M.Sc. PHYSICS - I YEAR

DKP13 : DIGITAL ELECTRONICS

SYLLABUS

UNIT I Number systems

Binary coded decimal number system, Grey code, Grey code to Binary conversion, Binary to Grey code, Excess 3 code, Decimal to excess 3 code, ASCII code.

Universal logic gates: NAND and NOR gates as universal logic gates – Simplification of logic circuits – De Morgan's laws – Boolean laws – Karnaugh maps – three variable and four variable maps – max and min terms.

UNIT II Arithematic circuits

Half adder – Truth table and circuit – Full adder – Truth table and circuit – Four bit adder – Half subtractor – Full subtractor – Multiplexer: Four input multiplexer – Applications of Multiplexer – demultiplexer – Decoders 2 to 4 decoder – BCD to seven segment decoder – encoders.

UNIT III Flipflops

Introduction – NAND LATCH, J K flipflop – J K Master – slave flipflop – D flipflop and T flipflop – Registers and Counters: Shift registers – serial in – parallelout, serial in – serial out, parallel in – serial out, parallel in – parallel out shift registers – wave forms for the above – Counters – up counters, down counters, decade counters, timing sequences, Mod – n counters.

UNIT IV MULTIVIBRATORS

Classification of multivibrators – Astable, monostable, bistable multivibrators using operational amplifier.

D/A and A/D converters: Binary weighted register D/A converter using Op-Amp – R-2R ladder D/A converter with Op-Amp – Analog to Digital converters (ADC) – their characteristics.

UNIT V SEMICONDUCTOR MEMORIES

Memory cell unit – ROM, RAM – Their classifications – ROM, PROM, EPROM, EEPROM, RAM,Static RAM, dynamic RAM, Memory read and memory write operations – Flash memory - Charge coupled Device (CCD).

Books for Study and Reference:

1. Digital Electronics principles and applications – Soumitra Kumar Mandal - Tata MCGraw Hill publications – New Delhi.
2.Integrated Electronics – Digital and Analog – V.Vijayendran (S.Viswanathan printers and publications) - 2005
3.Digital Electronics by Millman and Taub
4.Electronics Fundamentals and Applications- John D Ryder

MS UNIVERSITY M.Sc., Physics Correspondence Course

Paper Title : DIGITAL ELECTRONICS

UNIT 1 NUMBER SYSTEM Binary coded decimal number system, Grey code, Grey code to Binary conversion, Binary to Grey code, Excess 3 code, Decimal to excess 3 code, ASCII code. Universal logic gates: NAND and NOR gates as universal logic gates – Simplification of logic circuits – De Morgan"s laws – Boolean laws – Karnaugh maps – three variable and four variable maps – max and min terms.

Binary Coded number system

Binary codes are codes which are represented in binary system with modification from the original ones. There are two types of binary codes: Weighted codes and Non-Weighted codes. BCD and the 2421 code are examples of weighted codes. In a weighted code, each bit position is assigned a weighting factor **in such a way that each digit ca n be evaluated by adding the weight of all the 1's** in the coded combination.

- **Weighted Binary Systems**

✓ **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

Weighted Code

– 8421 code
• **Most common**
• **Default**
• **The corresponding decimal digit is determined by adding the weights associated with the 1s in the** code group.
– 62310 = 0110 0010 0011
– **2421, 5421,7536, etc... codes**
• **The** weights associated with the bits in each code group are given by the name of the code

Nonweighted Codes

– 2-out-of-5
Non Weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value.
• **Actually weighted 74210 except for the digit 0**
• **Used by the post office for scanning bar codes for zip codes**
• **Has error detection properties**

- ✓ **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.

- ✓ **Sequential code**

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and Excess-3 codes are sequential, whereas the 2421 and 5211 codes are 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).

Example: 1000 of 8421 = 1011 in Excess-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. In digital Gray code has got a special place.

Decimal Number	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110

12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code.

Important when an analog quantity must be converted to a digital representation. Only one bit changes between two successive integers which are being coded.

✓ Error Detecting and Correction Codes

- **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.

- **Error correcting code**

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.

- **Hamming codes**

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-bit errors and cannot distinguish between 1-bit and 2-bits inconsistencies. It can't - in general - detect 3(or more)-bits errors.

- **Parity codes**

A parity bit is an extra bit included with a message to make the total number of 1's either even or odd. 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.

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 **one's** 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 the number of **one's** is even then parity bit is set to 1.

✓ Alphanumeric codes

The binary codes that can be used to represent all the letters of the alphabet, numbers and mathematical symbols, punctuation marks, are known as alphanumeric codes or character codes. These codes enable us to interface the input-output devices like the keyboard, printers, video displays with the computer.

- **ASCII codes**

Codes to handle alphabetic and numeric information, special symbols, punctuation marks, and control characters.

• **ASCII (American Standard Code for** Information Interchange) is the best known.

• **Unicode** – a 16-bit coding system provides for foreign languages, mathematical symbols, geometrical shapes, dingbats, etc. It has become a world standard alphanumeric code for microcomputers and computers. It is a 7-bit code representing $2^7 = 128$ 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.

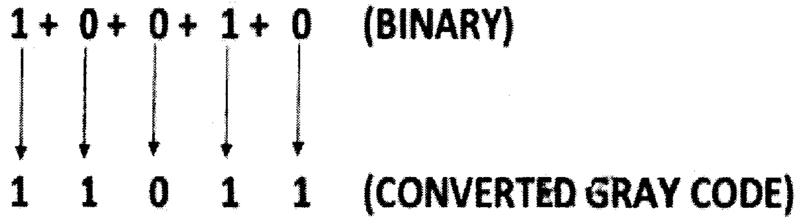
- **EBCDIC codes**

EBCDIC stands for Extended Binary Coded Decimal Interchange. It is mainly used with large computer systems like mainframes. EBCDIC is an 8-bit code and thus accommodates up to 256 characters. An EBCDIC code is divided into two portions: 4 zone bits (on the left) and 4 numeric bits (on the right).

Example 1: Give the binary, BCD, Excess-3, gray code representations of numbers: 5,8,14.

Decimal Number	Binary code	BCD code	Excess-3 code	Gray code
5	0101	0101	1000	0111
8	1000	1000	1011	1100
14	1110	0001 0100	0100 0111	1001

Example 2: Binary To Gray Code Conversion



Example 3: Gray Code To Binary Code Conversion

1.7 BOOLEAN ALGEBRA AND THEOREMS

Ref: 1) **A.P Godse & D.A Godse "Digital Electronics", Technical publications, Pune, Revised third** edition, 2008. Pg.No:2.1-2.10
2) Morris Mano M. and Michael D. Ciletti, **"Digital Design", IV Edition, Pearson Education,** 2008.Pg.No:36-44.

In 1854, George Boole developed an algebraic system now called *Boolean algebra.* In 1938, C. E. Shannon introduced a two-valued Boolean algebra called *switching algebra* that represented the properties of bistable electrical switching circuits.

Boolean algebra is an algebraic structure defined by a set of elements B, together with two binary operators. '+' and '-', provided that the following (Huntington) postulates are satisfied;

Principle of Duality

It states that every algebraic expression is deducible from the postulates of Boolean algebra, and it remains valid if the operators & identity elements are interchanged. If the inputs of a NOR gate are inverted we get a AND equivalent circuit. Similarly when the inputs of a NAND gate are inverted, we get a OR equivalent circuit.

1. Interchanging the OR and AND operations of the expression.
2. Interchanging the 0 and 1 elements of the expression.
3. Not changing the form of the variables.

Theorems of Boolean algebra:

The theorems of Boolean algebra can be used to simplify many a complex Boolean expression 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 the theorem 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.

T1: Commutative Law

(a) $A + B = B + A$

(b) $A B = B A$

T2: Associative Law

(a) $(A + B) + C = A + (B + C)$

(b) $(A B) C = A (B C)$

T3: Distributive Law

(a) $A (B + C) = A B + A C$

(b) $A + (B C) = (A + B) (A + C)$

T4: Identity Law

(a) $A + A = A$

(b) $A A = A$

T5: Negation Law.

- and

= () =

T6: Redundancy

(a) $A + A B = A$

(b) $A (A + B) = A$

T7: Operations with ‘0’ & ‘1’

(a) $0 + A = A$

(b) $1 A = A$

(c) $1 + A = 1$

(d) $0 A = 0$

T10: De Morgan's 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 maȳ+beexpressed=̣ by the following Boolean expression.
- **It states that the —Complement of the product of variables is equal to the sum of complements** of each individual variable. Boolean expressioñfor =this̄theorem+ is

Order of Precedence

NOT operations have the highest precedence, followed by AND operations, followed by OR operations. Brackets can be used as with other forms of algebra.

e.g. X.Y + Z and X.(Y + Z) are not the same function.

Truth Tables

Truth tables are a means of representing the results of a logic function using a table. They are constructed by defining all possible combinations of the inputs to a function, and then calculating the output for each combination in turn.

AND

X	Y	F(X,Y)
0	0	0
0	1	0
1	0	0
1	1	1

NOT

X	F(X)
0	1
1	0

OR

X	Y	F(X,Y)
0	0	0
0	1	1
1	0	1
1	1	1

Minterms and maxterms

A binary variable may appear either in its normal form *(x)* or in its complement form (x'). Now consider two binary variables *x* and *y* combined with an AND operation. Since each variable may appear in either form, there are four possible combinations: *x' y', x'y. xy '*, and *xy*. Each of these four AND term s is called a ***minterm***, or a *standard product.*

In a similar fashion, n variables forming g an OR term with each variable being primed or Unprimed provide 2" possible combinations called ***maxterm***. or *standard* sums.

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

For a two-variable expression, the minterms and maxterms are as follows

X	Y	Minterm	Maxterm
0	0	X'.Y'	X+Y
0	1	X'.Y	X+Y'
1	0	X.Y'	X'+Y
1	1	X.Y	X'+Y'

For a three-variable expression, the minterms and maxterms are as follows

X	Y	Z	Minterm	Maxterm
0	0	0	X'.Y'.Z'	X+Y+Z

0	0	1	X'.Y'.Z	X+Y+Z'
0	1	0	X'.Y.Z'	X+Y'+Z
0	1	1	X'.Y.Z	X+Y'+Z'
1	0	0	X.Y'.Z'	X'+Y+Z
1	0	1	X.Y'.Z	X'+Y+Z'
1	1	0	X.Y.Z'	X'+Y'+Z
1	1	1	X.Y.Z	X'+Y'+Z'

This allows us to represent expressions in either Sum of Products or Product of Sums forms

Sum Of Products (SOP): F(X, Y, ...) = Sum (ak.mk), where ak is 0 or 1 and mk is a minterm.
To derive the Sum of Products form from a truth table, OR together all of the minterms which give a value of 1.Consider the truth table as example,

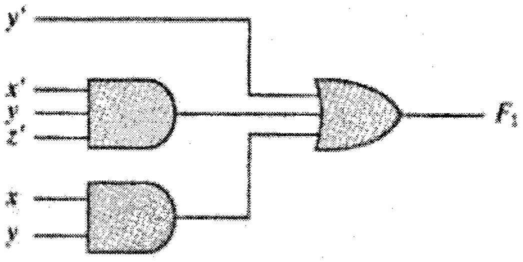
X	Y	F	Minterm
0	0	0	X'.Y'
0	1	0	X'Y
1	0	1	X.Y'
1	1	1	X.Y

Here SOP is f(X.Y) = X.Y' + X.Y

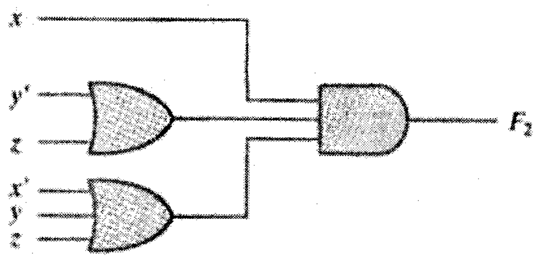
Product Of Sum (POS): The Product of Sums form represents an expression as a product of maxterms.F(X, Y, ...) = Product (bk + Mk), where bk is 0 or 1 and Mk is a maxterm. To derive the Product of Sums form from a truth table, AND together all of the maxterms which give a value of 0.Consider the truth table from the previous example

X	Y	F	Maxterm
0	0	1	X+Y
0	1	0	X+Y'
1	0	1	X'+Y
1	1	1	X'+Y'

Here POS is F(X,Y) = (X+Y')



(a) Sum of Products



(b) Product of Sums

Conversion between POS and SOP: Conversion between the two forms is done by application of DeMorgans Laws.

DIGITAL LOGIC GATES

A logic gate is an electronic circuit/device which makes the logical decisions. To arrive at this decisions, the most common logic gates used are OR, AND, NOT, NAND, and NOR gates. The NAND and NOR gates are called universal gates. The exclusive-OR gate is another logic gate which can be constructed using AND, OR and NOT gate.

Logic gates have one or more inputs and only one output. The output is active only for certain input combinations. Logic gates are the building blocks of any digital circuit. Logic gates are also called switches. With the advent of integrated circuits, switches have been replaced by TTL (Transistor Transistor Logic) circuits and CMOS circuits. Here I give example circuits on how to construct simples gates.

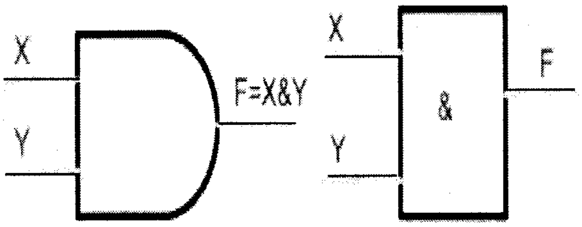
- **AND**
- **OR**
- **NOT**
- **BUF**
- **NAND**
- **NOR**
- **XOR**
- **XNOR**

AND Gate

The AND gate performs logical multiplication, commonly known as AND function. The AND gate has two or more inputs and single output. The output of AND gate is HIGH only when all its inputs are HIGH (i.e. even if one input is LOW, Output will be LOW).

If X and Y are two inputs, then output F can be represented mathematically as F = X.Y, Here dot (.) denotes the AND operation. Truth table and symbol of the AND gate is shown in the figure below.

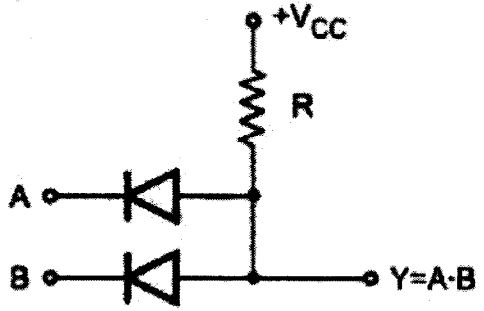
Symbol



Truth Table

X	Y	F(X,Y)
0	0	0
0	1	0
1	0	0
1	1	1

Two input AND gate using "diode-resistor" logic is shown in figure below, where X, Y are inputs and F is the output.



If X = 0 and Y = 0, then both diodes D1 and D2 are forward biased and thus both diodes conduct and pull F low.

If X = 0 and Y = 1, D2 is reverse biased, thus does not conduct. But D1 is forward biased, thus conducts and thus pulls F low.

If X = 1 and Y = 0, D1 is reverse biased, thus does not conduct. But D2 is forward biased, thus conducts and thus pulls F low.

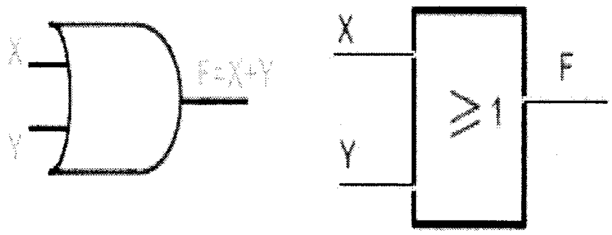
If X = 1 and Y = 1, then both diodes D1 and D2 are reverse biased and thus both the diodes are in cut-off and thus there is no drop in voltage at F. Thus F is HIGH.

OR Gate

The OR gate performs logical addition, commonly known as OR function. The OR gate has two or more inputs and single output. The output of OR gate is HIGH only when any one of its inputs are HIGH (i.e. even if one input is HIGH, Output will be HIGH).

If X and Y are two inputs, then output F can be represented mathematically as F = X+Y. Here plus sign (+) denotes the OR operation. Truth table and symbol of the OR gate is shown in the figure below.

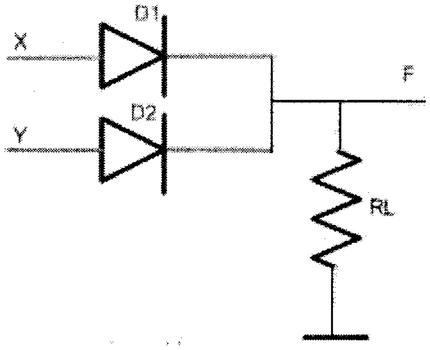
Symbol



Truth Table

X	Y	F(X,Y)
0	0	0
0	1	1
1	0	1
1	1	1

Two input OR gate using "diode-resistor" logic is shown in figure below, where X, Y are inputs and F is the output.



