twonumbers, the relevant output changes state. If the two numbers, let us say, are four-bit binarynumbers and are designated as (A3 A2 A1 A0) and (B3 B2 B1 B0), the two numbers will beequal if all pairs of significant digits are equal, that is, A3= B3, A2 = B2, A1= B1 and A0 =B0. In order to determine whether A is greater than or less than B we inspect the relativemagnitude of pairs of significant digits, starting from the most significant position. Thecomparison is done by successively comparing the next adjacent lower pair of digits if thedigits of the pair under examination are equal. The comparison continues until a pair ofunequal digits is reached. In the pair of unequal digits, if Ai = 1 and Bi = 0, then A > B, and ifAi = 0, Bi= 1 then A < B. If X, Y and Z are three variables respectively representing the A =B, A > B and A < B conditions, then the Boolean expression representing these conditionsare given by the equations

$$X = x_3.x_2.x_1.x_0 \quad \text{where } x_i = A_i.B_i + \overline{A_i}.\overline{B_i}$$

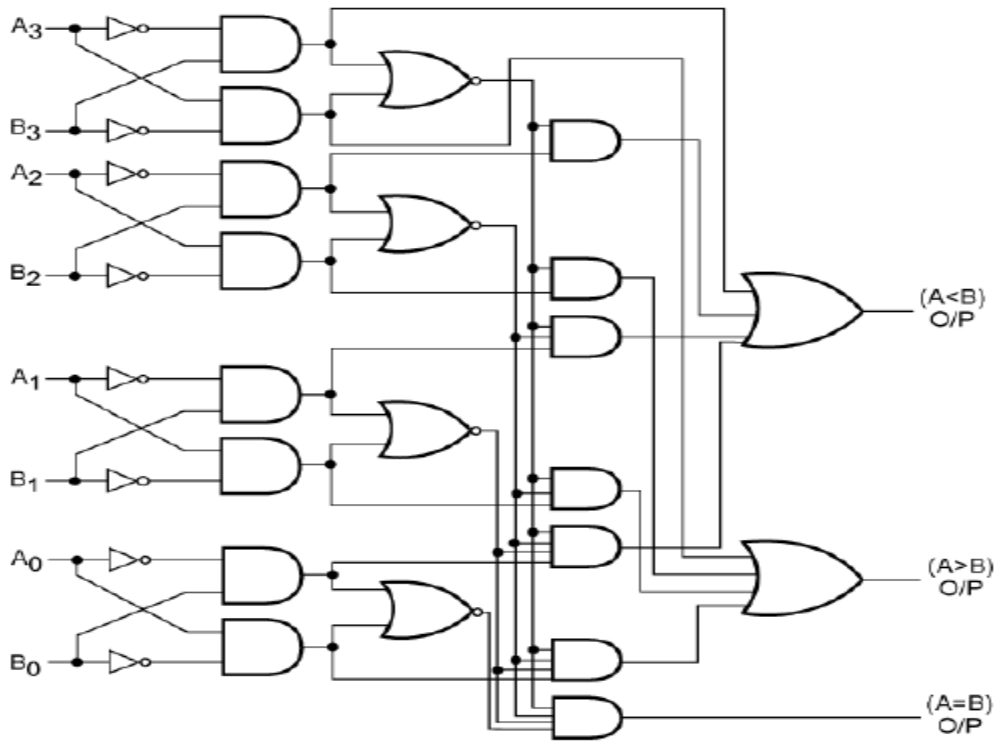
$$Y = A_3.\overline{B_3} + x_3.A_2.\overline{B_2} + x_3.x_2.A_1.\overline{B_1} + x_3.x_2.x_1.A_0.\overline{B_0}$$

$$Z = \overline{A_3}.B_3 + x_3.\overline{A_2}.B_2 + x_3.x_2.\overline{A_1}.B_1 + x_3.x_2.x_1.\overline{A_0}.B_0$$

Let us examine equations .x3 will be '1' only when both A3 and B3 are equal.Similarly, conditions for x2, x1 and x0 to be '1' respectively are equal A2 and B2, equalA1 and B1 and equal A0 and B0. ANDing of x3, x2, x1 and x0 ensures that X will be '1'when x3, x2, x1 and x0 are in the logic '1' state. Thus, X = 1 means that A = B. Onsimilar lines, it can be visualized that equations and respectively represent A> B and A < B conditions. Figure shows the logic diagram of a four-bit magnitudecomparator.

Magnitude comparators are available in IC form. For example, 7485 is a four-bitmagnitude comparator of the TTL logic family. IC 4585 is a similar device in the CMOSfamily. 7485 and 4585 have the same pin connection diagram and functional table. Thelogic circuit inside these devices determines whether one four-bit number, binary or BCD,is less than, equal to or greater than a second four-bit number. It can perform comparisonof straight binary and straight BCD (8-4-2-1) codes. These devices can be cascadedtogether to perform operations on larger bit numbers without the help of any externalgates. This is facilitated by three additional inputs called cascading or expansion inputsavailable on the IC. These cascading inputs are also designated as A = B, A > B and A <B inputs. Cascading of individual magnitude comparators of the type 7485 or 4585 isdiscussed in the following paragraphs. IC 74AS885 is another common magnitudecomparator. The device is an eight bit magnitude comparator belonging to the advancedSchottky TTL family. It can perform high-speed arithmetic or logic comparisons on twoeight-bit binary or 2's complement numbers and produces two fully decoded decisions at

the output about one number being either greater than or less than the other. More than one of these devices can also be connected in a cascade arrangement to performcomparison of numbers of longer lengths.



Four Bit Magnitude Comparator

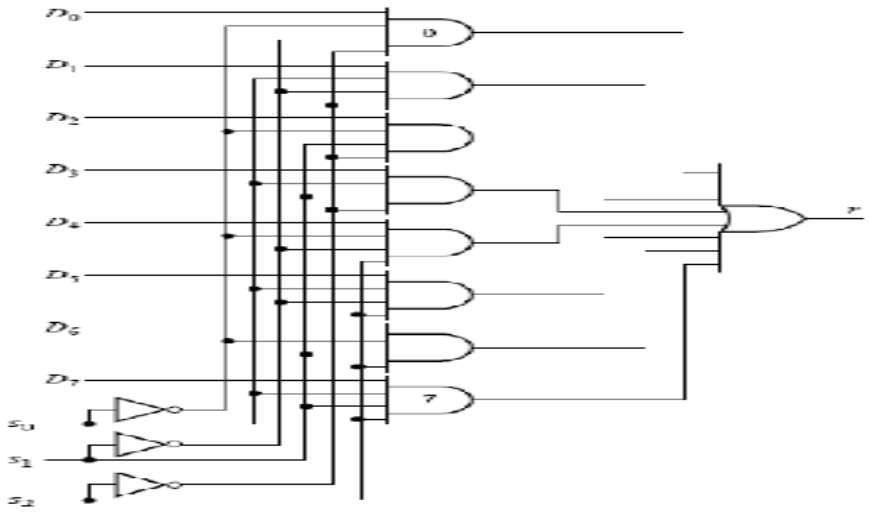
MULTIPLEXERS

Data generated in one location is to be used in another location; A method is needed totransmit it from one location to another through some communications channel. The data isavailable, in parallel, on many different lines but must be transmitted over a singlecommunications link.

A mechanism is needed to select which of the many data lines toactivate sequentially at any one time so that the data this line carries can be transmitted at thattime.This process is called multiplexing.Anexample is the multiplexing of conversations on the telephone system. A number of telephoneconversations are alternately switched onto the telephone line many times per second.Because of the nature of the human auditory system, listeners cannot detect that what they arehearing is chopped up and that other people‘s conversations are interspersed with their own inthe transmission process.

Needed at the other end of the communications link is a device that will undo themultiplexing: a demultiplexer. Such a device must accept the incoming serial data and directit in parallel to one of many output lines. The interspersed snatches of

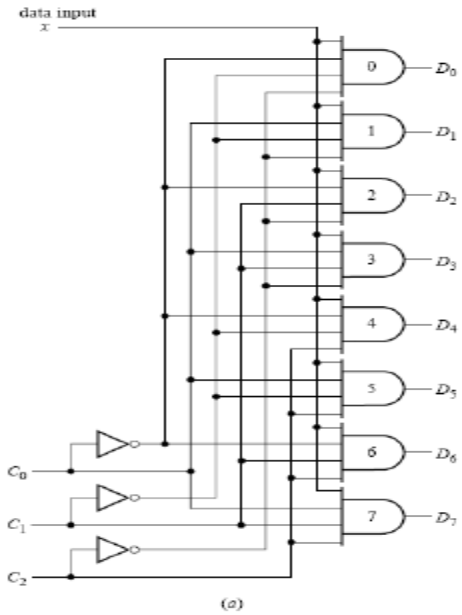
telephoneconversations, for example, must be sent to the correct listeners.
A digital multiplexer is a circuit with 2n data input lines and one output line. It must alsohave a way of determining the specific data input line to be selected at any one time. This isdone with n other input lines, called the select or selector inputs, whose function is to selectone of the 2n data inputs for connection to the output. A circuit for n = 3 is shown in Figure13. The n selector lines have 2n = 8 combinations of values that constitute binary selectnumbers



Multiplexer with eight data inputs

Demultiplexers

The demultiplexer shown there is a single-input, multiple-output circuit. However, inaddition to the data input, there must be other inputs to control the transmission of the data tothe appropriate data output line at any given time. Such a demultiplexer circuit having eightoutput lines is shown in Figure 16a. It is instructive to compare this demultiplexer circuit withthe multiplexer circuit in Figure 13. For the same number of control (select) inputs, there arethe same number of AND gates. But now each AND gate output is a circuit output. Ratherthan each gate having its own separate data input, the single data line now forms one of theinputs to each AND gate, the other AND inputs being control inputs.