If X = 0 and Y = 0, then both diodes D1 and D2 are reverse biased and thus both the diodes are in cut-off and thus F is low.

If X = 0 and Y = 1, D1 is reverse biased, thus does not conduct. But D2 is forward biased, thus conducts and thus pulling F to HIGH.

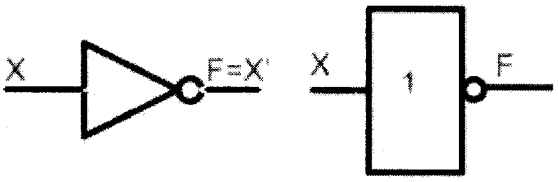
If X = 1 and Y = 0, D2 is reverse biased, thus does not conduct. But D1 is forward biased, thus conducts and thus pulling F to HIGH.

If X = 1 and Y = 1, then both diodes D1 and D2 are forward biased and thus both the diodes conduct and thus F is HIGH.

NOT Gate

The NOT gate performs the basic logical function called inversion or complementation. NOT gate is also called inverter. The purpose of this gate is to convert one logic level into the opposite logic level. It has one input and one output. When a HIGH level is applied to an inverter, a LOW level appears on its output and vice versa.

Symbol

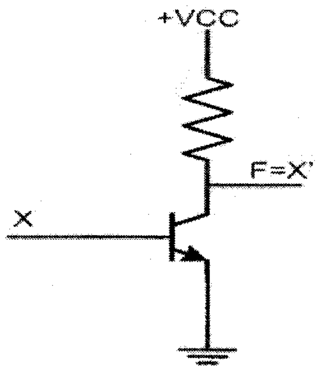


Truth Table

X	F(X)
0	1
1	0

If X is the input, then output F can be represented mathematically as F = X', Here apostrophe (') denotes the NOT (inversion) operation. There are a couple of other ways to represent inversion, F= !X, here ! represents inversion. Truth table and NOT gate symbol is shown in the figure below.

NOT gate using "transistor-resistor" logic is shown in the figure below, where X is the input and F is the output.



When X = 1, The transistor input pin 1 is HIGH, this produces the forward bias across the emitter base junction and so the transistor conducts. As the collector current flows, the voltage drop across RL increases and hence F is LOW.

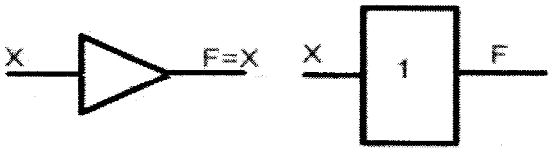
When X = 0, the transistor input pin 2 is LOW: this produces no bias voltage across the transistor base emitter junction. Thus Voltage at F is HIGH.

BUF Gate

Buffer or BUF is also a gate with the exception that it does not perform any logical operation on its input. Buffers just pass input to output. Buffers are used to increase the drive strength or sometime just to introduce delay. We will look at this in detail later.

If X is the input, then output F can be represented mathematically as F = X. Truth table and symbol of the Buffer gate is shown in the figure below.

Symbol



Truth Table

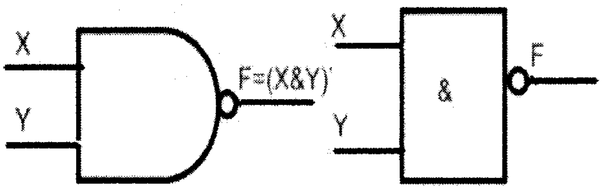
X	F(X)
0	0
1	1

NAND Gate

NAND gate is a cascade of AND gate and NOT gate, as shown in the figure below. It has two or more inputs and only one output. The output of NAND gate is HIGH when any one of its input is LOW (i.e. even if one input is LOW, Output will be HIGH).

If X and Y are two inputs, then output F can be represented mathematically as F = (X.Y)', Here dot (.) denotes the AND operation and (') denotes inversion. Truth table and symbol of the N AND gate is shown in the figure below.

Symbol



Truth Table

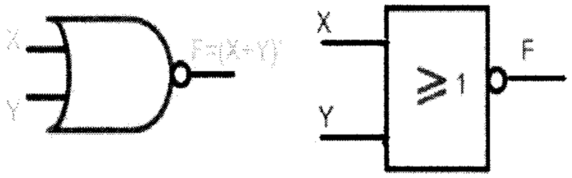
X	Y	F(X,Y)
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

NOR gate is a cascade of OR gate and NOT gate, as shown in the figure below. It has two or more inputs and only one output. The output of NOR gate is HIGH when any all its inputs are LOW (i.e. even if one input is HIGH, output will be LOW).

Truth Table

Symbol



X	Y	F(X,Y)
0	0	1
0	1	0
1	0	0
1	1	0

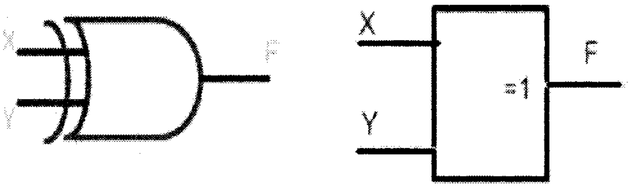
XOR Gate

An Exclusive-OR (XOR) gate is gate with two or three or more inputs and one output. The output of a two-input XOR gate assumes a HIGH state if one and only one input assumes a HIGH state. This is equivalent to saying that the output is HIGH if either input X or input Y is HIGH exclusively, and LOW when both are 1 or 0 simultaneously.

If X and Y are two inputs, then output F can be represented mathematically as F = X ■Y, Here ■ denotes the XOR operation. X ■Y and is equivalent to X.Y' + X'.Y. Truth table and symbol of the XOR gate is shown in the figure below.

Truth Table

Symbol



X	Y	F(X,Y)
0	0	0
0	1	1
1	0	1
1	1	0

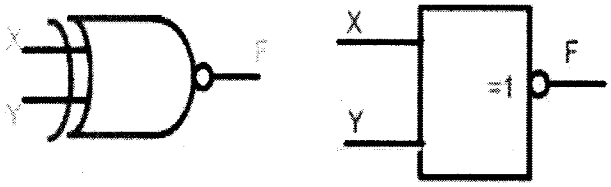
XNOR Gate

An Exclusive-NOR (XNOR) gate is gate with two or three or more inputs and one output. The output of a two-input XNOR gate assumes a HIGH state if all the inputs assumes same state. This is equivalent to

saying that the output is HIGH if both input X and input Y is HIGH exclusively or same as input X and input Y is LOW exclusively, and LOW when both are not same.

If X and Y are two inputs, then output F can be represented mathematically as $F = X \odot Y$, Here $\odot$ denotes the XNOR operation. $X \odot Y$ and is equivalent to $X.Y + X'.Y'$. Truth table and symbol of the XNOR gate is shown in the figure below.

Symbol



Truth Table

X	Y	F(X,Y)
0	0	1
0	1	0
1	0	0
1	1	1

Universal Gates

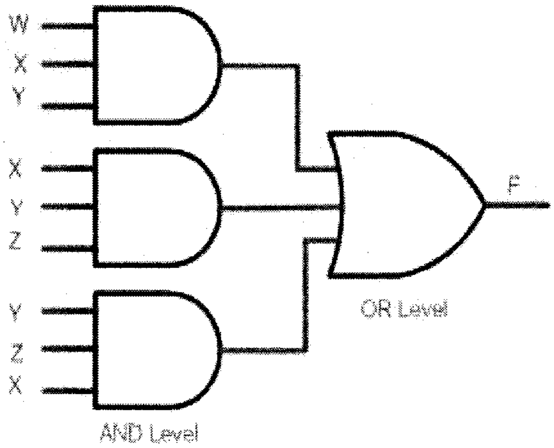
Universal gates are the ones which can be used for implementing any gate like AND, OR and NOT, or any combination of these basic gates; **NAND and NOR gates are universal gates.** But there are some rules that need to be followed when implementing NAND or NOR based gates.

1.6 NAND and NOR implementation

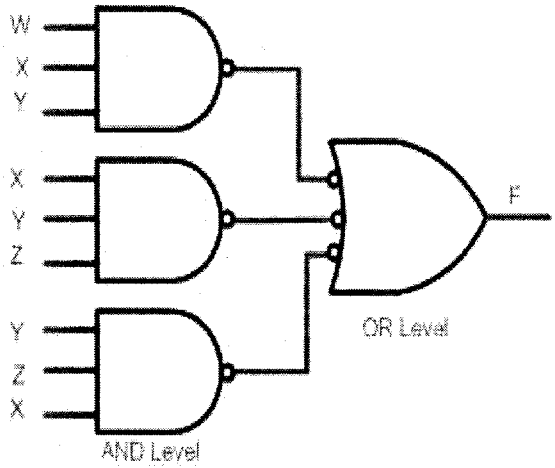
Any logic function can be implemented using NAND gates. To achieve this, first the logic function has to be written in Sum of Product (SOP) form. Once logic function is converted to SOP, then is very easy to implement using NAND gate. In other words any logic circuit with AND gates in first level and OR gates in second level can be converted into a NAND-NAND gate circuit.

Consider the following SOP expression
$F = W.X.Y + X.Y.Z + Y.Z.W$

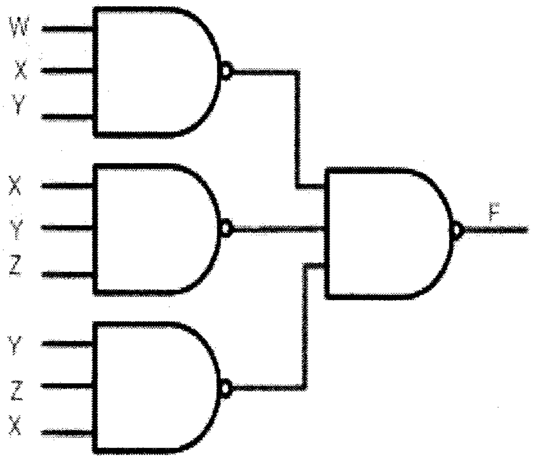
The above expression can be implemented with three AND gates in first stage and one OR gate in second stage as shown in figure.



If bubbles are introduced at AND gates output and OR gates inputs (the same for NOR gates), the above circuit becomes as shown in figure.



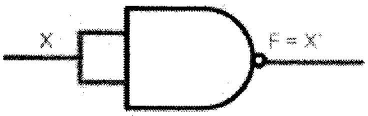
Now replace OR gate with input bubble with the NAND gate. Now we have circuit which is fully implemented with just NAND gates.



✓ Realization of logic gates using NAND gates

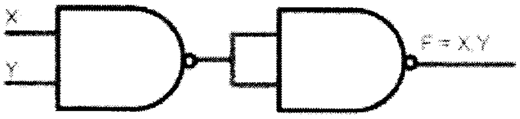
Implementing an inverter using NAND gate

Input	Output	Rule
(X.X)'	= X'	Idempotent



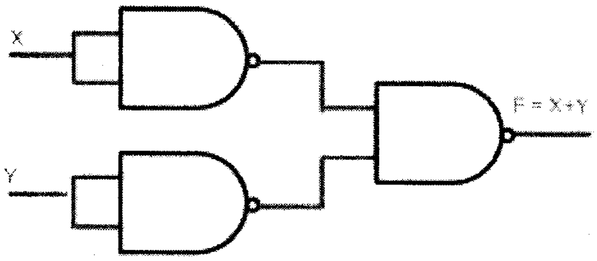
Implementing AND using NAND gates

Input	Output	Rule
((XY)'(XY)')'	= ((XY)')'	Idempotent
	= (XY)	Involution



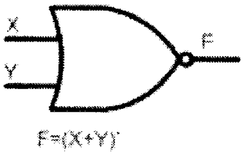
Implementing OR using NAND gates

Input	Output	Rule
((XX)'(YY)')'	= (X'Y')'	Idempotent
	= X"+Y"	DeMorgan
	= X+Y	Involution

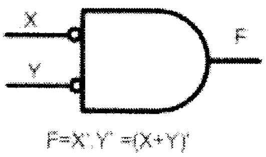


Implementing NOR using NAND gates

Input	Output	Rule
((XX)'(YY)')'	=(X'Y')'	Idempotent
	=X"+Y"	DeMorgan
	=X+Y	Involution
	=(X+Y)'	Idempotent



F=(X+Y)'



F=X'.Y' =(X+Y)'

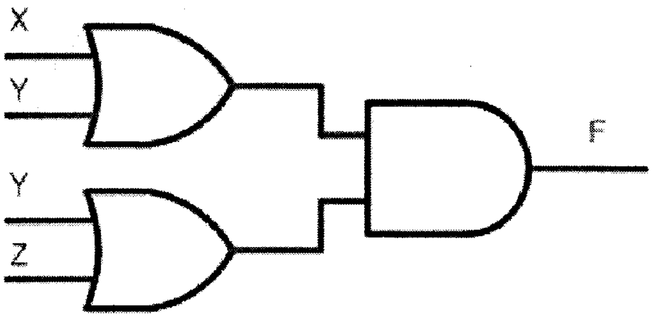
✓ Realization of logic function using NOR gates

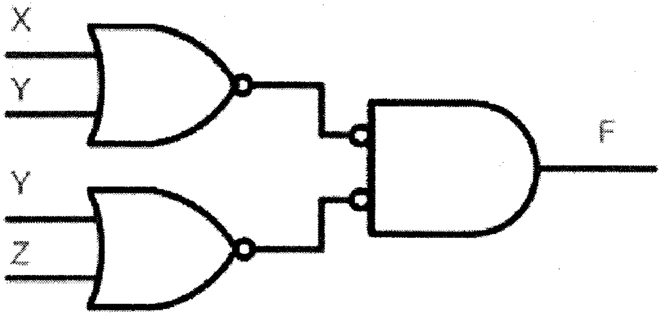
Any logic function can be implemented using NOR gates. To achieve this, first the logic function has to be written in Product of Sum (POS) form. Once it is converted to POS, then it's very easy to implement using NOR gate. In other words any logic circuit with OR gates in first level and AND gates in second level can be converted into a NOR-NOR gate circuit.

Consider the following POS expression

F = (X+Y) . (Y+Z)

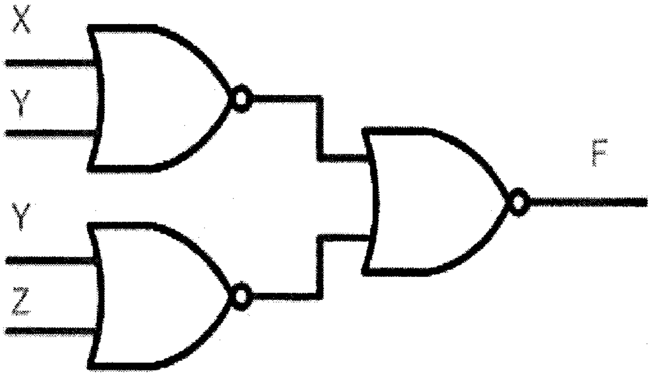
The above expression can be implemented with three OR gates in first stage and one AND gate in second stage as shown in figure.





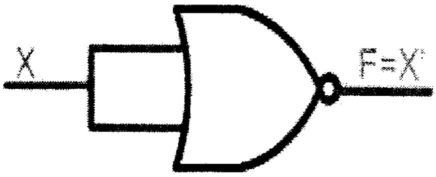
If bubble are introduced at the output of the OR gates and the inputs of AND gate, the above circuit becomes as shown in figure.

Now replace AND gate with input bubble with the NOR gate. Now we have circuit which is fully implemented with just NOR gates.