Control Inputs			Data Outputs							
C_2	C_1	C_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	*x*	0	0	0	0	0	0	0
0	0	1	0	*x*	0	0	0	0	0	0
0	1	0	0	0	*x*	0	0	0	0	0
0	1	1	0	0	0	*x*	0	0	0	0
1	0	0	0	0	0	0	*x*	0	0	0
1	0	1	0	0	0	0	0	*x*	0	0
1	1	0	0	0	0	0	0	0	*x*	0
1	1	1	0	0	0	0	0	0	0	*x*

(b)

A demultiplexer circuit (a) and its truth table (b).

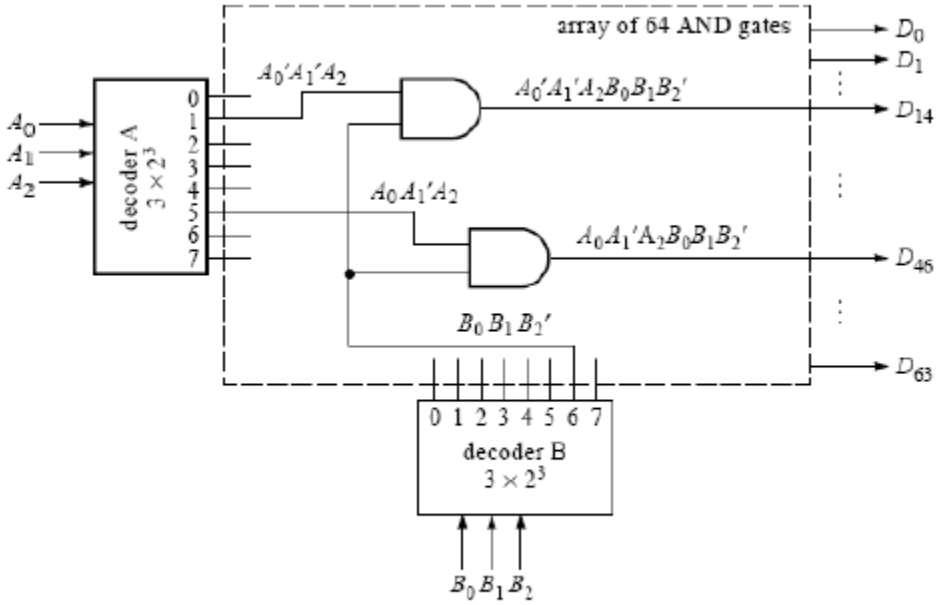
Decoders and Encoders

The previous section began by discussing an application: Given 2n data signals, theproblem is to select, under the control of n select inputs, sequences of these 2n data signals tosend out serially on a communications link. The reverse operation on the receiving end of thecommunications link is to receive data serially on a single line and to convey it to one of 2noutput lines. This again is controlled by a set of control inputs. It is this application that needsonly one input line; other applications may require more than one.We will now investigatesuch a generalized circuit.

Conceivably, there might be a combinational circuit that accepts n inputs (not necessarily 1,but a small number) and causes data to be routed to one of many, say up to 2n, outputs. Suchcircuits have the generic name decoder.Semantically, at least, if something is to be decoded, it must have previously been encoded,the reverse operation from decoding. Like a multiplexer, an encoding circuit must accept datafrom a large number of input lines and convert it to data on a smaller number of output lines(notnecessarily just one). This section will discuss a number of implementations of decoders andencoders.

n-to-2n-Line Decoder

In the demultiplexer circuit in Figure, suppose the data input line is removed.(Draw the circuit for yourself.) Each AND gate now has only n (in this case three) inputs, andthere are 2n (in this case eight) outputs. Since there isn‘t a data input line to control, whatused to be control inputs no longer serve that function. Instead, they are the data inputs to bedecoded. This circuit is an example of what is called an n-to-2n-line decoder. Each outputrepresents a minterm. Output k is 1 whenever the combination of the input variable values isthe binary equivalent of decimal k. Now suppose that the data input line from thedemultiplexer in Figure 16 is not removed but retained and viewed as an enable input. Thedecoder now operates only when the enable x is 1. Viewed conversely, an n-to-2n-linedecoder with an enable input can also be used as a demultiplexer, where the enable becomesthe serial data input and the data inputs of the decoder become the control inputs of thedemultiplexer.7 Decoders of the type just described are available as integrated circuits (MSI);n = 3 and n = 4 are quite common. There is no theoretical reason why n can‘t be increased tohigher values. Since, however, there will always be practical limitations on the fan-in (thenumber of inputs that a physical gate can support), decoders of higher order are oftendesigned using lower-order decoders interconnected with a network of other gates.

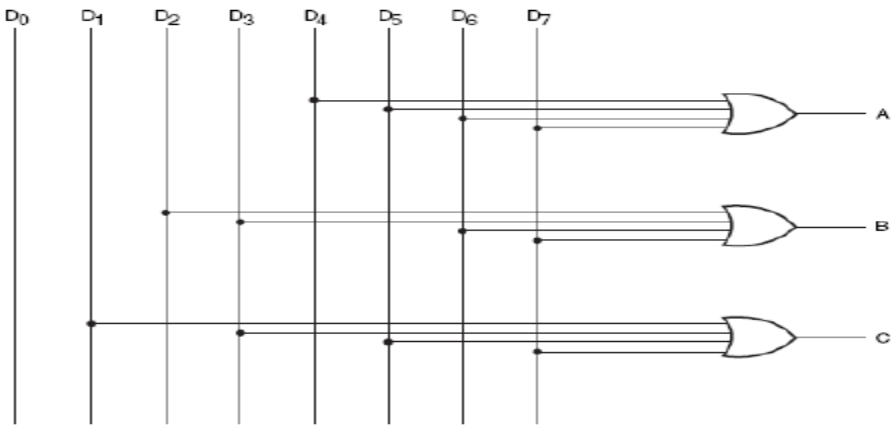


Encoder

An encoder is a combinational circuit that performs the inverse operation of a decoder. If a device output code has fewer bits than the input code has, the device is usually called an encoder. e.g. 2n-to-n, priority encoders.The simplest encoder is a 2n-to-n binary encoder, where it has only one of 2n inputs =1 and the output is the n-bit binary number corresponding to the active input.

Priority Encoder

A priority encoder is a practical form of an encoder. The encoders available in ICform are all priority encoders. In this type of encoder, a priority is assigned to each input sothat, when more than one input is simultaneously active, the input with the highest priority isencoded. We will illustrate the concept of priority encoding with the help of an example. Letus assume that the octal to-binary encoder described in the previous paragraph has an inputpriority for higher-order digits. Let us also assume that input lines D2, D4 and D7 are all simultaneously in logic '1' state. In that case, only D7 will be encoded and the output will be111. The truth table of such a priority



Octal to binary encoder

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Truth table of encoder

encoder will then be modified to what is shown above in truth table. Looking at the last rowof the table, it implies that, if D7 = 1, then, irrespective of the logic status of other inputs, theoutput is 111 as D7 will only be encoded. As another example, Fig.shows the logicsymbol and truth table of a 10-line decimal to four-line BCD encoder providing priorityencoding for higher-order digits, with digit 9 having the highest priority. In the functionaltable shown, the input line with highest priority having a LOW on it is encoded irrespectiveof the logic status of the other input lines.

MODULE-III

Synchronous Sequential logic

Latches

The following 3 figures are equivalent representations of a simple circuit. In general these are called flip-flops. Specially, these examples are called SR ("set-reset") flip-flops, or SR latches.

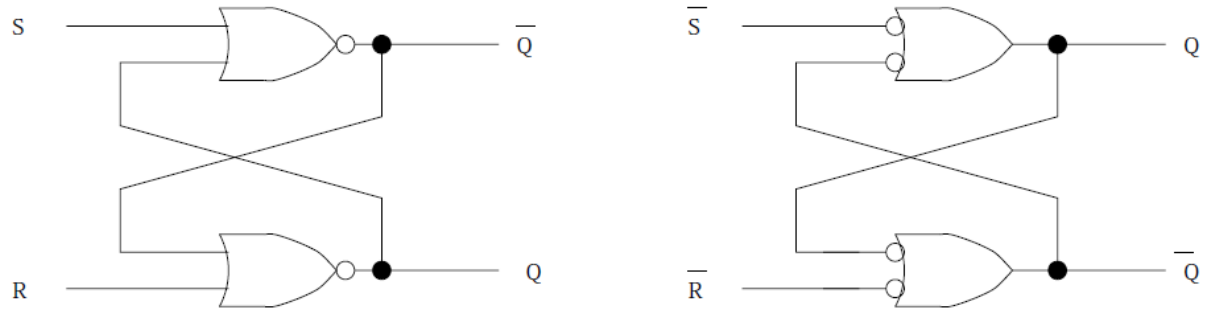


Figure : Two equivalent versions of an SR flip-flop (or "SR latch").

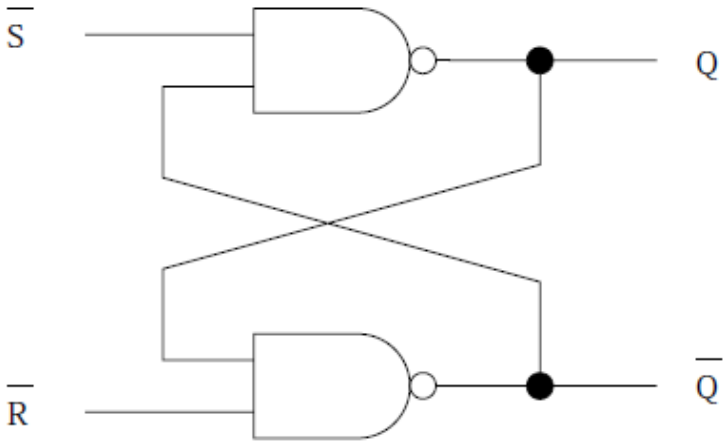


Figure : Yet another equivalent SR flip-flop

The truth table for the SR latch is given below.

S	$\overline{S}$	R	$\overline{R}$	Q	$\overline{Q}$
1	0	0	1	1	0
0	1	1	0	0	1
0	1	0	1	`retains previous`	
1	0	1	0	0	0

The state described by the last row is clearly problematic, since Q and $\overline{Q}$ should not be the same value. Thus, the $S = R = 1$ inputs should be avoided.

From the truth table, we can develop a sequence such as the following:

1. $R = 0$, $S = 1 \Rightarrow Q = 1$ (`set`)
2. $R = 0$, $S = 0 \Rightarrow Q = 1$ ($Q = 1$ state retained: "memory")
3. $R = 1$, $S = 0 \Rightarrow Q = 0$ (`reset`)
4. $R = 0$, $S = 0 \Rightarrow Q = 0$ ($Q = 0$ state retained)

In alternative language, the first operation "writes" a `true` state into one bit of memory. It can subsequently be "read" until it is erased by the reset operation of the third line.

Flip Flops

The flip-flop is an important element of such circuits. It has the interesting property of AnSRFlip-flop has two inputs: S for setting and R for Resetting the flip- flop : It can be set toa state which is retained until explicitly reset.

R-S Flip-Flop

A flip-flop, as stated earlier, is a bistable circuit. Both of its output states arestable. The circuit remains in a particular output state indefinitely until something is doneto change that output status. Referring to the bistablemultivibrator circuit discussedearlier, these two states were those of the output transistor in saturation (representing aLOW output) and in cut-off (representing a HIGH output). If the LOW and HIGH outputs are respectively regarded as '0' and '1', then the output can either be a '0' or a '1'. Since either a '0' or a '1' can be held indefinitely until the circuit is appropriately triggered togo to the other state, the circuit is said to have memory. It is capable of storing one binarydigit or one bit of digital information. Also, if we recall the functioning of the bistablemultivibrator circuit, we find that, when one of the transistors was in saturation, the otherwas in cut-off. This implies that, if we had taken outputs from the collectors of bothtransistors, then the two outputs would be complementary.

J-K Flip-Flop

A J-K flip-flop behaves in the same fashion as an R-S flip-flop except for one ofthe entries in the function table. In the case of an R-S flip-flop, the input combination S =R = 1 (in the case of a flip-flop with active HIGH inputs) and the input combination S = R= 0 (in the case of a flip-flop with active LOW inputs) are prohibited. In the case of a J-Kflip-flop with active HIGH inputs, the output of the flip-flop toggles, that is, it goes to theother state, for J = K = 1 . The output toggles for J = K = 0 in the case of the flip-flophaving active LOW inputs. Thus, a J-K flip-flop overcomes the problem of a forbiddeninput combination of the R-S flip-flop. Figures below respectively show the circuitsymbol of level-triggered J-K flip-flops with active HIGH and active LOW inputs, alongwith their function tables.

The characteristic tables for a J-K flip-flop with active HIGH J and K inputs and a J-Kflip-flop with active LOW J and K inputs are respectively shown in Figs (a) and (b). The corresponding Karnaugh maps are shown in Fig below for the characteristics table ofFig and in below for the characteristic table below. The characteristic equations for theKarnaugh maps of below figure is shown next

$$Q_{n+1} = J.\overline{Q_n} + \overline{K}.Q_n$$
$$Q_{n+1} = \overline{J}.\overline{Q_n} + K.Q_n$$

Operation Mode	J	K	Clk	Q_{n+1}
SET	1	0	1	1
RESET	0	1	1	0
NO CHANGE	0	0	1	Q_n
TOGGLE	1	1	1	$\overline{Q_n}$

(a)

Operation Mode	J	K	Clk	Q_{n+1}
SET	0	1	1	1
RESET	1	0	1	0
NO CHANGE	1	1	1	Q_n
TOGGLE	0	0	1	$\overline{Q_n}$

(b)

FIG a. JK flip flop with active high inputs, b. JK flip flop with active low inputs

Toggle Flip-Flop (T Flip-Flop)

The output of a toggle flip-flop, also called a T flip-flop, changes state every time it istriggered at its T input, called the toggle input. That is, the output becomes '1' if it was '0'and '0' if it was '1'.

Positive edge-triggered and negative edge-triggered T flip-flops, along with their functiontables.If we consider the T input as active when HIGH, the characteristic table of such a flip-flopis shown in Fig. If the T input were active when LOW, then the characteristic table wouldbe as shown in Fig. The Karnaugh maps for the characteristic tables of Figs shownrespectively. The characteristic equations as written from the Karnaugh maps are asfollows:

$$Q_{n+1} = T.Q_n + T.Q_n$$

$$Q_{n+1} = \overline{T}.\overline{Q_n} + T.Q_n$$

—	Q_n	Q_{n+1}
↓	0	1
↓	1	0

(a)

—	Q_n	Q_{n+1}
↓	0	1
↓	1	0

(b)

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

(c)

Q_n	T	Q_{n+1}
0	0	1
0	1	0
1	0	0
1	1	1

(d)

D Flip-Flop

A D flip-flop, also called a delay flip-flop, can be used to provide temporary storage of one bit of information. Figure shows the circuit symbol and function table of a negative edge-triggered D flip-flop. When the clock is active, the data bit (0 or 1) present at the D input is transferred to the output. In the D flip-flop of Fig the data transfer from D input to Q output occurs on the negative-going (HIGH-to-LOW) transition of the clock input. The D input can acquire new status

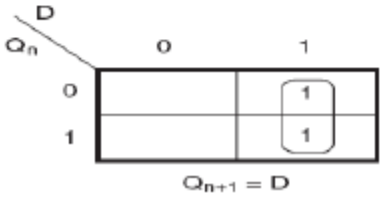
(a)

D	Clk	Q
0	↓	0
1	↓	1

(b)

Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

(c)



$Q_{n+1} = D$

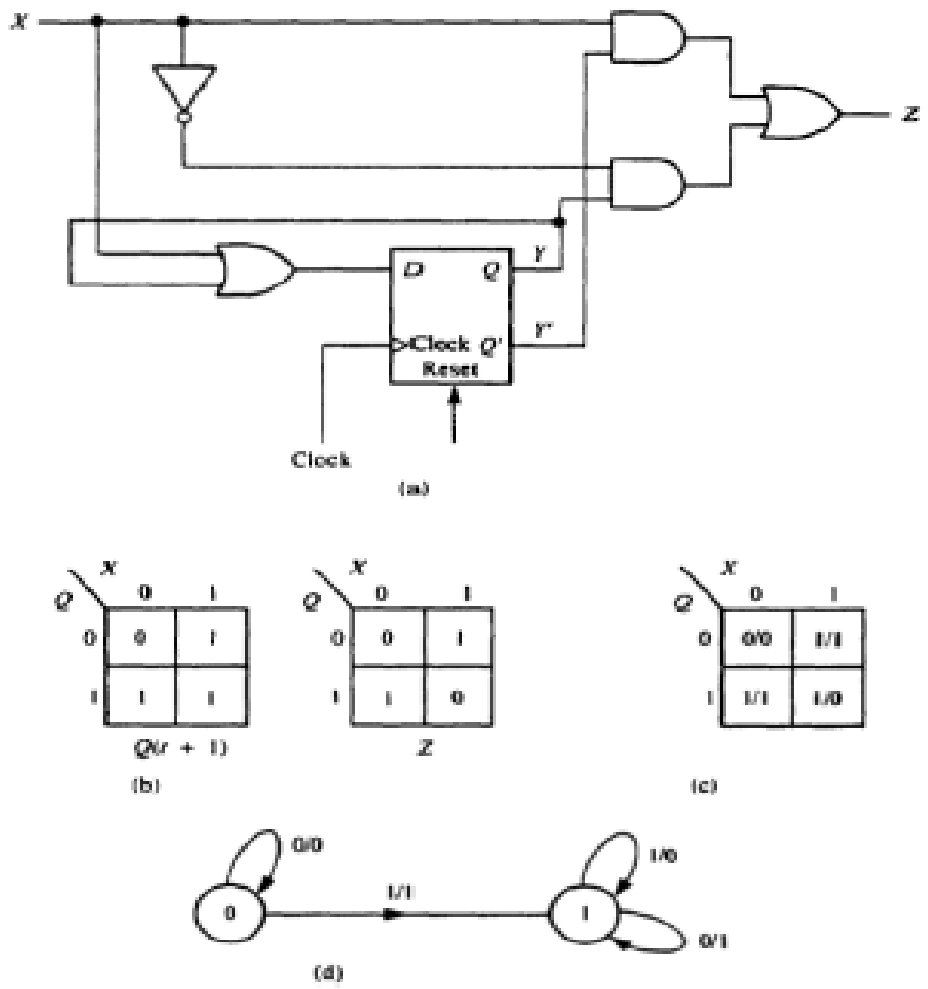
(d)

Analysis of Clocked Sequential circuits

The analysis of a synchronous sequential circuit is the process of determining the functional relation that exists between its outputs, its inputs, and its internal states. The contents of all the flip-flops in the circuit combined determine the internal state of the circuit. Thus, if the circuit contains n flip-flops, it can be in one of the 2^n states. Knowing the present state of the circuit and the input values at any time t, we should be able to derive its next state (i.e., the state at time $t + 1$) and the output produced by the circuit at t.