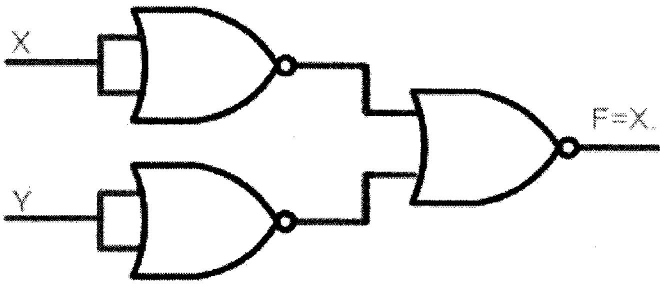
Implementing an inverter using NOR gate

Input	Output	Rule
(X+X)'	= X'	Idempotent



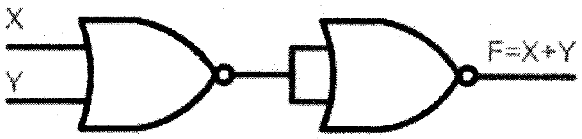
Implementing AND using NOR gates

Input	Output	Rule
((X+X)'+(Y+Y)')'	=(X'+Y')	Idempotent
	= X".Y"	DeMorgan
	= (X.Y)	Involution



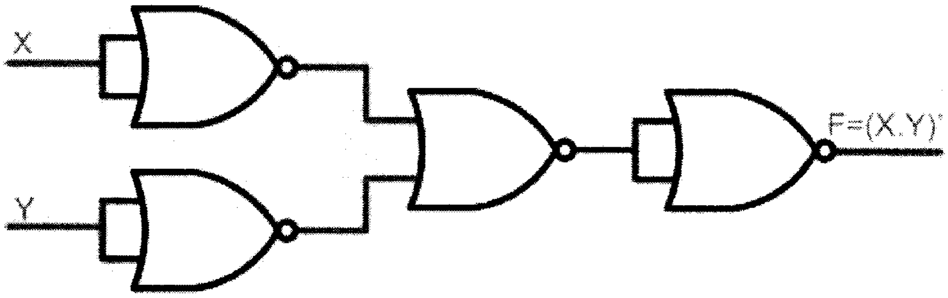
Implementing OR using NOR gates

Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution



Implementing NAND using NOR gates

Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution
	= (X+Y)'	Idempotent



Minimization Technique

The primary objective of all simplification procedures is to obtain an expression that has the minimum number of terms. Obtaining an expression with the minimum number of literals is usually the secondary objective. If there is more than one possible solution with the same 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 is usually called logic minimization and the goal is to form a result which is efficient. Two methods we will discuss are algebraic minimization and Karnaugh maps. For very complicated problems the former method can be done using special software analysis programs. Karnaugh maps are also limited to problems with up to 4 binary inputs. The Quine–McCluskey tabular method is used for more than 4 binary inputs.

1.6 KARNAUGH MAPS

Maurice Karnaugh, a telecommunications engineer, developed the Karnaugh map at Bell Labs in 1953 while designing digital logic based telephone switching circuits. Karnaugh maps reduce logic functions more quickly and easily compared to Boolean algebra.

A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The Karnaugh map can also be described as a special arrangement of a truth table.

Construction of a Karnaugh Map

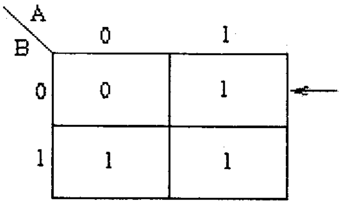
1. **Each square containing a _1‘ must be considered at least once, although it can be** considered as often as desired.
2. The objective should be to account for all the marked squares in the minimum number of groups.
3. The number of squares in a group must always be a power of 2, i.e. groups can have 1, 2, 4_ 8, 16, squares.
4. Each group should be as large as possible, which means that a square should not be accounted for by itself if it can be accounted for by a group of two squares; a group of two squares should not be made if the involved squares can be included in a group of four 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, not necessary **to account for all _don‘t care‘** entries. Only such entries that can be used to advantage should be used.

The diagram below illustrates the correspondence between the Karnaugh map and the truth table for the general case of a two variable problem.

The values inside the squares are copied from the output column of the truth table, therefore there is one square in the map for every row in the truth table. Around the edge of the Karnaugh map are the values of the two input variable. A is along the top and B is down the left hand side. The diagram below explains this:

A	B	F
0	0	0
0	1	1
1	0	1 ←
1	1	1

Truth Table.



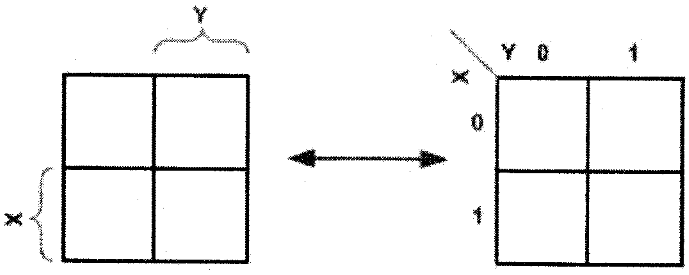
B \ A	0	1
0	0	1 ←
1	1	1

F.

The values around the edge of the map can be thought of as coordinates. So as an example, the square on the top right hand corner of the map in the above diagram has coordinates A=1 and B=0. This square corresponds to the row in the truth table where A=1 and B=0 and F=1. Note that the value in the F column represents a particular function to which the Karnaugh map corresponds.

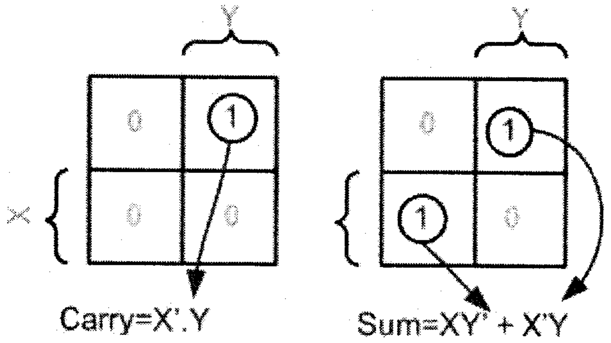
Two variable K-map

There are four minterms for two variables: hence, the map consists of four squares, one for each minterm. In any K-Map, each square represents a minterm. Adjacent squares always differ by just one literal (So that the unifying theorem may apply: X + X' = 1). For the 2-variable case (e.g.: variables X, Y), the map can be drawn as below. Two variable map is the one which has got only two variables as input.



Example- Carry and Sum of a half adder

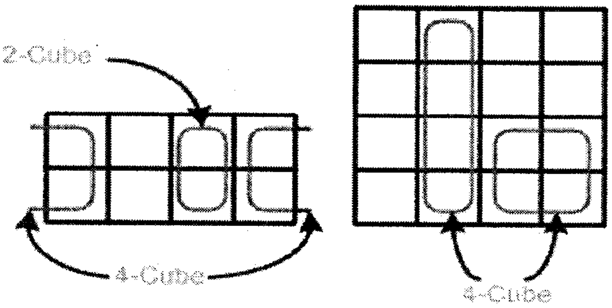
In this example we have the truth table as input, and we have two output functions. Generally we may have n output functions for m input variables. Since we have two output functions, we need to draw two k-maps (i.e. one for each function). Truth table of 1 bit adder is shown below. Draw the k-map for Carry and Sum as shown below.

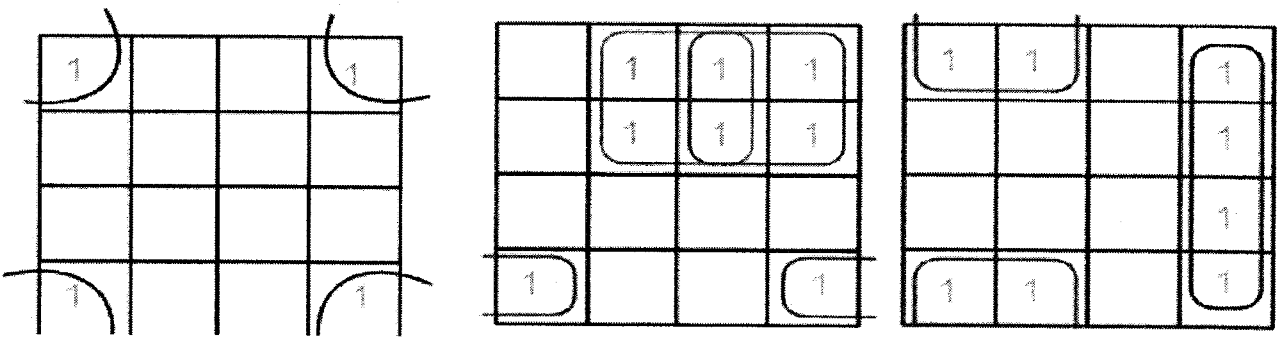


Grouping/Circling K-maps

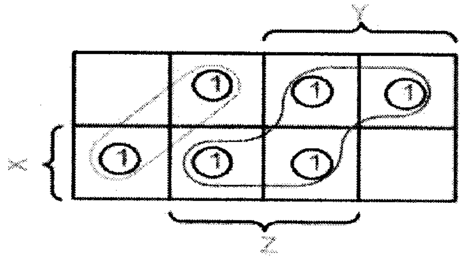
The power of K-maps is in minimizing the terms, K-maps can be minimized with the help of grouping the terms to form single terms. When forming groups of squares, observe/consider the following:

- Every square containing 1 must be considered at least once.
- A square containing 1 can be included in as many groups as desired.
- A group must be as large as possible.
- If a square containing 1 cannot be placed in a group, then leave it out to include in final expression.
- The number of squares in a group must be equal to 2 .i.e. 2,4,8,.
- The map is considered to be folded or spherical, therefore squares at the end of a row or column are treated as adjacent squares.
- The simplified logic expression obtained from a K-map is not always unique. Groupings can be made in different ways.
- Before drawing a K-map the logic expression must be in canonical form.



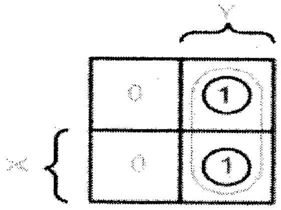


Example of invalid groups



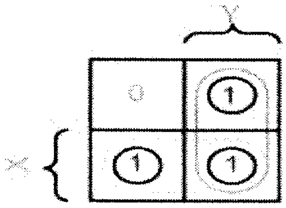
Example (1)- X'Y+XY: In this example we have the equation as input, and we have one output function. Draw the k-map for function F with marking 1 for X'Y and XY position. Now combine two 1's as shown in figure to form the single term. As you can see X and X' get canceled and only Y remains

F = Y



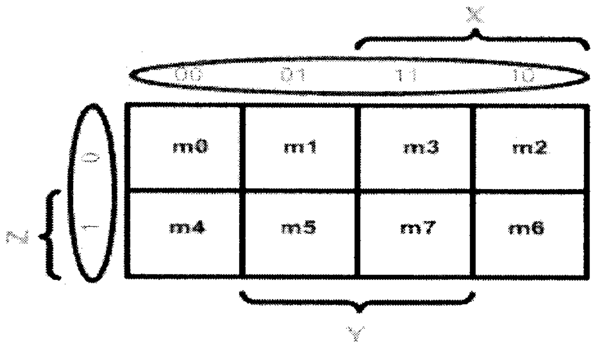
Example (2)- X'Y+XY+XY' : In this example we have the equation as input, and we have one output function. Draw the k-map for function F with marking 1 for X'Y, XY and XY position. Now combine two 1's as shown in figure to form the two single terms.

F = X + Y



3-Variable K-Map

There are 8 minterms for 3 variables (X, Y, Z). Therefore, there are 8 cells in a 3-variable K-map. One important thing to note is that K-maps follow the gray code sequence, not the binary one. Each cell in a 3-variable K-map has 3 adjacent neighbours. In general, each cell in an n-variable K-map has n adjacent neighbours.



There is wrap-around in the K-map

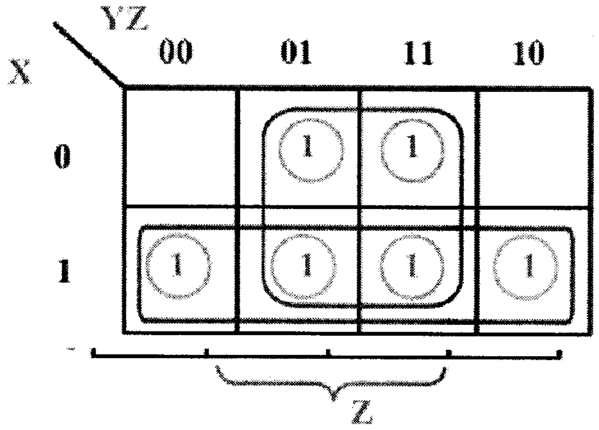
- X'Y'Z' (m0) is adjacent to X'YZ' (m2)
- XY'Z' (m4) is adjacent to XYZ' (m6)

Example (3) F = XYZ'+XYZ+X'YZ

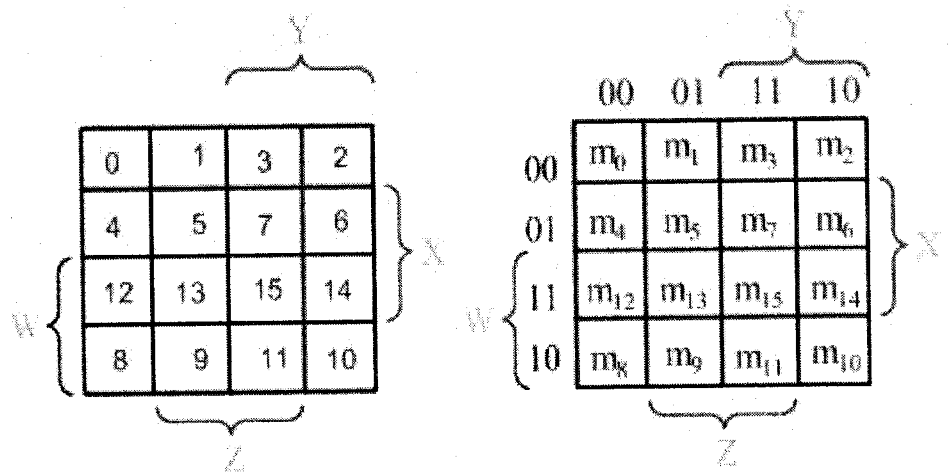
F = XY + YZ

Example (4) F(X,Y,Z) = ■(1,3,4,5,6,7)

F = X + Z



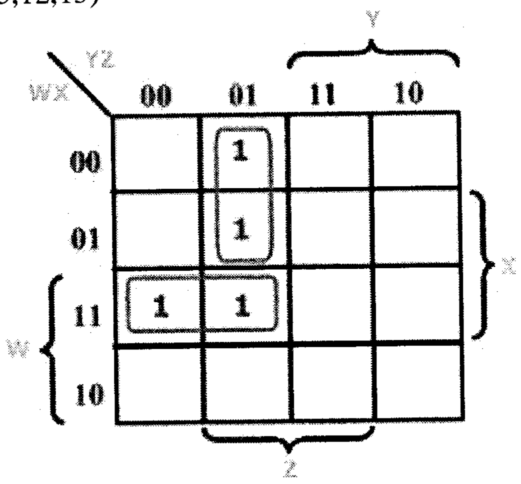
4-Variable K-Map: There are 16 cells in a 4-variable (W, X, Y, Z); K-map as shown in the figure below



0	1	3	2
4	5	7	6
12	13	15	14
8	9	11	10

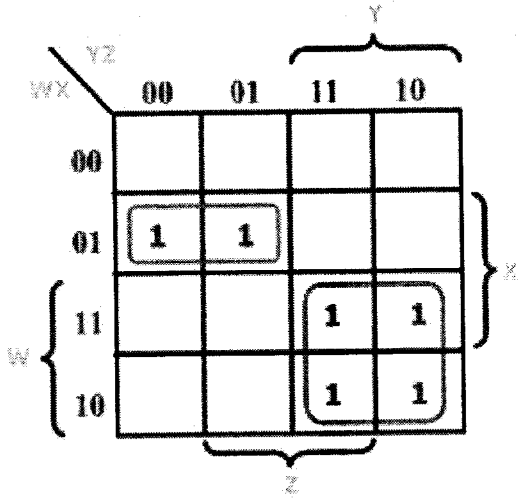
	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Example (5) F(W,X,Y,Z) = (1,5,12,13)



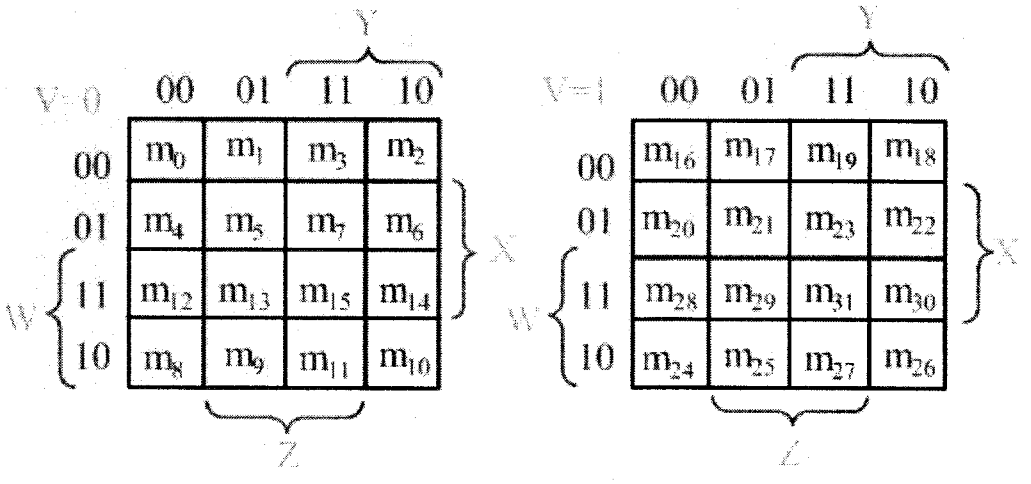
	00	01	11	10
00		1		
01		1		
11	1	1		
10				

Example (6) F(W,X,Y,Z) = (4, 5, 10, 11, 14,15)



	00	01	11	10
00				
01	1	1		
11			1	1
10			1	1

5-Variable K-Map: There are 32 cells in a 5-variable (V, W, X, Y, Z); K-map as shown in the figure below.