A sequential circuit can be described completely by a state table that is very similar to the ones shown for flip-flops

For a circuit with n flip-flops, there will be 2^n rows in the state table. If there are m inputs to the circuit, there will be 2^m columns in the state table. At the intersection of each row and column, the next-state and the output information are recorded. A *state diagram* is a graphical representation of the state table, in which each state is represented by a circle and the state transitions are represented by arrows between the circles. The input combination that brings about the transition and the corresponding output information are shown on the arrow. Analyzing a sequential circuit thus corresponds to generating the state table and the state diagram for the circuit. The state table or state diagram can be used to determine the output sequence generated by the circuit for a given input sequence if the *initial state* is known. It is important to note that for proper operation, a sequential circuit must be in its initial state before the inputs to it can be applied. Usually the power-up circuits are used to initialize the circuit to the appropriate state when the power is turned on.



Sequential circuit analysis. (a) Circuit; (b) next-state and output tables; (c) transition table; (d) state diagram; (e) timing diagram for level input; (f) timing diagram for synchronous pulse input.

Design of synchronous sequential circuit

The design of a sequential circuit is the process of deriving a logic diagram from the specification of the circuit's required behavior. The circuit's behavior is often expressed in words. The first step in the design is then to derive an exact specification of the required behavior in terms of either a state diagram or a state table. This is probably the most difficult step in the design, since no definite rules can be established to derive the state diagram or a state table. The designer's intuition and experience are the only guides. Once the description is converted into the state diagram or a state table, the remaining steps become mechanical. We will examine the classical design procedure through the examples in this section. It is not always necessary to follow this classical procedure, as some designs lend themselves to more direct and intuitive design methods. (The design of shift registers, described in Chapter 7, is one such example.) The classical design procedure consists of the following steps:

1. Deriving the state diagram (and state table) for the circuit from the problem statement.
2. Deriving the number of flip-flops (p) needed for the design from the number of states in the state diagram, by the formula

$$2^{p-1} < n \leq 2^p$$

   where n = number of states.
3. Deciding on the types of flip-flops to be used. (This often simply depends on the type of flip-flops available for the particular design.)
4. Assigning a unique p-bit pattern (state vector) to each state.
5. Deriving the state transition table and the output table.
6. Separating the state transition table into p tables, one for each flip-flop.
7. Deriving an input table for each flip-flop input using the excitation tables
8. Deriving input equations for each flip-flop input and the circuit output equations.
9. Drawing the circuit diagram.

Counters

In digital logic and computing, a counter is a device which stores (and sometimesdisplays) the number of times a particular event or process has occurred, often inrelationship to a clock signal.

In practice, there are two types of counters:
□ up counters which increase (increment) in value
□ down counters which decrease (decrement) in value

Counters Types

In electronics, counters can be implemented quite easily using register-type circuits such as the flip-flop, and a wide variety of designs exist,

e.g:
□ Asynchronous (ripple) counters
□ Synchronous counters
□ Johnson counters
□ Decade counters
□ Up-Down counters
□ Ring counters

Each is useful for different applications. Usually, counter circuits are digital in nature, andcount in binary, or sometimes binary coded decimal. Many types of counter circuit areavailable as digital building blocks, for example a number of chips in the 4000 seriesimplement different counters.

Asynchronous (ripple) counters

The simplest counter circuit is a single D-type flip flop, with its D (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero toone before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternatebetween a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a newclock with a 50% duty cycle at exactly half the frequency of the input clock. If this outputis then used as the clock signal for a similarly arranged D flip flop (remembering to invertthe output to the input), you will get another 1 bit counter that counts half as fast. Puttingthem together yields a two bit counter:

Decade counters

Decade counters are a kind of counter that counts in tens rather than having a binaryrepresentation. Each output will go high in turn, starting over after ten outputs haveoccurred. This type of circuit finds applications in multiplexers and demultiplexers, orwherever a scanning type of behaviour is useful. Similar counters with different numbers of outputs are also common.

Up-Down Counters

It is a combination of up counter and down counter, counting in straight binary sequence.There is an up-down selector. If this value is kept high, counter increments binary valueand if the value is low, then counter starts decrementing the count. The Down counters aremade by using the complemented output to act as the clock for the next flip-flop in the case of Asynchronous counters. An Up counter is constructed by linking the Q out of the

J-K Flip flop and putting it into a Negative Edge Triggered Clock input. A Down Counter is constructed by taking the Q output and putting it into a Positive Edge Triggered input Ring Counters A ring counter is a counter that counts up and when it reaches the last number that isdesigned to count up to, it will reset itself back to the first number. For example, a ringcounter that is designed using 3 JK Flip Flops will count starting from 001 to 010 to 100 and back to 001. It will repeat itself in a 'Ring' shape and thus the name Ring Counter isgiven.

Asynchronous Counter

Flip-flops can be connected in series, as shown in Fig. 21. The resulting outputs are given in Fig. 22. (Note that labels in these two figures correspond when $A \equiv 2^0$, $B \equiv 2^1$, $C \equiv 2^2$, and $D \equiv 2^3$. Hence, this is a 4-bit counter, with maximum count $2^4 - 1 = 15$. It is clearly possible to expand such a counter to an indefinite number of bits.

While asynchronous counters are easy to assemble, they have serious drawbacks for some applications. In particular, the input must propogate through the entire chain of flip-flops before the correct result is achieved. Eventually, at high input rate, the two ends of the chain, representing the LSB and MSB, can be processing different input pulses entirely. (Perhaps in lab you can see this effect on the oscilloscope with a very high input frequency.) The solution to this is the *synchronous counter*, which we will discuss below as an example of a state machine.

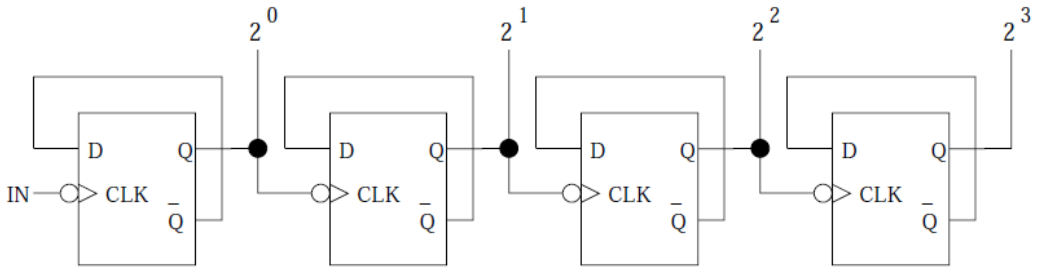


Figure : Asynchronous ("ripple") counter made from cascaded D-type flip-flops.

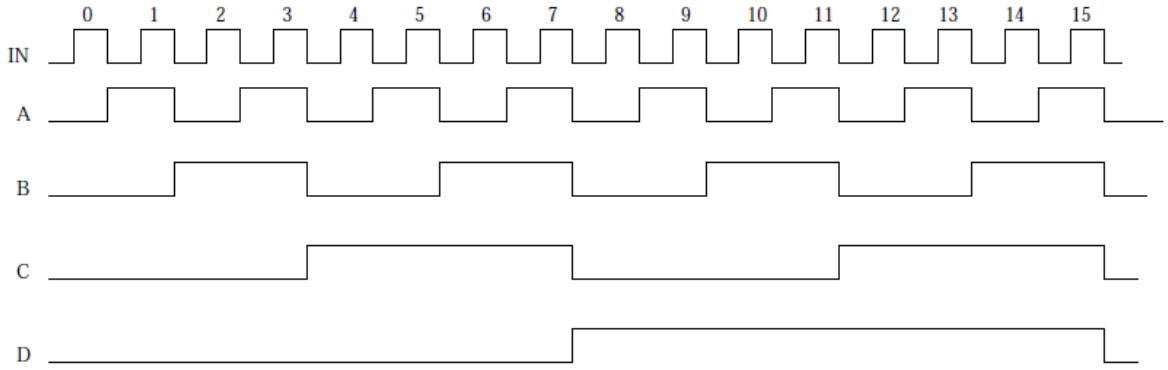


Figure : Waveforms generated by the ripple counter.

Shift register

In digital circuits a shift register is a group of flip flops set up in a linear fashion which havetheir inputs and outputs connected together in such a way that the data is shifted down theline when the circuit is activatedShift register
In digital circuits a shift register is a group of flip flops set up in a linear fashion which havetheir inputs and outputs connected together in such a way that the data is shifted down theline when the circuit is activated

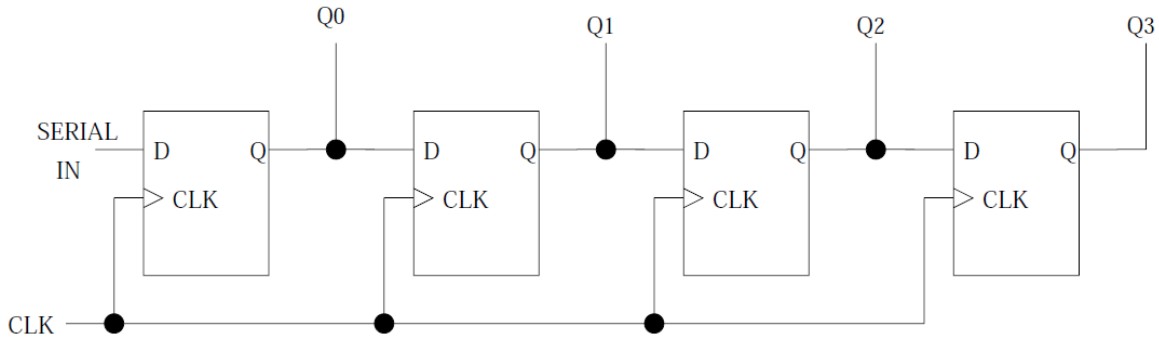


Figure : 4-bit shift register.

State Table

The state table representation of a sequential circuit consists of three sections labeled presentstate, next state and output. The present state designates the state of flip-flops before theoccurrence of a clock pulse. The next state shows the states of flip-flops after the clock pulse,and the output section lists the value of the output variables during the present state.

State Diagram

In addition to graphical symbols, tables or equations, flip-flops can also be representedgraphically by a state diagram. In this diagram, a state is represented by a circle, and thetransition between states is indicated by directed lines (or arcs) connecting the circles. Anexample of a state diagram is shown in Figure 3 below.

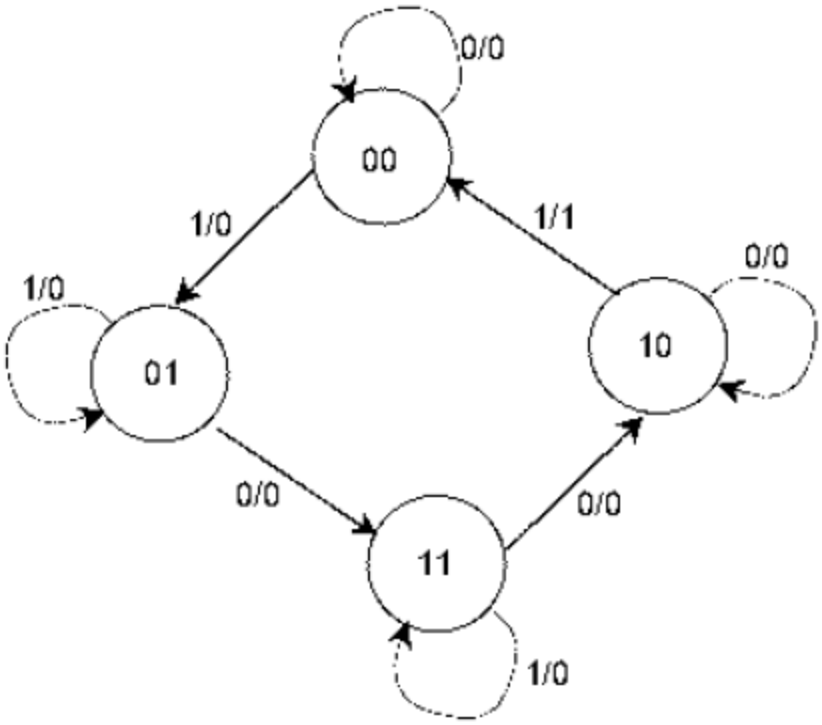


Figure 3. State Diagram

The binary number inside each circle identifies the state the circle represents. The directed lines are labeled with two binary numbers separated by a slash (/). The input value thatcauses the state transition is labeled first. The number after the slash symbol / gives the valueof the output. For example, the directed line from state 00 to 01 is labeled 1/0, meaning that,if the sequential circuit is in a present state and the input is 1, then the next state is 01 and theoutput is 0. If it is in a present state 00 and the input is 0, it will remain in that state. Adirected line connecting a circle with itself indicates that no change of state occurs. The statediagram provides exactly the same information as the state table and is obtained directly fromthe state table.

Consider a sequential circuit shown in Figure 4. It has one input x, one output Z and two statevariables Q1Q2 (thus having four possible present states 00, 01, 10, 11).

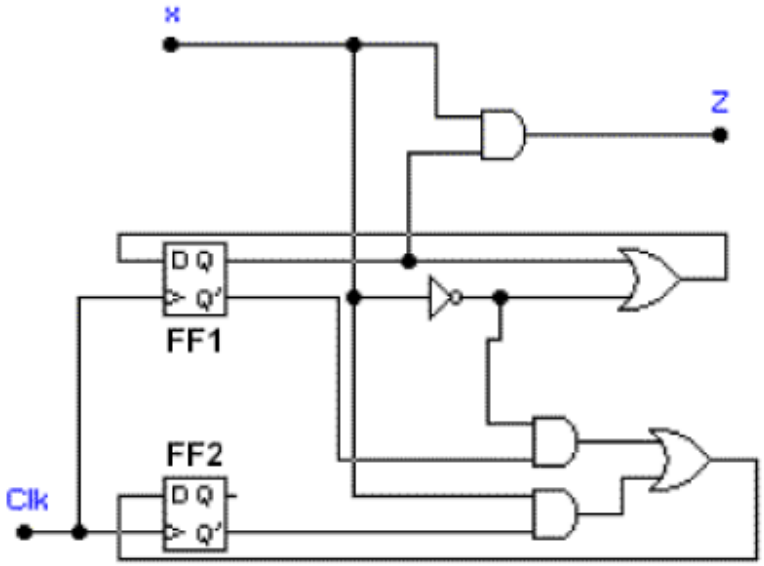


Figure 4. A Sequential Circuit

The behaviour of the circuit is determined by the following Boolean expressions:

Z = x*Q1
D1 = x' + Q1
D2 = x*Q2' + x'*Q1'

These equations can be used to form the state table. Suppose the present state (i.e. Q1Q2) =00 and input x = 0. Under these conditions, we get Z = 0, D1 = 1, and D2 = 1. Thus the nextstate of the circuit D1D2 = 11, and this will be the present state after the clock pulse has beenapplied. The output of the circuit corresponding to the present state Q1Q2 = 00 and x = 1 is Z= 0. This data is entered into the state table as shown in Table .

Present State	Next State		Output	
Q1Q2	x = 0	x = 1	x = 0	x = 1
0 0	1 1	0 1	0	0
0 1	1 1	0 0	0	0
1 0	1 0	1 1	0	1
1 1	1 0	1 0	0	1

State table for the sequential circuit in Figure 4

The state diagram for the sequential circuit in Figure 4 is shown in Figure 5.

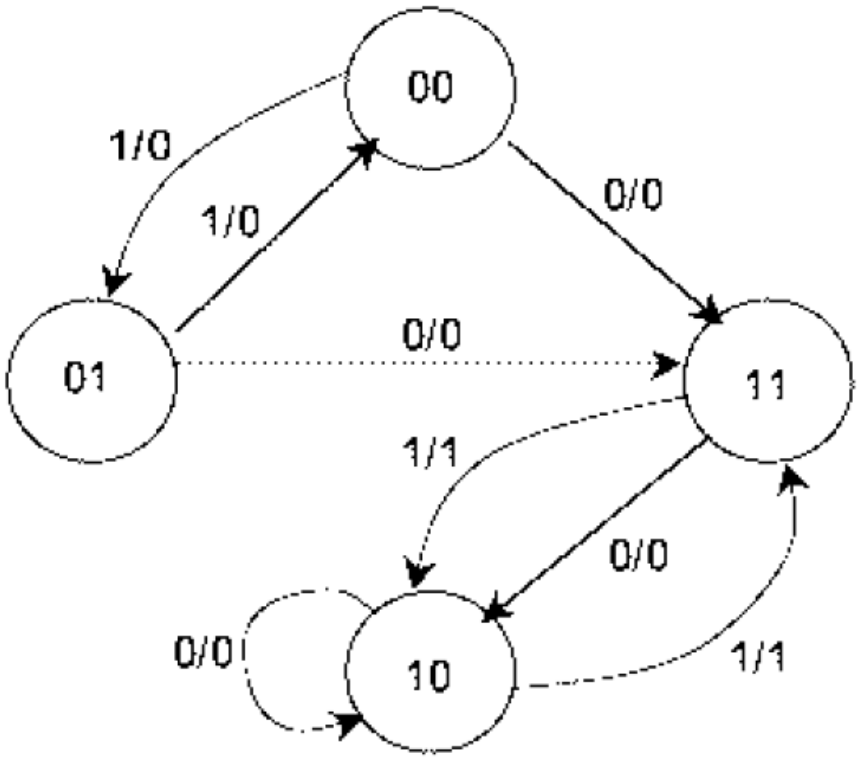


Figure 5. State Diagram of circuit in Figure 4.