V=0	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

V=1	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

1.7 QUINE- MCCLUSKEY METHOD

The tabular method which is also known as the Quine-McCluskey method is particularly useful when minimising functions having a large number of variables, e.g. The six-variable functions. Computer programs have been developed employing this algorithm. The method reduces a function in standard sum of products form to a set of prime implicants from which as many variables are eliminated as possible. These prime implicants are then examined to see if some are redundant.

The tabular method makes repeated use of the law A + ■= 1. Note that Binary notation is used for the function, although decimal notation is also used for the functions. As usual a variable in true form is denoted by 1, in inverted form by 0, and the abscence of a variable by a dash (-).

Rules of Tabular Method

1. The Boolean expression to be simplified is expanded if it is not in expanded form.

2. Different terms in the expression are divided into groups depending upon the number of 1s they have.

3. The terms of the first group are successively matched with those in the next adjacent higher order group to look for any possible matching and consequent reduction. The terms are considered matched when all literals except for one match. The pairs of matched terms are replaced with a single term where the position of the unmatched literals is replaced with a dash (—). These new terms* formed as a result of the matching process find a place in the second table. The terms in the first table that do not find a match are called the prime implicants and are marked with an asterisk ( ). The matched terms are ticked (_).

4. Terms in the second group are compared with those in the third group to look for a possible match.

Again, terms in the second group that do not find a match become the prime implicants.

5. The process continues until we reach the last group. This completes the first round of matching. The terms resulting from the matching in the first round are recorded in the second table.

6. The next step is to perform matching operations in the second table. While comparing the terms for a match, it is important that a dash (—) is also treated like any other literal, that is, the dash signs also need to match. The process continues on to the third table, the fourth tables and so on until the terms become irreducible any further.

7. An optimum selection of prime implicants to account for all the original terms constitutes the terms for the minimized expression. Although optional (also called _do ‘t care‘) ter s are considered for matching, they do not have to be accounted for once prime implicants have been identified.

Example 1: Let us consider an example. Consider the following sum-of-products expression:

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

In the first step, we write the expanded version of the given expression. It can be written as follows:

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

The formation of groups, the placement of terms in different groups and the first-round matching are shown as follows:

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

A	B	C	D	
0	0	0	1	✓
0	1	0	0	✓
0	0	1	1	✓
0	1	0	1	✓
0	1	1	0	✓
1	0	0	1	✓
1	1	0	0	✓
0	1	1	1	✓
1	1	0	1	✓

A	B	C	D	
0	0	–	1	✓
0	–	0	1	✓
–	0	0	1	✓
0	1	0	–	✓
0	1	–	0	✓
–	1	0	0	✓
0	–	1	1	✓
0	1	–	1	✓
–	1	0	1	✓
0	1	1	–	✓
1	–	0	1	✓
1	1	0	–	✓

The second round of matching begins with the table shown on the previous page. Each term in the first group is compared with every term in the second group. For instance, the first term in the first group **00–1 matches with the second term in the second group 01–1 to yield 0– –1,** which is recorded in the table shown below. The process continues until all terms have been compared for a possible match. Since this new table has only one group, the terms contained therein are all prime implicants.

In the present example, the terms in the first and second tables have all found a match. But that is not always the case.

A	B	C	D	
0	–	–	1	*
–	–	0	1	*
0	1	–	–	*
–	1	0	–	*

The next table is what is known as the prime implicant table. The prime implicant table contains all the original terms in different columns and all the prime implicants recorded in different rows as shown below:

0001	0011	0100	0101	0110	0111	1001	1100	1101		
✓	✓		✓		✓				0– –1	$P \rightarrow \overline{A}.D$
✓			✓			✓		✓	– –01	$Q \rightarrow \overline{C}.D$
		✓	✓	✓	✓				01– –	$R \rightarrow \overline{A}.B$
		✓	✓				✓	✓	-10–	$S \rightarrow B.\overline{C}$

Each prime implicant is identified by a letter. Each prime implicant is then examined one by one and the terms it can account for are ticked as shown. The next step is to write a product-of-sums expression using the prime implicants to account for all the terms. In the present illustration, it is given as follows.

$$(P+Q).(P).(R+S).(P+Q+R+S).(R).(P+R).(Q).(S).(Q+S)$$

Obvious simplification reduces this expression to PQRS which can be interpreted to mean that all prime implicants, that is, P, Q, R and S, are needed to account for all the original terms.

Therefore, the minimized expression = $\overline{A}.D+\overline{C}.D+\overline{A}.B+B.\overline{C}$.

Example 2: $(\overline{A}+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D).(\overline{A}+\overline{B}+C+\overline{D}).(A+\overline{B}+\overline{C}+\overline{D}).(A+\overline{B}+C+\overline{D})$

The procedure is similar to that described for the case of simplification of sum-of-products expressions. The resulting tables leading to identification of prime implicants are as follows:

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

A	B	C	D	
0	1	0	1	✓
0	1	1	1	✓
1	1	0	1	✓
1	1	1	0	✓
1	1	1	1	✓

A	B	C	D	
0	1	–	1	✓
–	1	0	1	✓
–	1	1	1	✓
1	1	–	1	✓
1	1	1	–	*

A	B	C	D	
–	1	–	1	*

The prime implicant table is constructed after all prime implicants have been identified to look for the optimum set of prime implicants needed to account for all the original terms. The prime implicant table shows that both the prime implicants are the essential ones:

0101	0111	1101	1110	1111	Prime implicants
			✓	✓	111–
✓	✓	✓		✓	–1–1

The minimized expression = $(\overline{A}+\overline{B}+\overline{C}).(\overline{B}+\overline{D})$.

Example 3: Consider the function f(A, B, C, D) = [redacted](0,1,2,3,5,7,8,10,12,13,15), note that this is in decimal form.

[redacted](0000,0001,0010,0011,0101,0111,1000,1010,1100,1101,1111) in binary form.
(0,1,1,2,2,3,1,2,2,3,4) in the index form.

The prime implicants are: ¯+¯+¯+ +¯+ ¯

The chart is used to remove redundant prime implicants. A grid is prepared having all the prime implicants listed at the left and all the minterms of the function along the top. Each minterm covered by a given prime implicant is marked in the appropriate position.

From the above chart, BD is an essential prime implicant. It is the only prime implicant that covers the minterm decimal 15 and it also includes 5, 7 and 13. [redacted]so an essential prime implicant. It is the only prime implicant that covers the minterm denoted by decimal 10 and it also includes the terms 0, 2 and 8. The other minterms of the function are 1, 3 and 12. Minterm 1 is present in [redacted] and [redacted]D. Similarly for minterm 3, We can therefore use either of these prime implicants for these minterms. Minterm 12 is present in A and AB , so again either can be used.

Thus, one minimal solution is: =¯ + + ¯+

UNIT II Arithematic circuits Half adder – Truth table and circuit – Full adder – Truth table and circuit – Four bit adder – Half subtractor – Full subtractor – Multiplexer: Four input multiplexer – Applications of Multiplexer – demultiplexer – Decoders 2 to 4 decoder – BCD to seven segment decoder – encoders.

Arithmetic circuits are the ones which perform arithmetic operations like addition, subtraction, multiplication, division, parity calculation. Most of the time, designing these circuits is the same as designing mux, encoders and decoders.

1. Adders

Adders are the basic building blocks of all arithmetic circuits; adders add two binary numbers and give out sum and carry as output. Basically we have two types of adders.

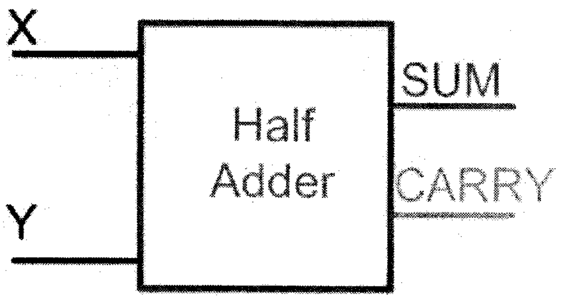
- Half Adder.
- Full Adder.

✓ Half Adder

A half-adder is an arithmetic circuit block that can be used to add two bits. Such a circuit thus has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.

Adding two single-bit binary values X, Y produces a sum S bit and a carry out C-out bit. This operation is called half addition and thus the circuit to realize it is called a half adder.

Symbol



Truth table

X	Y	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

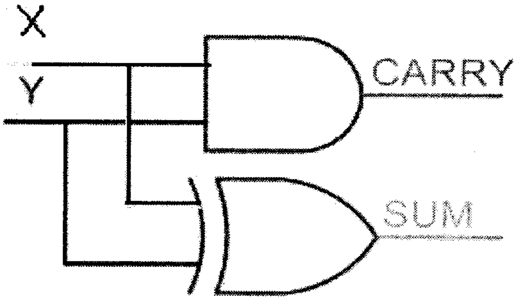
The expression for the sum and carry are,

■

Sum = XY+XY

Carry = XY

Circuit



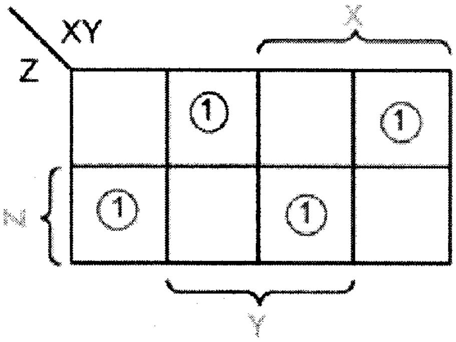
✓ Full Adder

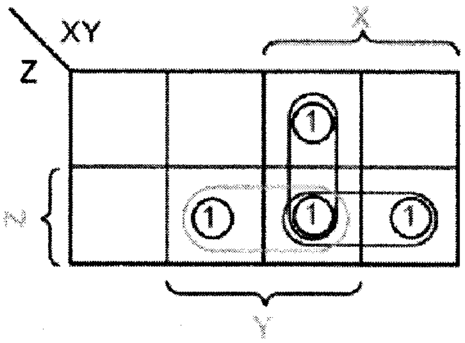
A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY output. Such a building block becomes a necessity when it comes to adding binary numbers with a large number of bits. The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only.

Full adder takes a three-bits input. Adding two single-bit binary values X, Y with a carry input bit C-in produces a sum bit S and a carry out C.

Truth Table

X	Y	Z	SUM	CARRY
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



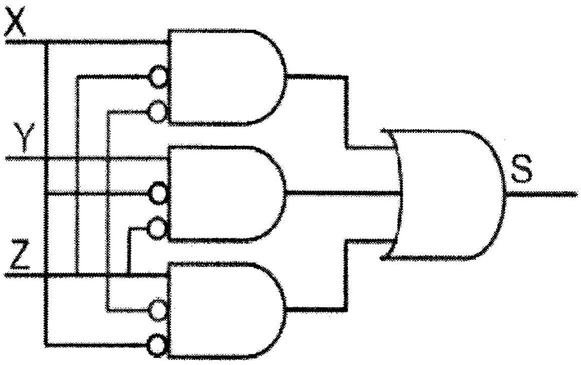


=

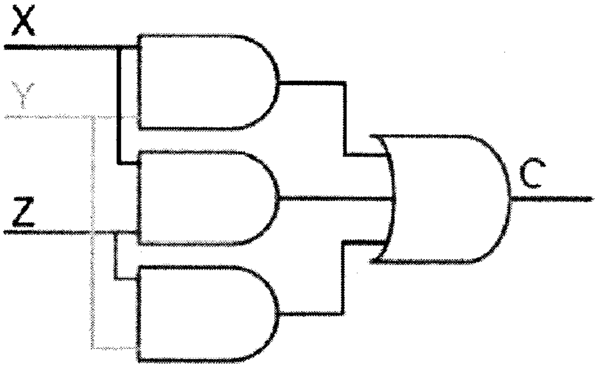
Full Adder using AND-OR

The below implementation shows implementing the full adder with AND-OR gates, instead of using XOR gates. The basis of the circuit below is from the above K-map

Circuit-SUM

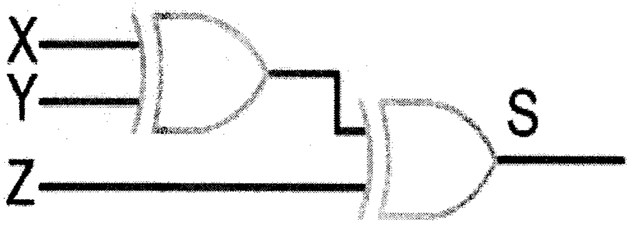


Circuit-CARRY

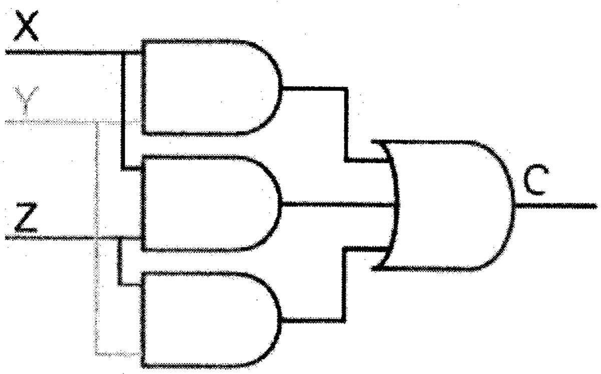


Full Adder using AND-OR

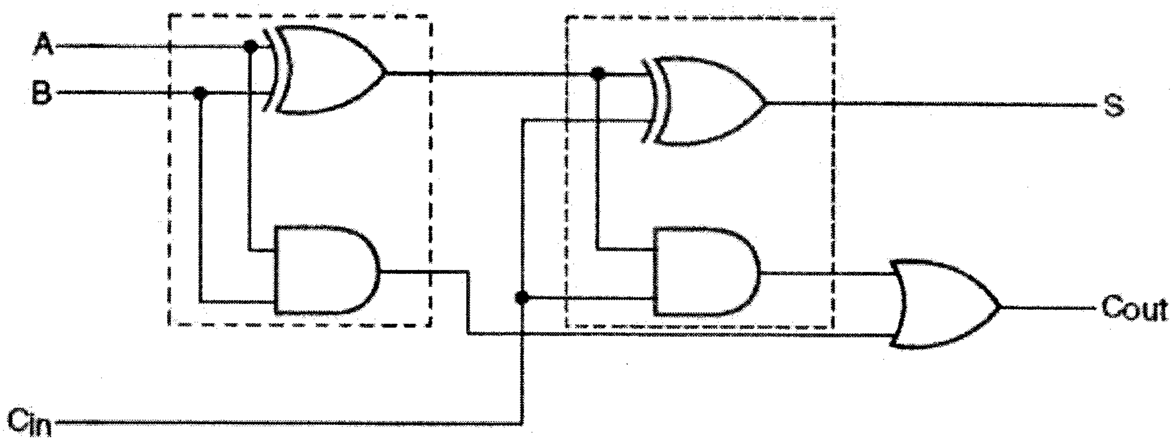
Circuit-SUM

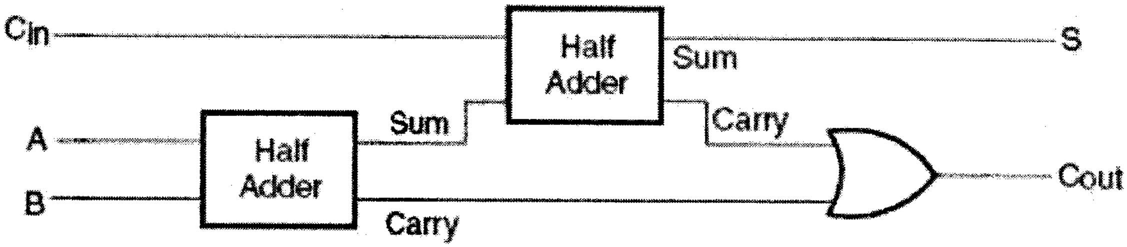


Circuit-CARRY



Logic Implementation of a full adder with Half Adders





- ✓ **n-bit Carry Ripple Adder**

An n-bit adder used to add two n-bit binary numbers can be built by connecting n full adders in series. Each full adder represents a bit position j (from 0 to n-1).