MODULE-IV

Memory & Programmable logic

The important common element of the memories we will study is that they are random accessmemories, or RAM. This means that each bit of information can be individually stored orretrieved | with a valid input address. This is to be contrasted with sequential memories inwhich bits must be stored or retrieved in a particular sequence, for example with data storageon magnetic tape. Unfortunately the term RAM has come to have a more specific meaning: Amemory for which bits can both be easily stored or retrieved ("written to" or "read from").

Classification of memories

RAM.

In general, refers to random access memory. All of the devices we are considering to be "memories" (RAM, ROM, etc.) are random access. The term RAM has alsocome to mean memory which can be both easily written to and read from. There aretwo main technologies used for RAM:

Static RAM.

These essentially are arrays of flip-flops. They can be fabricated in ICs as large arrays of tint flip-flops.) "SRAM" is intrinsically somewhat faster than dynamic RAM.

Dynamic RAM.

Uses capacitor arrays. Charge put on a capacitor will produce a HIGH bit if its voltage V = Q=C exceeds the threshold for the logic standard in use. Since the charge will "leak" through the resistance of the connections in times of order 1 msec, the stored informationmust be continuously refreshed (hence the term \dynamic"). Dynamic RAM can be fabricatedwith more bits per unit area in an IC than static RAM. Hence, it is usually the technology ofchoice for most large-scale IC memories.

Read-only memory.

Information cannot be easily stored. The idea is that bits are initially stored and arenever changed thereafter. As an example, it is generally prudent for the instructions used

toinitialize a computer upon initial power-up to be stored in ROM. The following terms refer toversions of ROM for which the stored bits can be over-written, but not easily.

Programmable ROM.

Bits can be set on a programming bench by burning fusible links, or equivalent. Thistechnology is also used for programmable array logic (PALs), which we will briefly discussin class.

ROM Organization

A circuit for implementing one or more switching functions of several variables was described in the preceding section and illustrated in Figure 20. The components of the circuit are

• An n × 2n decoder, with n input lines and 2n output lines

• One or more OR gates, whose outputs are the circuit outputs

• An interconnection network between decoder outputs and OR gate inputs

The decoder is an MSI circuit, consisting of 2n n-input AND gates, that produces all theminterms of n variables. It achieves some economy of implementation, because the same decoder can be used for any application involving the same number of variables.What isspecial to any application is the number of OR gates and the specific outputs of the decoderthat become inputs to those OR gates. Whatever else can be done to result in a general-purpose circuit would be most welcome. The most general-purpose approach is to include the maximum number of OR gates, with provision to interconnect all 2n outputs of the decoder with the inputs to every one of the OR gates. Then, for any given application, two thingswould have to be done:

• The number of OR gates used would be fewer than the maximum number, the othersremaining unused.

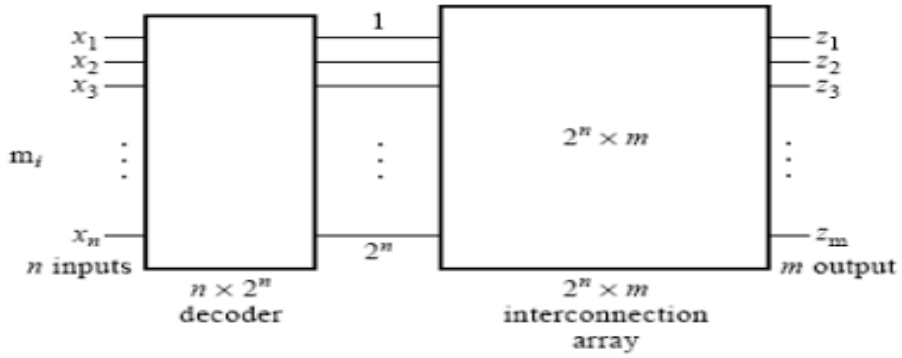
• Not every decoder output would be connected to all OR gate inputs. This scheme would beterribly wasteful and doesn‘t sound like a good idea. Instead, suppose a smaller number, m, isselected for the number of OR gates to be included, and an interconnection network is set upto interconnect the 2n decoder outputs to the m OR gate inputs. Such a structure is illustrate in above figure. It is an LSI combinational circuit with n inputs and m outputs that, for reasonsthat will become clear shortly, is called a read-only memory (ROM).

A ROM consists of two parts:

• An n × 2n decoder

• A 2n × m array of switching devices that form interconnections between the 2n lines fromthe decoder and the m output lines The 2n output lines from the decoder are called the wordlines. Each of the 2n combinations that constitute the inputs to the interconnection arraycorresponds to a minterm and specifies an address.The memory consists of those connectionsthat are actually made in the connection matrix between the word lines and the output lines.Once made, the connections in the memory array are permanent. So this memory is not one whose contents can be changed readily from time to time; we ―write‖ into this memory but once. However, it is possible to ―read‖ the information already stored (the connections actually made) as often as desired, by applying input words and observing the outputwords.That's why the circuit is called read-only memory. Before you continue reading, thinkof two possible ways in which to fabricate a ROM so that one set of connections can be madeand another set leftunconnected.Continue reading after you have thought about it.

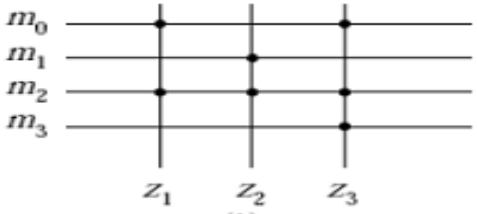
A ROM can be almost completely fabricated except that none of the connections are made.Such a ROM is said to be blank. Forming the connections for a particular application is calledprogramming the ROM. In the process of programming the ROM, a mask is produced tocover those connections that are not to be made. For this reason, the blank form of the ROMis called mask programmable



Basic structure of a ROM.

x_1	x_2	z_1	z_2	z_3
0	0	1	0	1
0	1	0	1	0
1	0	1	1	1
1	1	0	0	1

(a)

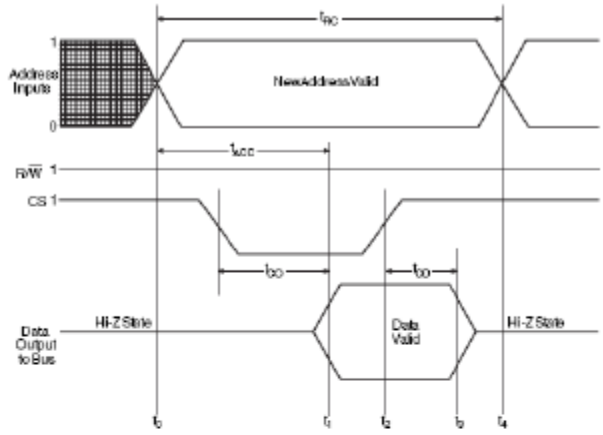


(b)

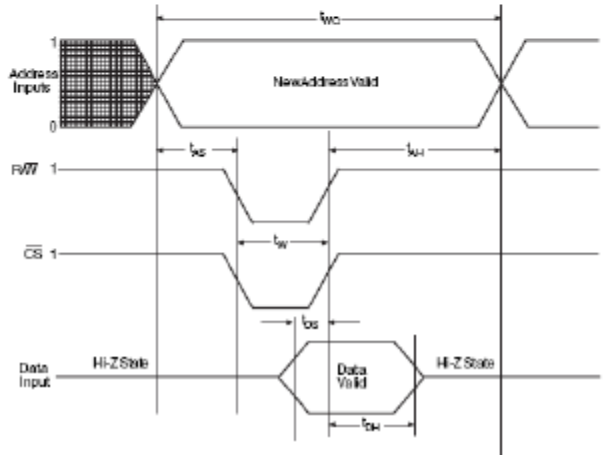
A ROM truth table and its program.

Random Access Memory

RAM has three basic building blocks, namely an array of memory cells arranged inrows and columns with each memory cell capable of storing either a ‘0‘ or a ‘1‘, an addressdecoder and a read/write control logic. Depending upon the nature of the memory cell used,there are two types of RAM, namely static RAM (SRAM) and dynamic RAM (DRAM). InSRAM, the memory cell is essentially a latch and can store data indefinitely as long as theDC power is supplied. DRAM on the other hand, has a memory cell that stores data in theform of charge on a capacitor. Therefore, DRAM cannot retain data for long and hence needsto be refreshed periodically. SRAM has a higher speed of operation than DRAM but has asmaller storage capacity



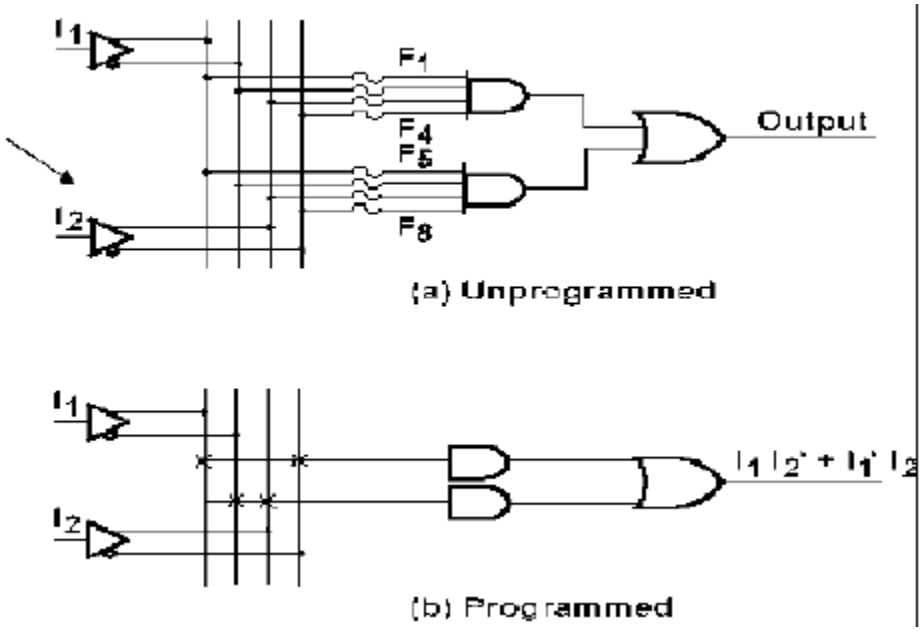
(a)



PROGRAMMABLE ARRAY LOGIC

The PAL device is a special case of PLA which has a programmable AND arrayand afixed OR array. The basic structure of Rom is same as PLA. It is cheap comparedto PLA asonly the AND array is programmable. It is also easy to program a PALcompared to PLA asonly AND must be programmed.