Each carry out C-out from a full adder at position j is connected to the carry in C-in of the full adder at higher position j+1. The output of a full adder at position j is given by:

Sj= Xj Yj Cj
Cj+1 = Xj . Yj + Xj . Cj + Y . Cj

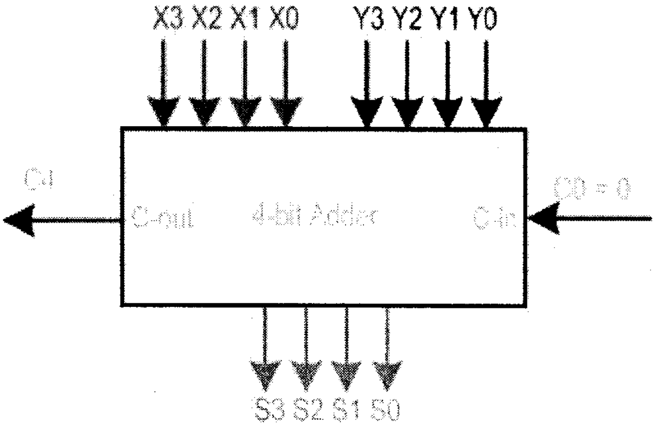
In the expression of the sum Cj must be generated by the full adder at lower position j. The propagation delay in each full adder to produce the carry is equal to two gate delays = 2 D Since the generation of the sum requires the propagation of the carry from the lowest position to the highest position , the total propagation delay of the adder is approximately:

Total Propagation delay = 2 nD

4-bit Carry Ripple Adder

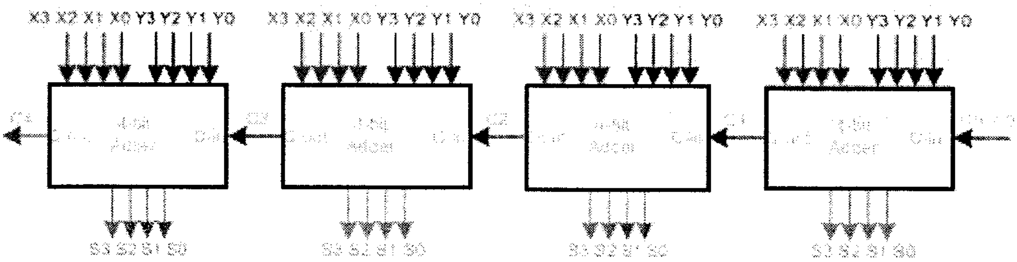
Adds two 4-bit numbers:

X = X3 X2 X1 X0
Y = Y3 Y2 Y1 Y0
Producing the sum S = S3 S2 S1 S0, C-out = C4 from the most significant position
j=3 Total Propagation delay = 2 nD = 8D or 8 gate delays



Larger Adder

Example: 16-bit adder using 4 4-bit adders. Adds two 16-bit inputs X (bits X0 to X15), Y (bits Y0 to Y15) producing a 16-bit Sum S (bits S0 to S15) and a carry out C16 from the most significant position. Propagation delay for 16-bit adder = 4 x propagation delay of 4-bit adder
= 4 x 2 nD = 4 x 8D = 32 D or 32 gate delays



✓ **Carry Look-Ahead Adder**

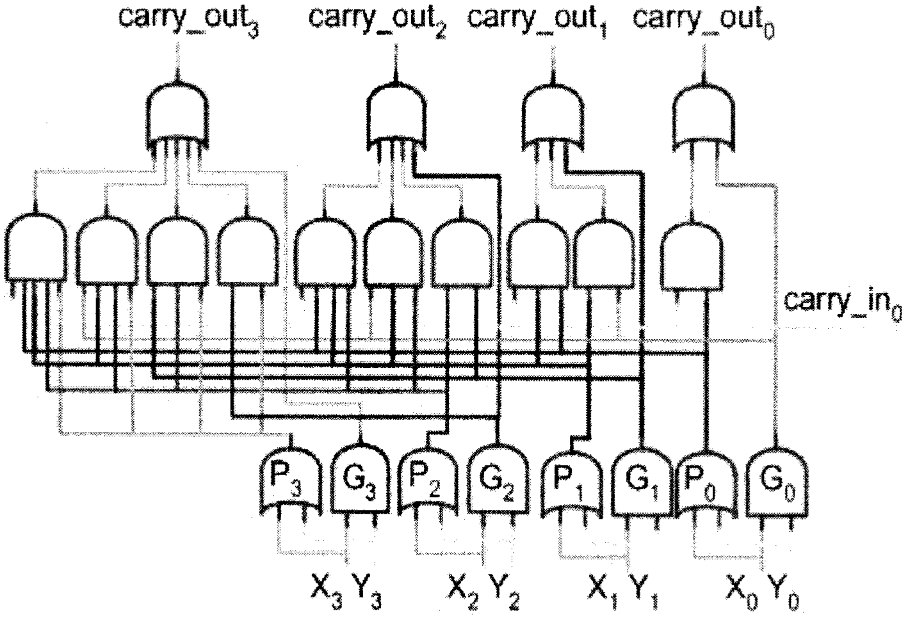
The delay generated by an N-bit adder is proportional to the length N of the two numbers X and Y that are added because the carry signals have to propagate from one full-adder to the next. For large values of N, the delay becomes unacceptably large so that a special solution needs to be adopted to accelerate the calculation of the carry bits. This solution involves a "look-ahead carry generator" which is a block that simultaneously calculates all the carry bits involved. Once these bits are available to the rest of the circuit, each individual three-bit addition (Xi+Yi+carry-ini) is implemented by a simple 3-input XOR gate. The design of the look-ahead carry generator involves two Boolean functions named Generate and Propagate. For each input bits pair these functions are defined as: Gi = Xi . Yi & Pi = Xi + Yi

The carry bit c-out(i) generated when adding two bits Xi and Yi is '1' if the corresponding function Gi is '1' or if the c-out(i-1)='1' and the function Pi = '1' simultaneously. In the first case, the carry bit is activated by the local conditions (the values of Xi and Yi). In the second, the carry bit is received from the less significant elementary addition and is propagated further to the more significant elementary addition. Therefore, the carry_out bit corresponding to a pair of bits Xi and Yi is calculated according to the equation:
carry_out(i) = Gi + Pi.carry_in(i-1)

For a four-bit adder the carry-outs are calculated as follows

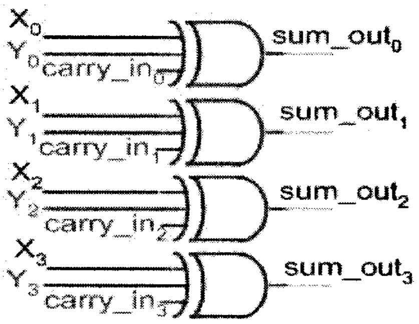
carry_out0 = G0 + P0 . carry_in0
carry_out1 = G1 + P1 . carry_out0 = G1 + P1G0 + P1P0 . carry_in0
carry_out2 = G2 + P2G1 + P2P1G0 + P2P1P0 . carry_in0
carry_out3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1 . carry_in0

The set of equations above are implemented by the circuit below and a complete adder with a look-ahead carry generator is next. The input signals need to propagate through a maximum of 4 logic gate in such an adder as opposed to 8 and 12 logic gates in its counterparts illustrated earlier.



Sums can be calculated from the following equations, where carry_out is taken from the carry calculated in the above circuit.

sum_out0 = X 0 Y0
carry_out0 sum_out1 = X 1 Y1
carry_out1 sum_out2 = X 2 Y2
carry_out2 sum_out3 = X 3 Y3
carry_out3



✓ **BCD Adder**

BCD addition is the same as binary addition with a bit of variation: whenever a sum is greater than 1001, it is not a valid BCD number, so we add 0110 to it, to do the correction. This will produce a carry, which is added to the next BCD position.

- Add the two 4-bit BCD code inputs.
- Determine if the sum of this addition is greater than 1001; if yes, then add 0110 to this sum and generate a carry to the next decimal position

2. Subtractor

Subtractor circuits take two binary numbers as input and subtract one binary number input from the other

binary number input. Similar to adders, it gives out two outputs, difference and borrow (carry-in the case of Adder). **The BORROW output here specifies whether a _1‘ has been borrowed to perform** the

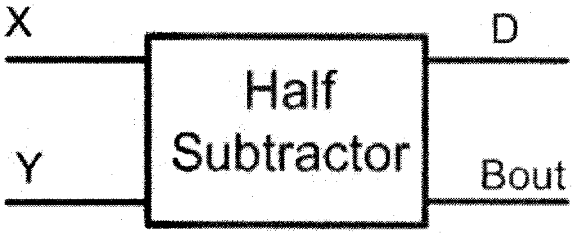
subtraction.

There are two types of subtractors,

✓ **Half Subtractor**

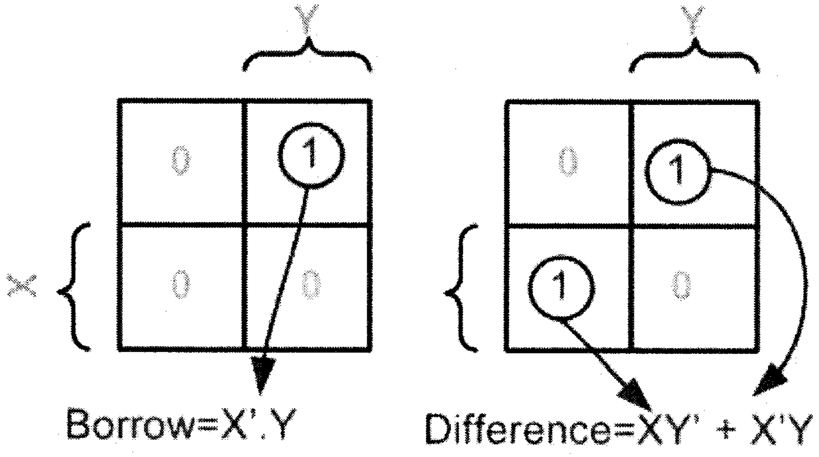
The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B (borrow). The logic symbol and truth table are shown below.

Symbol **Truth table**

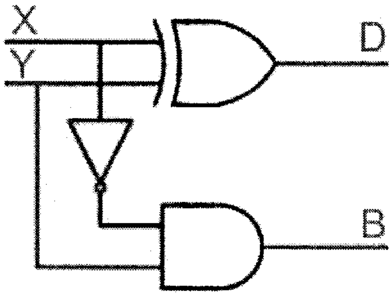


X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the above table we can draw the K-map as shown below for "difference" and "borrow". The Boolean expression for the difference and Borrow can be written.



From the equation we can draw the half-subtractor as shown in the figure below.



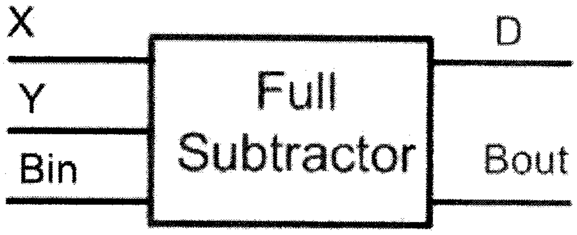
✓ Full Subtractor

A full subtractor is a combinational circuit that performs subtraction involving three bits, namely

minuend, subtrahend, and borrow-in. 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‘**

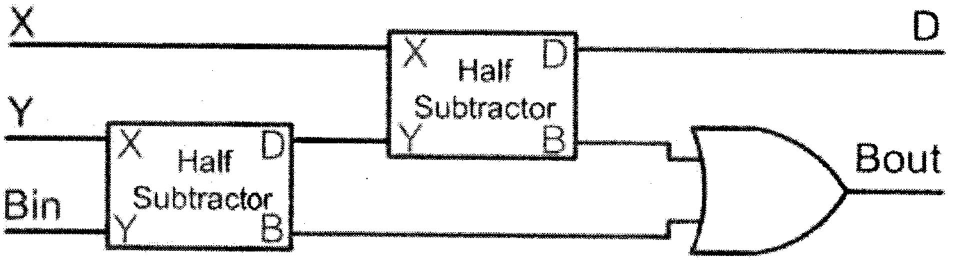
from the next possible higher minuend bit. The logic symbol and truth table are shown below.

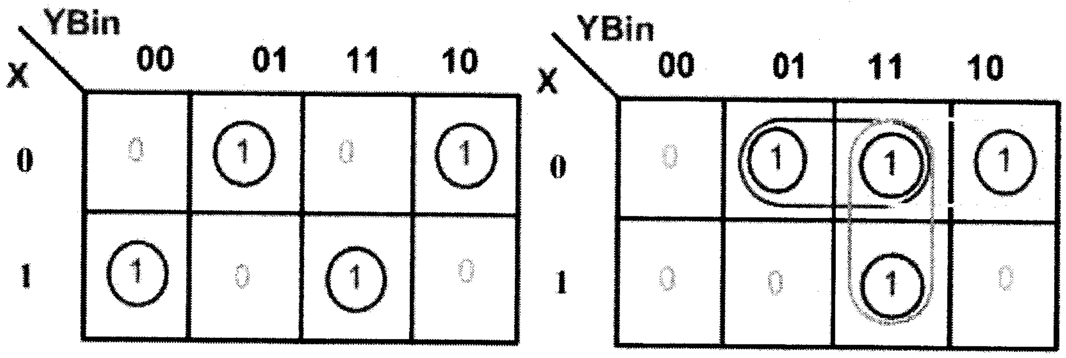
Symbol



Truth table

X	Y	Bin	D	Bout
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





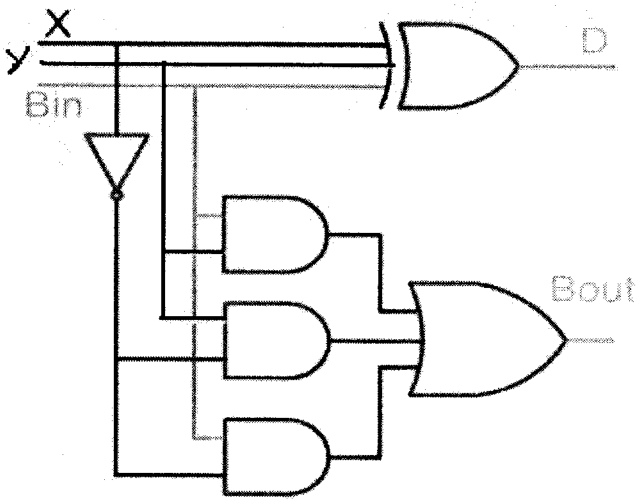
X \ YBin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

X \ YBin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Difference=X'Y'Bin + X'YBin' + XY'Bin' + XYBin

Borrow=X'Bin + X'Y + YBin

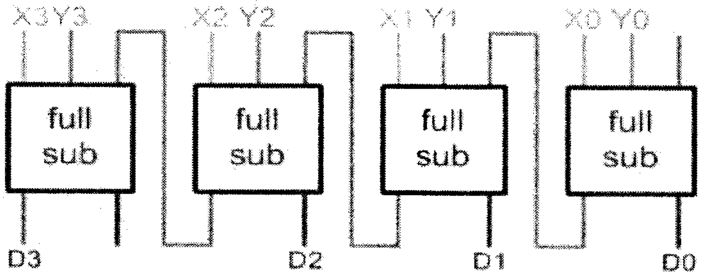
From the above expression, we can draw the circuit below. If you look carefully, you will see that a full-subtractor circuit is more or less same as a full-adder with slight modification.



✓ Parallel Binary Subtractor

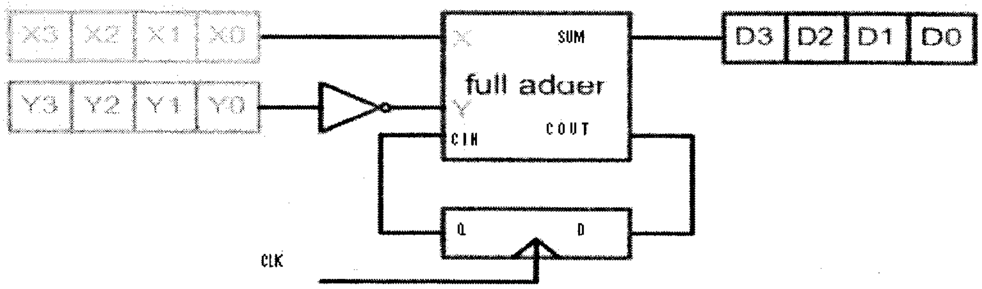
Parallel binary subtractor can be implemented by cascading several full-subtractors. Implementation and associated problems are those of a parallel binary adder, seen before in parallel binary adder section.