The figure 1 below shows a segment of an unprogrammed PAL. The input bufferwithnon inverted and inverted outputs is used, since each PAL must drive many ANDGatesinputs. When the PAL is programmed, the fusible links (F1, F2, F3…F8) areselectivelyblown to leave the desired connections to the AND Gate inputs. Connectionsto the AND Gateinputs in a PAL are represented byXs, as shown here:



Fixed Logic Versus Programmable Logic

Logic devices can be classified into two broad categories - fixed and programmable. As the name suggests, the circuits in a fixed logic device are permanent, they perform one function or set of functions - once manufactured, they cannot be changed. On the other hand,programmable logic devices (PLDs) are standard, off-the-shelf parts that offer customers awide range of logic capacity, features, speed, and voltage characteristics- and these devicescan be changed at any time to perform any number of functions.With fixed logic devices, the time required to go from design, to prototypes, to a finalmanufacturing

run can take from several months to more than a year, depending on thecomplexity of the device. And, if the device does not work properly, or if the requirementschange, a new design must be developed. The up-front work of designing and verifying fixedlogic devices involves substantial "non-recurring engineering" costs, or NRE.

HDL

In electronics, a hardware description language or HDL is any language from a classof computer languages and/or programming languages for formal description of digital logicand electronic circuits. It can describe the circuit's operation, its design and organization, andtests to verify its operation by means of simulation.HDLs are standard text-based expressions of the spatial and temporal structure and behavior of electronic systems. In contrast to a software programming language, HDL syntax andsemantics include explicit notations for expressing time and concurrency, which are theprimary attributes of hardware. Languages whose only characteristic is to express circuitconnectivity between hierarchies of blocks are properly classified as netlist languages used onelectric computer-aided design (CAD).

HDLs are used to write executable specifications of some piece of hardware. A simulationprogram, designed to implement the underlying semantics of the language statements,coupled with simulating the progress of time, provides the hardware designer with the abilityto model a piece of hardware before it is created physically. It is this executability that givesHDLs the illusion of being programming languages. Simulators capable of supportingdiscrete-event (digital) and continuous-time (analog) modeling exist, and HDLs targeted foreach are available.

Design using HDL

The vast majority of modern digital circuit design revolves around an HDL description of thedesired circuit, device, or subsystem.Most designs begin as a written set of requirements or a high-level architectural diagram. Theprocess of writing the HDL description is highly dependent on the designer's background andthe circuit's nature. The HDL is merely the 'capture language'—often begin with a high-levelalgorithmic description such as MATLAB or a C++ mathematical model. Control anddecision structures are often prototyped in flowchart applications, or entered in a state-diagram editor. Designers even use scripting languages (such as Perl) to automaticallygenerate repetitive circuit structures in the HDL language. Advanced text editors (such asEmacs) offer editor templates for automatic indentation, syntax-dependent coloration, andmacro-based expansion of entity/architecture/signal declaration.

HDLs may or may not play a significant role in the back-end flow. In general, as the designflow progresses toward a physically realizable form, the design database becomesprogressively more laden with technology-specific information, which cannot be stored in ageneric HDL-description. Finally, a silicon chip is manufactured in a fab.

VHDL Operators
Highest precedence first,
left to right within same precedence group,
use parenthesis to control order.
Unary operators take an operand on the right.
"result same" means the result is the same as the right operand.
Binary operators take an operand on the left and right.
"result same" means the result is the same as the left operand.

**** exponentiation, numeric ** integer, result numeric**
abs absolute value, abs numeric, result numeric
not complement,not logic or boolean, result same

*** multiplication, numeric * numeric, result numeric**
/ division,numeric / numeric, result numeric
mod modulo,integer mod integer, result integer
rem remainder,integer rem integer, result integer

+ unary plus,+ numeric, result numeric
- unary minus, - numeric, result numeric

+ addition,numeric + numeric, result numeric
- subtraction, numeric - numeric, result numeric
& concatenation, array or element & array or element,result array

sll shift left logical, logical array sll integer, result same
srl shift right logical, logical array srl integer, result same
sla shift left arithmetic, logical array sla integer, result same
sra shift right arithmetic, logical array sra integer, result same
rol rotate left,logical array rol integer, result same
ror rotate right,logical array ror integer, result same

= test for equality, result is boolean
/= test for inequality, result is boolean
< test for less than, result is boolean
<= test for less than or equal, result is boolean
> test for greater than, result is boolean
>= test for greater than or equal, result is boolean

and logical and, logical array or boolean, result is same
or logical or, logical array or boolean, result is same
nand logical complement of and, logical array or boolean, result is same
nor logical complement of or, logical array or boolean, result is same
xor logical exclusive or, logical array or boolean, result is same
xnor logical complement of exclusive or, logical array or boolean, result is same

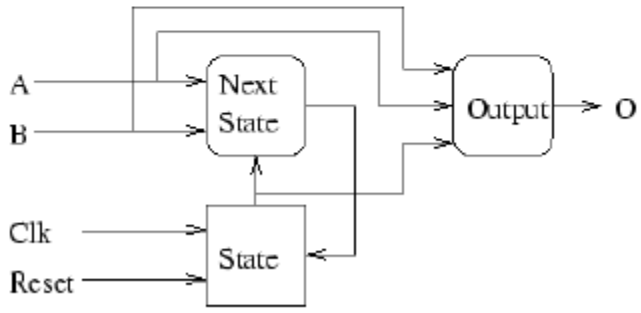
VHDL for Serial Comparator

Things to observe:

1. Flip-flop implementation: reset priority, event, rising edge sensitive.
2. If and case -- sequential statements -- are valid only within a process.
3. Concurrent assignment is a ``process."
4. Semantics of a process: sensitivity list, assignments:
5. b <= a;
6. c <= b;

   does not behave as it would in C.

7. VHDL architecture broken into three processes:
   1. State storage.
   2. Next state generation.
   3. Output generation.



Compare process inputs to sensitivity lists.

```
-- VHDL for serial comparator.  The inputs a and b are input lsb first.
-- The Mealy machine uses rising edge sensitive flip-flops and an
-- asynchronous active low reset.
--
-- The output is 1 if b > a, otherwise 0.


library ieee;
use ieee.std_logic_1164.all;


entity comparator is
  port
  (a, b, clk, reset : in std_logic;
   o                : out std_logic
  );
end comparator;




architecture process_defn of comparator is

  -- Two states needed.
  type state_type is (S0, S1);
  -- State assignment.
  attribute enum_encoding : string;
  attribute enum_encoding of state_type :
    type is "0 1";

  signal state, next_state : state_type;

  -- For convenience, concatenate a and b.
  signal inputs : std_logic_vector (1 downto 0);

begin

  -- Concurrent assignment executes the rhs changes.
  -- Concatenate a and b into inputs.
  inputs <= a & b;


  -- Processes execute whenever something on their sensitivity list
  -- changes.  All assignments take place when the process exits.
  --
  -- This process implements the D flip-flop.
```

```
state_register : process (clk, reset)
begin
  -- If/else construct only valid within a process.
  if (reset = '0') then
    state <= S0;
  elsif (clk'event AND clk = '1') then
    state <= next_state;
  end if;
end process;


-- This process computes the next state.

next_state_process : process (inputs, state)
begin
  case state is

    when S0 =>
      if (inputs = "01") then
        next_state <= S1;
      else
        next_state <= S0;
      end if;




    when S1 =>
      if (inputs = "10") then
        next_state <= S0;
      else
        next_state <= S1;
      end if;

  end case;
end process;


-- This process computes the output.

output_process : process (inputs, state)
begin
  case state is

    when S0 =>
      if (inputs = "01") then
        o <= '1';
      else
        o <= '0';
      end if;
```

```
        when S1 =>
          if (inputs = "10") then
            o <= '0';
          else
            o <= '1';
          end if;

      end case;
    end process;


end process_defn;
```

A test bench is a virtual environment used to verify the correctness or soundness of a design or model (e.g., a software product).

The term has its roots in the testing of electronic devices, where an engineer would sit at a lab bench with tools of measurement and manipulation, such as oscilloscopes, multimeters, soldering irons, wire cutters, and so on, and manually verify the correctness of the device under test.

In the context of software or firmware or hardware engineering, a test bench refers to an environment in which the product under development is tested with the aid of a collection of testing tools. Often, though not always, the suite of testing tools is designed specifically for the product under test.