Below is the block level representation of a 4-bit parallel binary subtractor, which subtracts 4-bit Y3Y2Y1Y0 from 4-bit X3X2X1X0. It has 4-bit difference output D3D2D1D0 with borrow output Bout.



✓ Serial Binary Subtracter

A serial subtracter can be obtained by converting the serial adder using the 2's complement system. The subtrahend is stored in the Y register and must be 2's complemented before it is added to the minuend stored in the X register. The circuit for a 4-bit serial subtracter using full-adder is shown in the figure below.



✓ Comparators

It is a combinational circuit that compares two numbers and determine their relative magnitude. The output of comparator is usually 3 binary variables indicating:

A<B, A=B, A>B

1-bit comparator: Let's begin with 1 bit comparator and from the name we can easily make out that this circuit would be used to compare 1 bit binary numbers.

A	B	A>B	A=B	A<B
0	0	0	1	0
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0

For a 2-bit comparator we have four inputs A1A0 and B1B0 and three output E (is 1 if two numbers are equal) G (is 1 when A > B) and L (is 1 when A < B) If we use truth table and K-map the result is

A>B

A \ B	0	1
0	0	0
1	1	0

Equation is $A>B = A.\overline{B}$

A<B

A \ B	0	1
0	0	1
1	0	0

Equation is $A<B = \overline{A}.B$

(A=B)

A \ B	0	1
0	1	0
1	0	1

The equation isf$(A=B) = \overline{A}.\overline{B} + A.B$
= A XNOR B

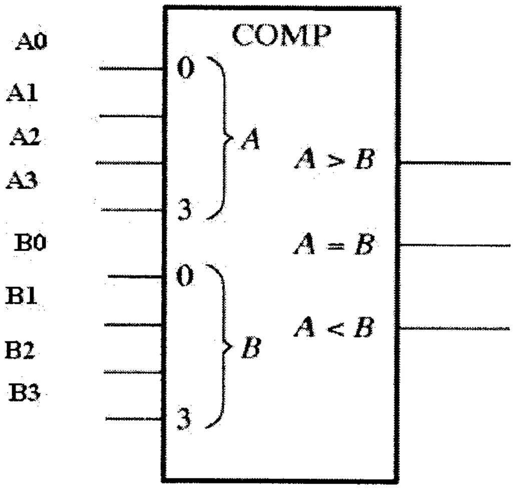
The comparison process of two positive numbers X and Y is performed in a bit-by-bit manner starting with the most significant bit:

If the most significant bits are Xn='1' and Yn='0' then number X is larger than Y.

- If Xn='0' and Yn='1' then number X is smaller than Y.
- If Xn=Yn then no decision can be taken about X and Y based only on these two bits.

If the most significant bits are equal then the result of the comparison is determined by the less significant bits Xn-1 and Yn-1. If these bits are equal as well, the process continues with the next pair of bits. If all bits are equal then the two numbers are equal.

4-bit comparator:

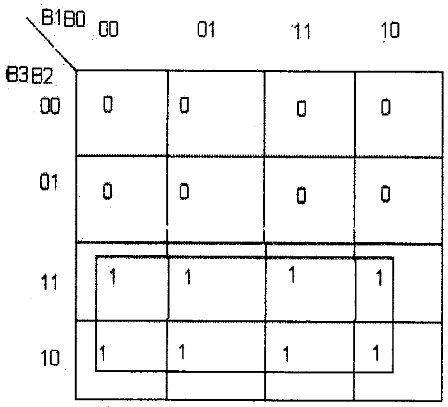


2.5 CODE CONVERSION- Binary to Gray converter

Truth Table

S. No	B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

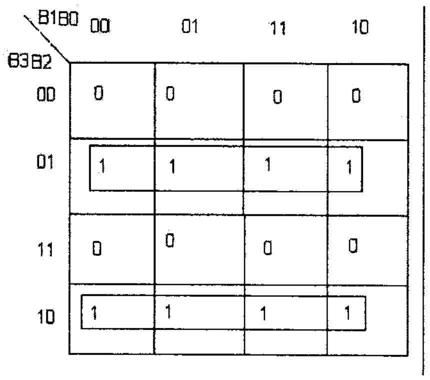
K-MAP FOR G3:



B3B2 \ B1B0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

G3= B3

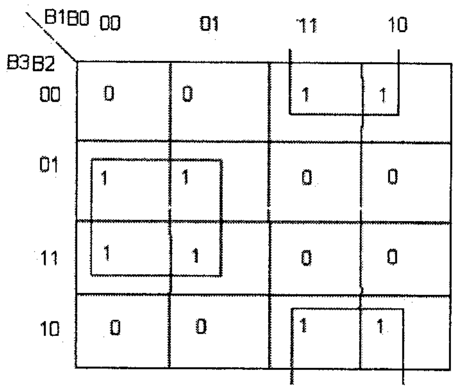
K-MAP FOR G2:



B3B2 \ B1B0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

G2= **B3’** B2 + B3 **B2’**= B3 ■ B2

K-MAP FOR G1:



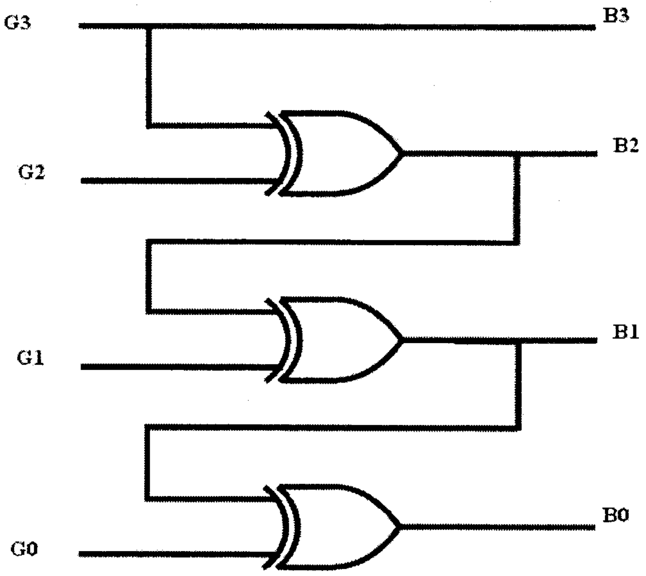
B3B2 \ B1B0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

G1= **B1’** B2 + B1 **B2’**= B1 ■ B2

K-MAP FOR G0

B3B2 \ B1B0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

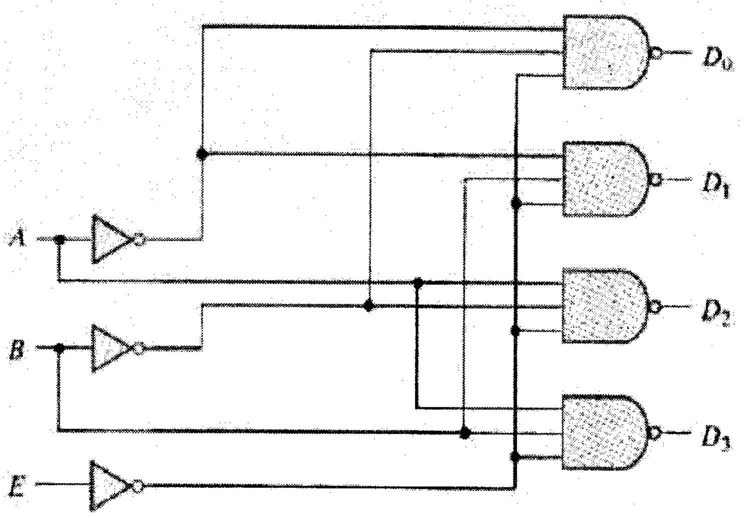
G0= B1' B0 + B1 B0'= B1 ■ B0



2.6 DECODERS

A **decoder** circuit can be used to implement AND-OR circuit SOP Boolean expression when decoder active state output is 1 and inactive 0 .

- **Number of binary inputs = n**
- **Number of binary outputs = 2n = Maximum n**umber of minterms, where n is the number of literals in F
- **Its outputs reflect the Mini**-terms with one term each at each of the output



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Figure: 2-to-4 line decoder with enable input

Truth Table of a Three-to-Eight-Line Decoder

Inputs			**Outputs**							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

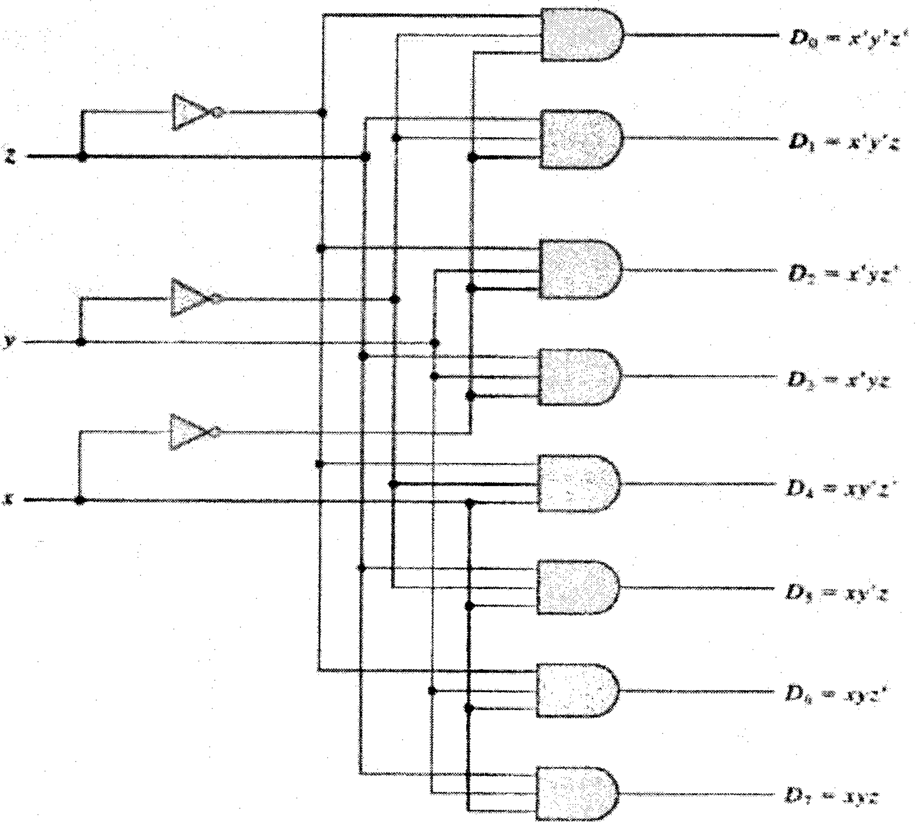


Fig: Circuit for 3-to-8 line decoder

2.7 ENCODERS

An **encoder** is a circuit that converts the binary information from one form to another. Gives a unique combination of outputs according to the information at a unique input at one-line (or at multiple lines).

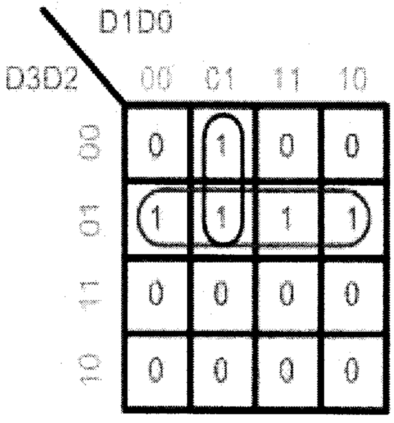
Action of a one active line input encoder is opposite of that of a one active line output decoder. An encoder, which has multi-**lines as the active inputs, is also called 'priority encoder'. Encoder can be** differentiated from decoder by greater number of inputs than outputs compared to the decoder. The priority encoder includes a priority function.

4to3 Priority Encoder-The truth table of a 4-input priority encoder is as shown below. The input D3 has the highest priority, D2 has next highest priority, D0 has the lowest priority. This means output Y2 and Y1 are 0 only when none of the inputs D1, D2, D3 are high and only D0 is high. A 4 to 3 encoder consists of four inputs and three outputs, truth table and symbols of which is shown below.

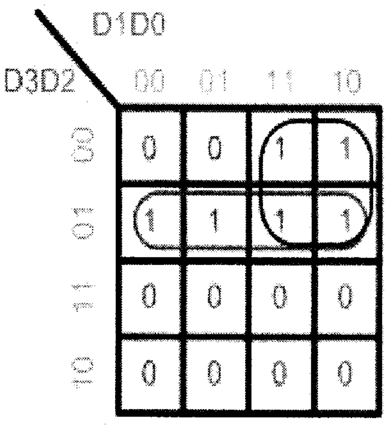
Truth Table

D3	D2	D1	D0	Y2	Y1	Y0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	0
0	1	x	x	0	1	1
1	x	x	x	1	0	0

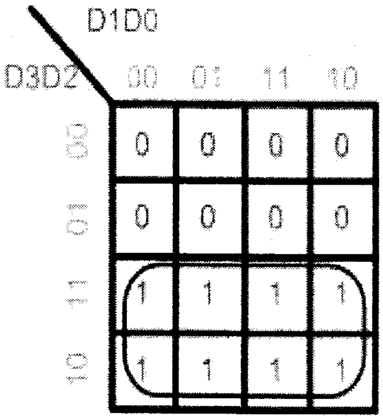
K-map



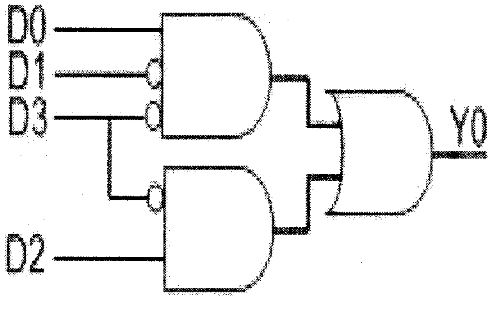
Y0=D3'.D2 + D3'.D0.D1'

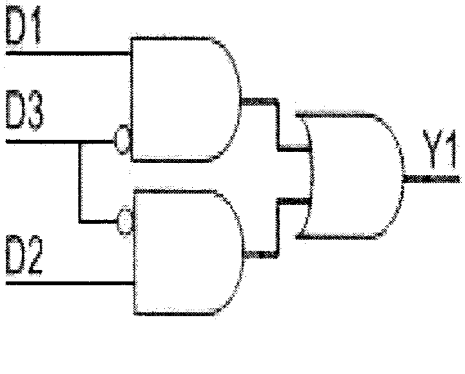


Y1=D3'.D2 + D3'.D1



Y2=D3

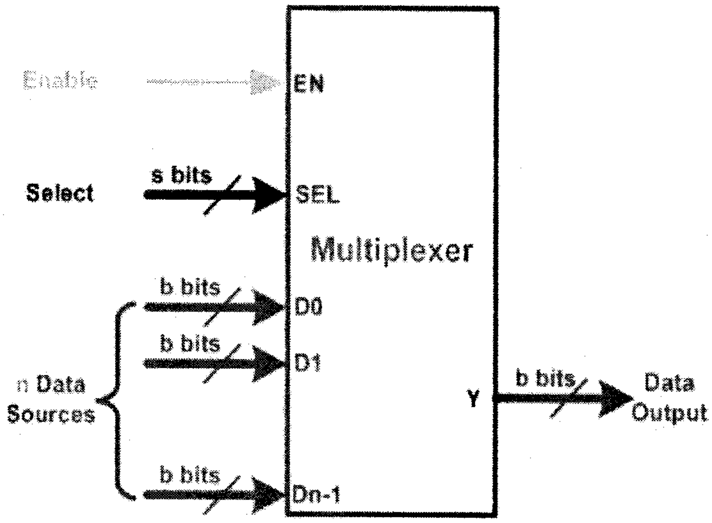




2.8 MULTIPLEXERS

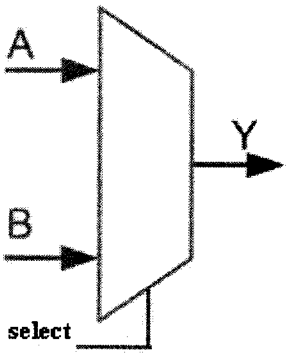
Many tasks in communications, control, and computer systems can be performed by combinational logic circuits. When a circuit has been designed to perform some task in one application, it often finds use in a different application as well.

A **multiplexer (MUX)** is a digital switch which connects data from one of n sources to the output. A number of select inputs determine which data source is connected to the output. The block diagram of MUX with n data sources of b bits wide and s bits wide select line is shown in below figure.



Example - 2x1 MUX

A 2 to 1 line multiplexer is shown in figure below, each 2 input lines A to B is applied to one input of an AND gate. Selection lines S are decoded to select a particular AND gate. The truth table for the 2:1 mux is given in the table below.



Truth table

S	Y
0	A
1	B

Design of a 2:1 Mux

To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.

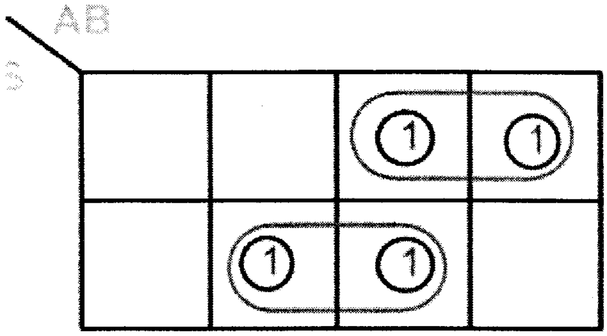
Combiningthe two 1' as shown in figure, we can drive the output y as shown below

$$Y = A.S' + B.S$$

Truth table

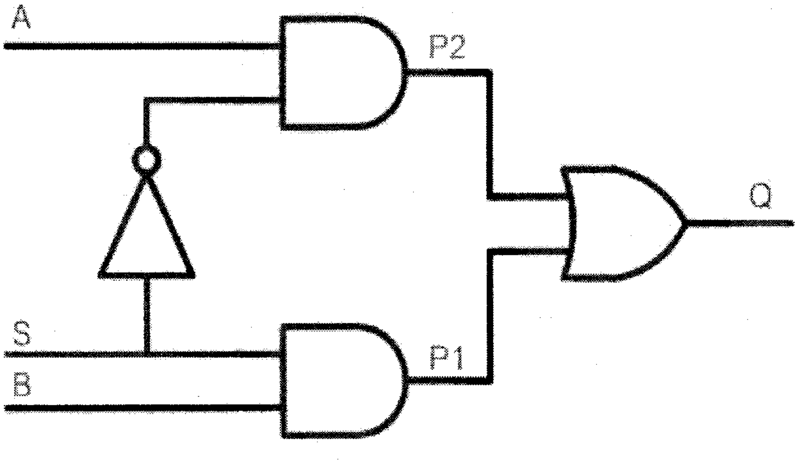
B	A	S	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

K-map



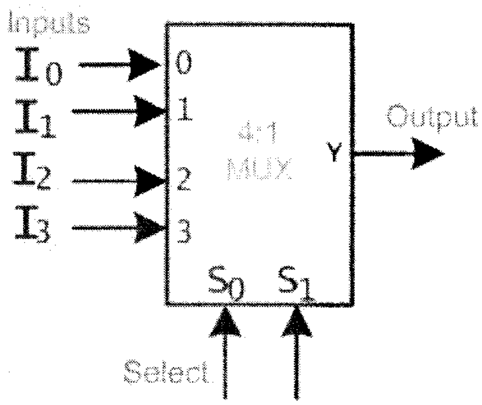
		1	1
	1	1	

Circuit



Example : 4:1 MUX

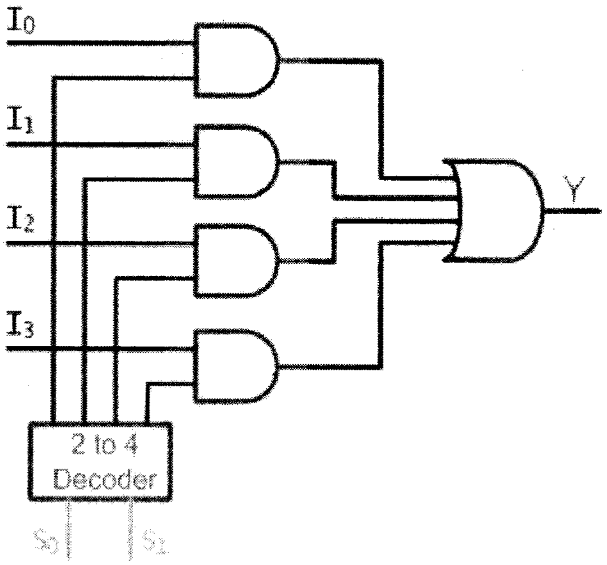
A 4 to 1 line multiplexer is shown in figure below, each of 4 input lines I0 to I3 is applied to one input of an AND gate. Selection lines S0 and S1 are decoded to select a particular AND gate. The truth table for the 4:1 mux is given in the table below.



Truth table

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

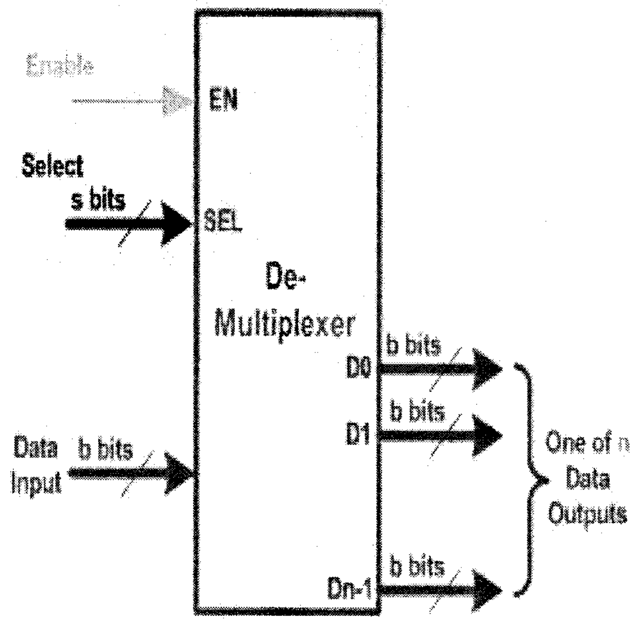
Circuit



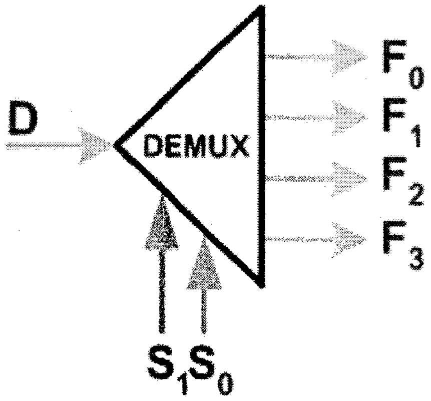
2.9 DEMULTIPLEXERS

They are digital switches which connect data from one input source to one of n outputs. Usually implemented by using n-to-2n binary decoders where the decoder enable line is used for data input of the de-multiplexer.

The figure below shows a de-multiplexer block diagram which has got s-bits-wide select input, one b-bits-wide data input and n b-bits-wide outputs.



Example: 1-to-4 De-multiplexer

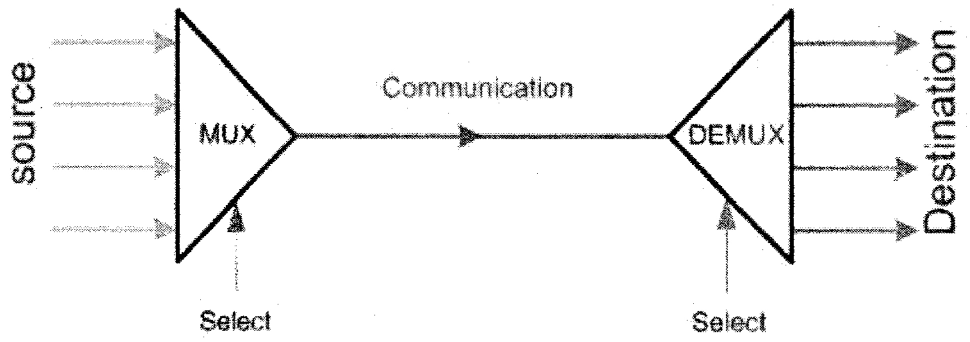


Truth table

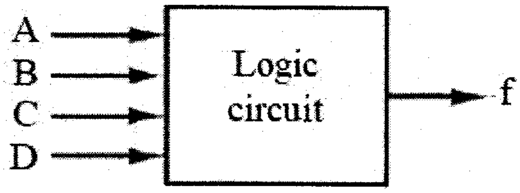
S1	S0	F0	F1	F2	F3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Mux- Demux: Application Example

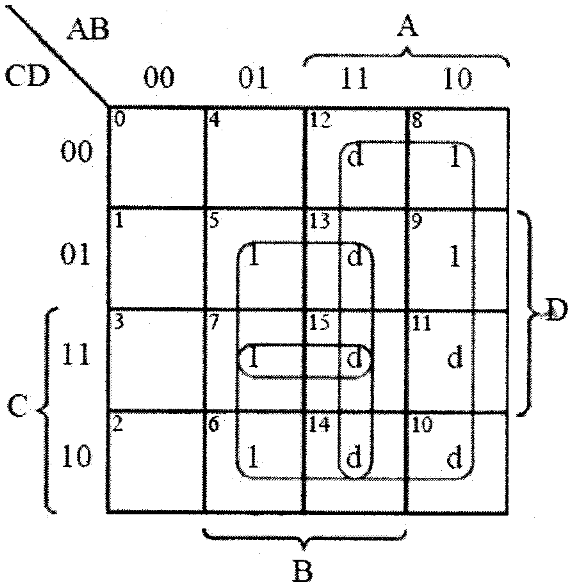
This enables sharing a single communication line among a number of devices. At any time, only one source and one destination can use the communication line.



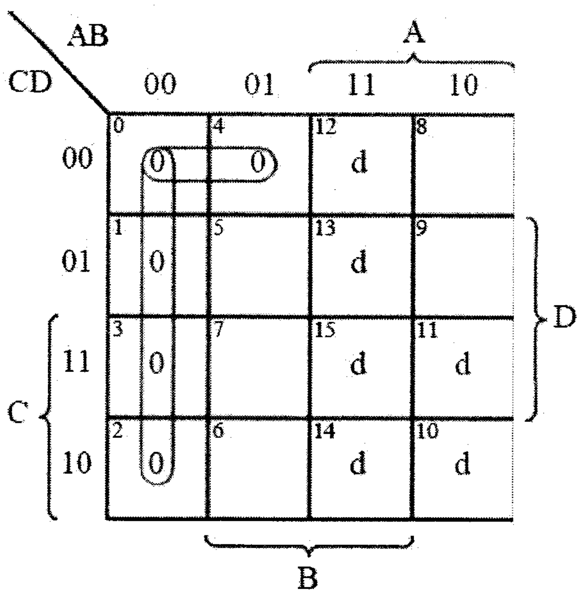
Example: Design a circuit to distinguish BCD digits ≥ 5 from those < 5.



ABCD	Minterm	f(A, B, C, D)
0000	0	0
0001	1	0
0010	2	0
0011	3	0
0100	4	0
0101	5	1
0110	6	1
0111	7	1
1000	8	1
1001	9	1
1010	10	d
1011	11	d
1100	12	d
1101	13	d
1110	14	d
1111	15	d



(a) MSOP



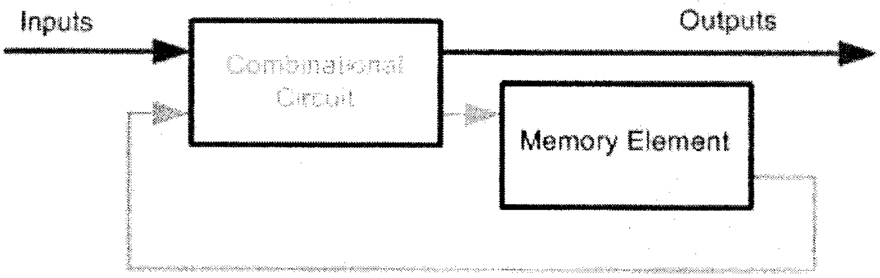
(b) MPOS

$f(A,B,C,D) = A + BD + BC;$ $f(A,B,C,D) = (A + B)(A + C + D)$

UNIT III

Flipflops Introduction – NAND LATCH, J K flipflop – J K Master – slave flipflop – D flipflop and T flipflop – Registers and Counters: Shift registers – serial in – parallelout, serial in – serial out, parallel in – serial out, parallel in – parallel out shift registers – wave forms for the above – Counters – up counters, down counters, decade counters, timing sequences, Mod – n counters.

Digital electronics is classified into combinational logic and sequential logic. Combinational logic output depends on the inputs levels, whereas sequential logic output depends on stored levels and also the present inputs.



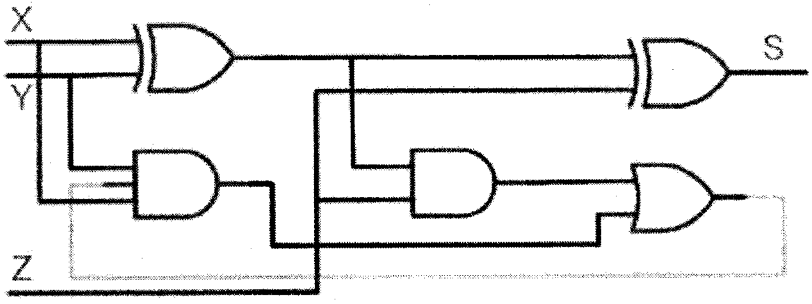
The memory elements are devices capable of storing binary info. The binary info stored in the memory elements at any given time defines the state of the sequential circuit. The input and the present state of the memory element determine the output. Memory elements next state is also a function of external inputs and present state. A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

There are two types of sequential circuits. Their classification depends on the timing of their signals:

- Synchronous sequential circuits
- Asynchronous sequential circuits

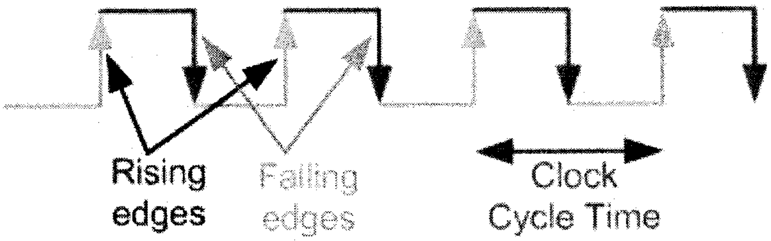
✓ ASYNCHRONOUS SEQUENTIAL CIRCUIT

This is a system whose outputs depend upon the order in which its input variables change and can be affected at any instant of time. Gate-type asynchronous systems are basically combinational circuits with feedback paths. Because of the feedback among logic gates, the system may, at times, become unstable. Consequently they are not often used.



✓ SYNCHRONOUS SEQUENTIAL CIRCUITS

This type of system uses storage elements called flip-flops that are employed to change their binary value only at discrete instants of time. Synchronous sequential circuits use logic gates and flip-flop storage devices. Sequential circuits have a clock signal as one of their inputs. All state transitions in such circuits occur only when the clock value is either 0 or 1 or happen at the rising or falling edges of the clock depending on the type of memory elements used in the circuit. Synchronization is achieved by a timing device called a clock pulse generator. Clock pulses are distributed throughout the system in such a way that the flip-flops are affected only with the arrival of the synchronization pulse. Synchronous sequential circuits that use clock pulses in the inputs are called clocked-sequential circuits. They are stable and their timing can easily be broken down into independent discrete steps, each of which is considered separately.



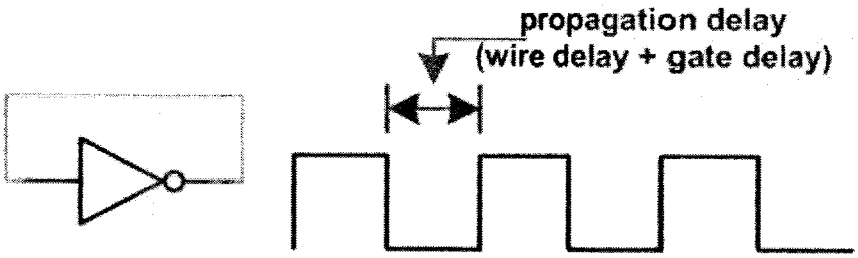
A clock signal is a periodic square wave that indefinitely switches from 0 to 1 and from 1 to 0 at fixed intervals. Clock cycle time or clock period: the time interval between two consecutive rising or falling edges of the clock.

Clock Frequency = 1 / clock cycle time (measured in cycles per second or Hz)

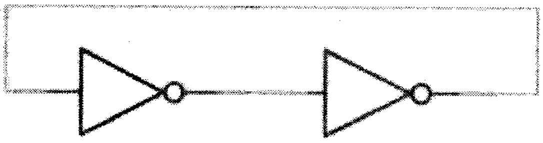
Example: Clock cycle time = 10ns clock frequency = 100M

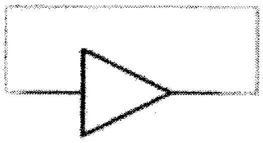
3.1 CONCEPT OF SEQUENTIAL LOGIC

A sequential circuit is a combinational logic with some feedback to maintain its current value, like a memory cell. To understand the basics let's consider the basic feedback logic circuit below, which is a simple NOT gate whose output is connected to its input. The effect is that output oscillates between HIGH and LOW (i.e. 1 and 0). Oscillation frequency depends on gate delay and wire delay. Assuming a wire delay of 0 and a gate delay of 10ns, then oscillation frequency would be (on time + off time = 20ns) 50Mhz.