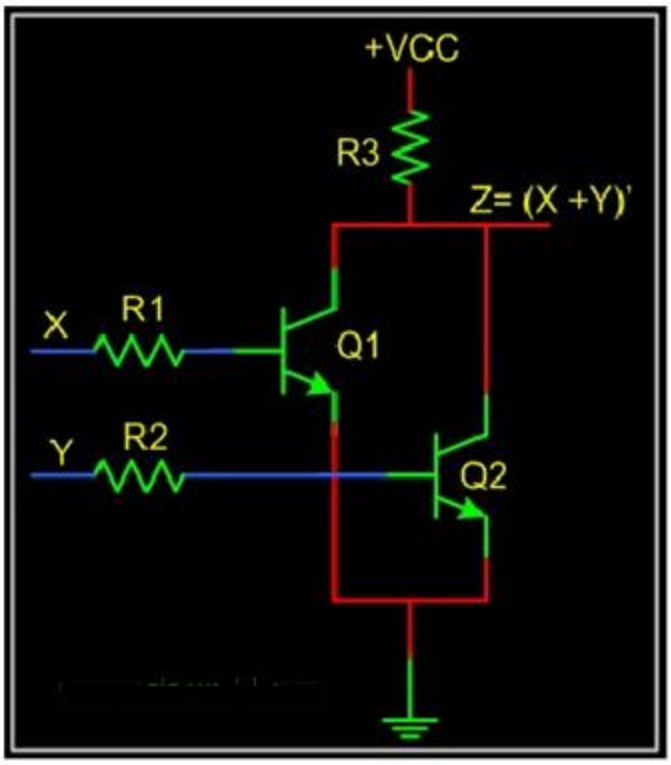
Digital Integrated logic Circuits:

Logic families can be classified broadly according to the technologies they are built with. In earlier dayswe had vast number of these technologies, as you can see in the list below.

RTL : Resistor Transistor Logic.
DTL : Diode Transistor Logic.
TTL : Transistor Transistor Logic.
ECL : Emitter coupled logic.
MOS : Metal Oxide Semiconductor Logic (PMOS and NMOS).
CMOS : Complementary Metal Oxide Semiconductor Logic.

Resistor Transistor Logic.

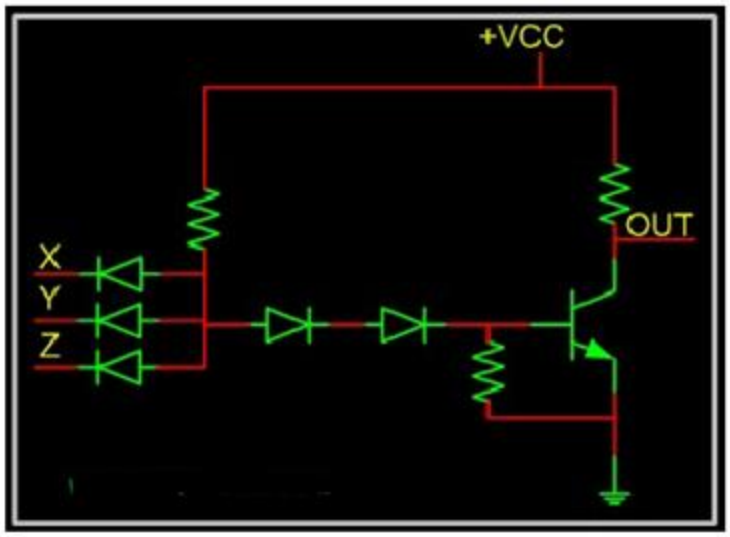
In RTL (resistor transistor logic), all the logic are implemented using resistors and transistors. One basic thing about the transistor (NPN), isthat HIGH at input causes output to be LOW (i.e. like a inverter). Belowis the example of a few RTL logic circuits.



A basic circuit of an RTL NOR gate consists of two transistors Q1 andQ2, connected as the figure above. When either input X or Y isdriven HIGH, the corresponding transistor goes to saturation and outputZ is pulled to LOW.

DTL : Diode Transistor Logic.

In DTL (Diode transistor logic), all the logic is implemented using diodes and transistors. A basic circuit in the DTL logic family is asshown in the figure below. Each input is associated with one diode. Thediodes and the 4.7K resistor form an AND gate. If input X, Y or Z is low, the correspondingdiode conducts current, through the 4.7K resistor.
Thus there is no current through the diodes connected in series to transistor base . Hence the transistor does not conduct, thus remains incut-off, and output out is High. If all the inputs X,Y,Z are driven high, the diodes in series conduct,driving the transistor into saturation. Thus output out is Low.



TTL : Transistor Transistor Logic.

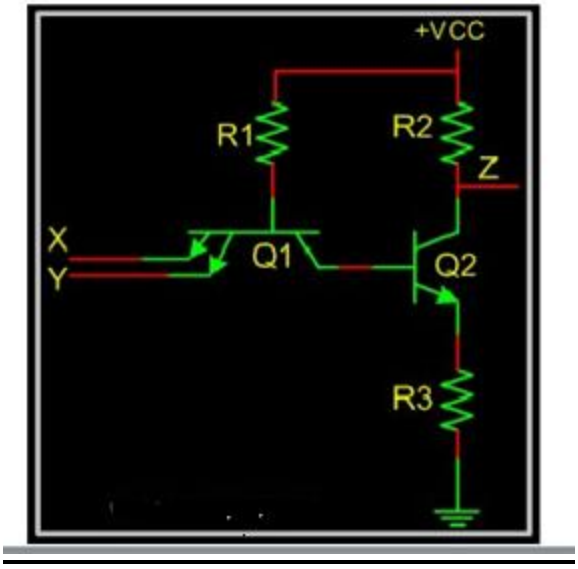
In Transistor Transistor logic or just TTL, logic gates are built only around transistors. TTL was developed in 1965. Through the years basicTTL has been improved to meet performancerequirements. There aremany versions or families of TTL.

- **Standard TTL.**
- **High Speed TTL**
- **Low Power TTL.**
- **Schhottky TTL.**

Here we will discuss only basic TTL as of now; maybe in the future Iwill add more details aboutother TTL versions. As such all TTLfamilies have three configurations for outputs.

- **Totem - Pole output.**
- **Open Collector Output.**
- **Tristate Output.**

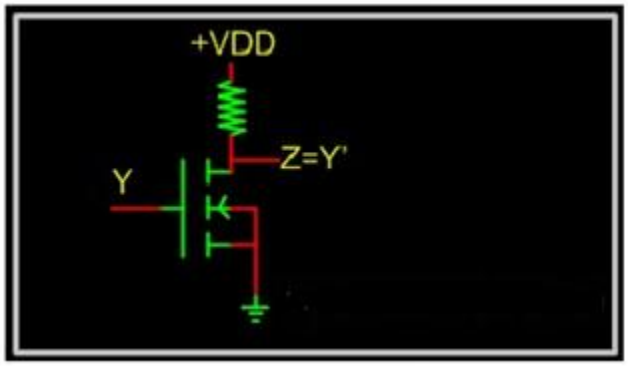
Before we discuss the output stage let's look at the input stage, which is used with almost allversions of TTL. This consists of an input transistorand a phase splitter transistor. Input stage consists of a multi emittertransistor as shown in the figure below. When any input is driven low,the emitter base junction is forward biased and input transistor conducts. This in turn drives the phase splitter transistor into cut-off.



]

MOS : Metal Oxide Semiconductor Logic (PMOS and NMOS).

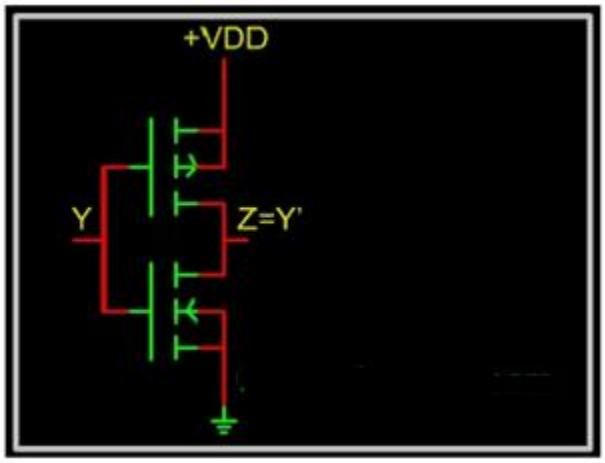
MOS or Metal Oxide Semiconductor logic uses nmos and pmos to implement logic gates. One needs to know the operation of FETand MOS transistors to understand the operation of MOS logic circuits transistor does not conduct, and thus output is HIGH. But when input is HIGH,NMOS transistor conducts and thus output is LOW.



CMOS : Complementary Metal Oxide Semiconductor Logic.

CMOS or Complementary Metal Oxide Semiconductor logic is built using both NMOS and PMOS. Below is the basic CMOSinverter circuit, which follows these rules: NMOS conductswhen its input is HIGH.PMOS conducts when its input is LOW. So when input is

HIGH, NMOS conducts, and thus output is LOW; when input is LOW PMOS conducts and thus output is HIGH.



References

[1] Digital Design,3rd edition by M. Morris Mano, Pearson Education

[2] Digital Design-Principle& practice, 3rd edition by John F. Wakerley, Pears

[3] Digital Electronics by R.Anitha NPRCET