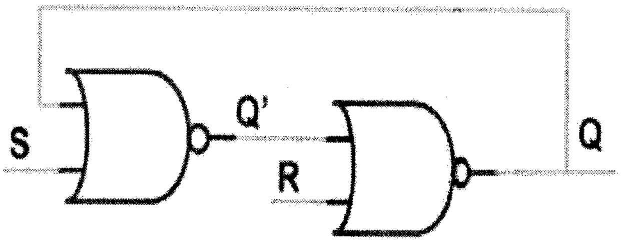


The basic idea of having the feedback is to store the value or hold the value, but in the above circuit, output keeps toggling. We can overcome this problem with the circuit below, which is basically cascading two inverters, so that the feedback is in-phase, thus avoids toggling. The equivalent circuit is the same as having a buffer with its output connected to its input.





The circuit below is the same as the inverters connected back to back with provision to set the state of each gate (NOR is gate with both inputs shorted like a inverter). I am not going to explain the operation, as it is clear from the truthtable. S is called set and R is called **Reset**.



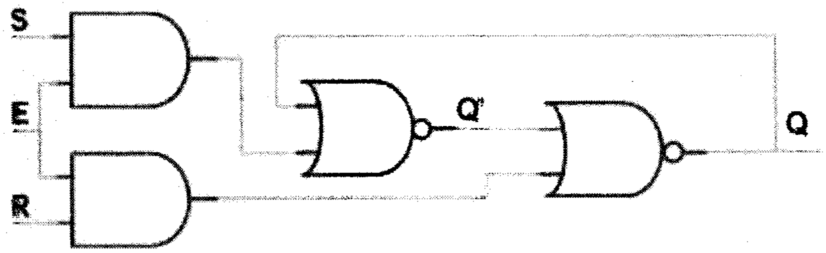
S	R	Q	Q+
0	0	0	0
0	0	1	1
0	1	X	0
1	0	X	1
1	1	X	0

There still seems to be some problem with the above configuration, we cannot control when the input should be sampled, in other words there is no enable signal to control when the input is sampled.

Normally input enable signals can be of two types.

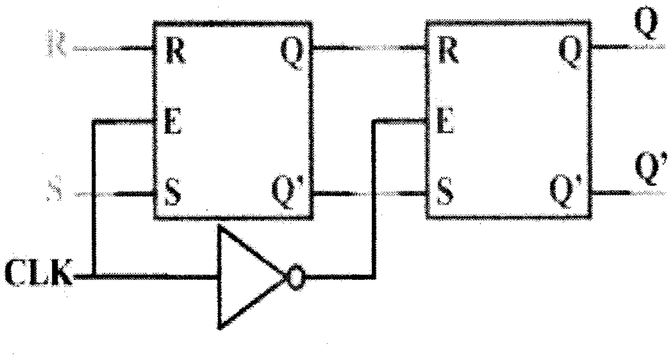
- ✓ Level Sensitive or (LATCH)
- ✓ Edge Sensitive or (Flip-Flop)

✓ **Level Sensitive:** The circuit below is a modification of the above one to have level sensitive enable input. Enable, when LOW, masks the input S and R. When HIGH, presents S and R to the sequential logic input (the above circuit two NOR Gates). Thus Enable, when HIGH, transfers input S and R to the sequential cell transparently, so this kind of sequential circuits are called **transparent Latch**. The memory element we get is an RS Latch with active high Enable.



✓ **Edge Sensitive:** The circuit below is a cascade of two level sensitive memory elements, with a phase shift in the enable input between first memory element and second memory element. The first RS latch (i.e. the first memory element) will be enabled when CLK input is HIGH and the second RS latch will be enabled when CLK is LOW. The net effect is input RS is moved to Q and

Q' when CLK changes state from HIGH to LOW, this HIGH to LOW transition is called falling edge. So the Edge Sensitive element we get is called negative edge RS flip-flop.



3.2 LATCHES AND FLIP-FLOPS

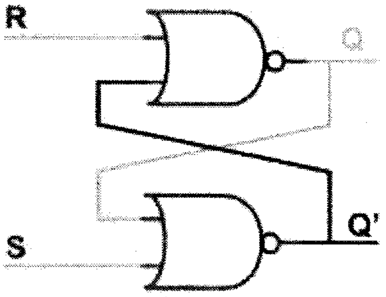
There are two types of sequential circuits.

- Asynchronous Circuits.
- Synchronous Circuits.

Latches and Flip-flops are one and the same with a slight variation: Latches have level sensitive control signal input and Flip-flops have edge sensitive control signal input. Flip-flops and latches which use this control signals are called synchronous circuits. So if they don't use clock inputs, then they are called asynchronous circuits.

✓ RS Latch

RS latch have two inputs, S and R. S is called set and R is called reset. The S input is used to produce HIGH on Q (i.e. store binary 1 in flip-flop). The R input is used to produce LOW on Q (i.e. store binary 0 in flip-flop). Q' is Q complementary output, so it always holds the opposite value of Q. The output of the S-R latch depends on current as well as previous inputs or state, and its state (value stored) can change as soon as its inputs change. The circuit and the truth table of RS latch is shown below.



S	R	Q	Q+
0	0	0	0
0	0	1	1
0	1	X	0
1	0	X	1
1	1	X	0

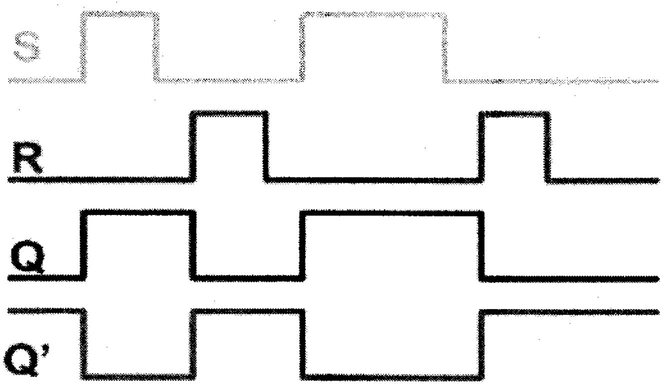
The operation has to be analyzed with the 4 inputs combinations together with the 2 possible previous states.

- **When S = 0 and R = 0:** If we assume Q = 1 and Q' = 0 as initial condition, then output Q after input is applied would be Q = (R + Q')' = 1 and Q' = (S + Q)' = 0. Assuming Q = 0 and Q' = 1 as initial condition, then output Q after the input applied would be Q = (R + Q')' = 0 and Q' = (S + Q)' = 1. So it is clear that when both S and R inputs are LOW, the output is retained as before the application of inputs. (i.e. there is no state change).
- **When S = 1 and R = 0:** If we assume Q = 1 and Q' = 0 as initial condition, then output Q after input is applied would be Q = (R + Q')' = 1 and Q' = (S + Q)' = 0. Assuming Q = 0 and Q' = 1 as initial condition, then output Q after the input applied would be Q = (R + Q')' = 1 and Q' = (S + Q)'

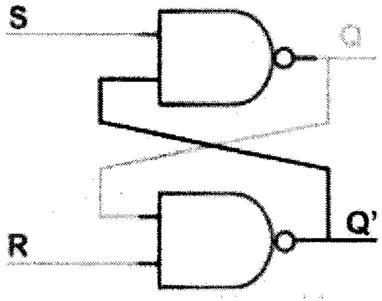
= 0. So in simple words when S is HIGH and R is LOW, output Q is HIGH.

- **When S = 0 and R = 1:** If we assume Q = 1 and Q' = 0 as initial condition, then output Q after input is applied would be Q = (R + Q')' = 0 and Q' = (S + Q)' = 1. Assuming Q = 0 and Q' = 1 as initial condition, then output Q after the input applied would be Q = (R + Q')' = 0 and Q' = (S + Q)'
= 1. So in simple words when S is LOW and R is HIGH, output Q is LOW.
- **When S = 1 and R =1 :** No matter what state Q and Q' are in, application of 1 at input of NOR gate always results in 0 at output of NOR gate, which results in both Q and Q' set to LOW (i.e. Q = Q'). LOW in both the outputs basically is wrong, so this case is invalid.

The waveform below shows the operation of NOR gates based RS Latch.



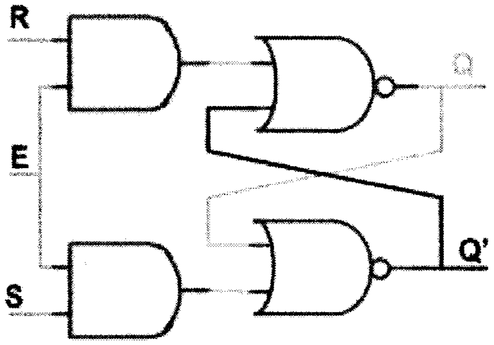
It is possible to construct the RS latch using NAND gates. The circuit and Truth table of RS latch using NAND is shown below.



S	R	Q	Q+
1	1	0	0
1	1	1	1
0	1	X	0
1	0	X	1
0	0	X	1

RS Latch with Clock

We have seen this circuit earlier with two possible input configurations: one with level sensitive input and one with edge sensitive input. The circuit below shows the level sensitive RS latch. Control signal "Enable" E is used to gate the input S and R to the RS Latch. When Enable E is HIGH, both the AND gates act as buffers and thus R and S appears at the RS latch input and it functions like a normal RS latch. When Enable E is LOW, it drives LOW to both inputs of RS latch. As we saw in previous page, when both inputs of a NOR latch are low, values are retained (i.e. the output does not change).

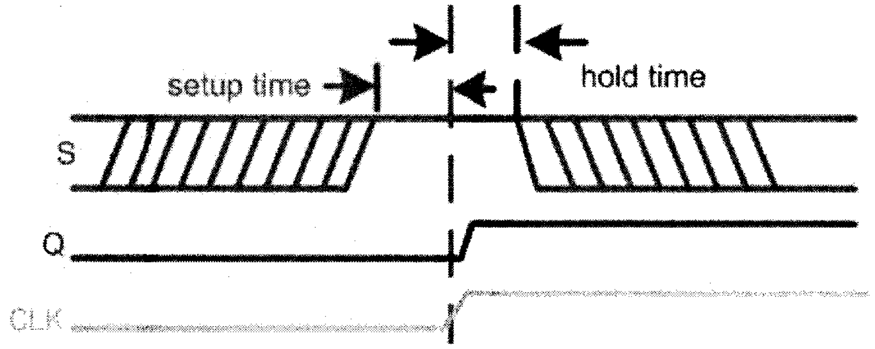


Setup and Hold Time -For synchronous flip-flops, we have special requirements for the inputs with respect to clock signal input. They are,

- **Setup Time:** Minimum time period during which data must be stable before the clock makes a valid transition. For example, for a posedge triggered flip-flop, with a setup time of 2 ns, Input Data (i.e. R and S in the case of RS flip-flop) should be stable for at least 2 ns before clock makes transition from 0 to 1.
- **Hold Time:** Minimum time period during which data must be stable after the clock has made a valid transition. For example, for a posedge triggered flip-flop, with a hold time of 1 ns. Input Data (i.e. R and S in the case of RS flip-flop) should be stable for at least 1 ns after clock has made transition from 0 to 1.

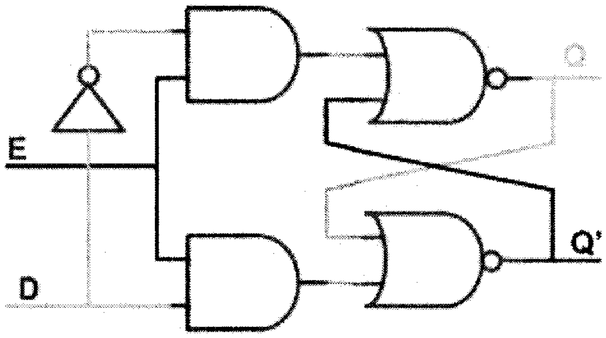
If data makes transition within this setup window and before the hold window, then the flip-flop output is not predictable, and flip-flop enters what is known as **meta stable state**. In this state flip-flop output oscillates between 0 and 1. It takes some time for the flip-flop to settle down. The whole process is called **metastability.**

The waveform below shows input S (R is not shown), and CLK and output Q (Q' is not shown) for a SR posedge flip-flop.



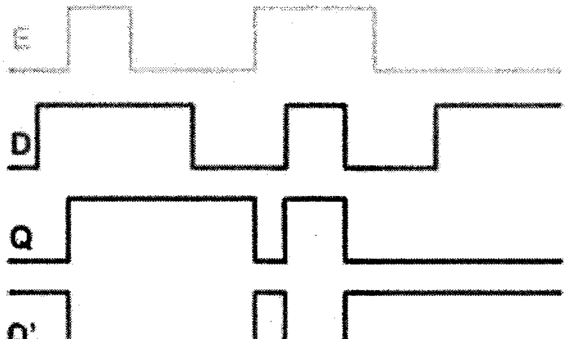
✓ D Latch

The RS latch seen earlier contains ambiguous state; to eliminate this condition we can ensure that S and R are never equal. This is done by connecting S and R together with an inverter. Thus we have D Latch: the same as the RS latch, with the only difference that there is only one input, instead of two (R and S). This input is called D or Data input. D latch is called D transparent latch for the reasons explained earlier. Delay flip-flop or delay latch is another name used. Below is the truth table and circuit of D latch. In real world designs (ASIC/FPGA Designs) only D latches/Flip-Flops are used.



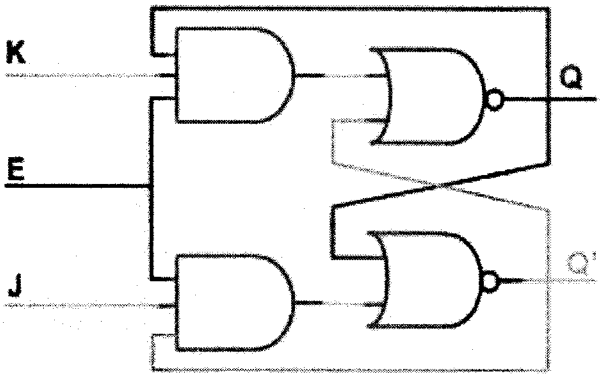
D	Q	Q+
1	X	1
0	X	0

Below is the D latch waveform, which is similar to the RS latch one, but with R removed.



✓ JK Latch

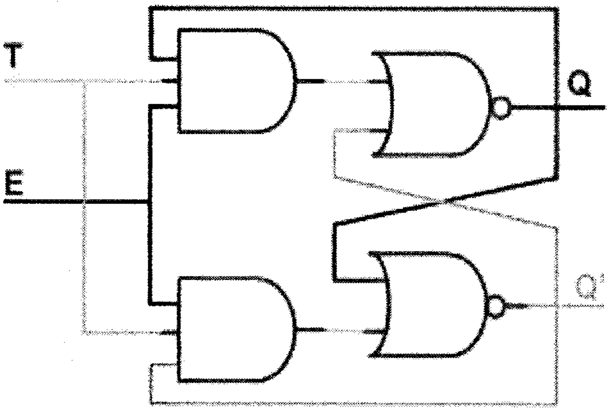
The ambiguous state output in the RS latch was eliminated in the D latch by joining the inputs with an inverter. But the D latch has a single input. JK latch is similar to RS latch in that it has 2 inputs J and K as shown figure below. The ambiguous state has been eliminated here: when both inputs are high, output toggles. The only difference we see here is output feedback to inputs, which is not there in the RS latch



J	K	Q
1	1	0
1	1	1
1	0	1
0	1	0

✓ T Latch

When the two inputs of JK latch are shorted, a T Latch is formed. It is called T latch as, when input is held HIGH, output toggles.

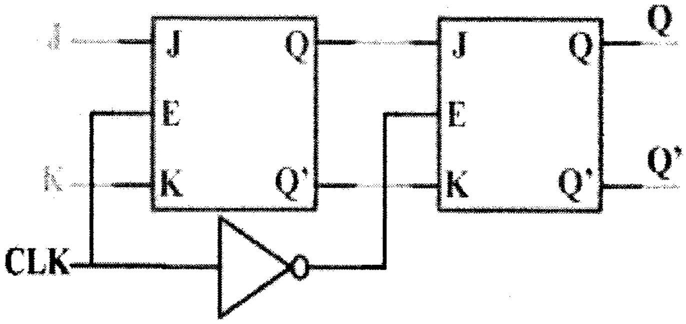


T	Q	Q+
1	0	1
1	1	0
0	1	1
0	0	0

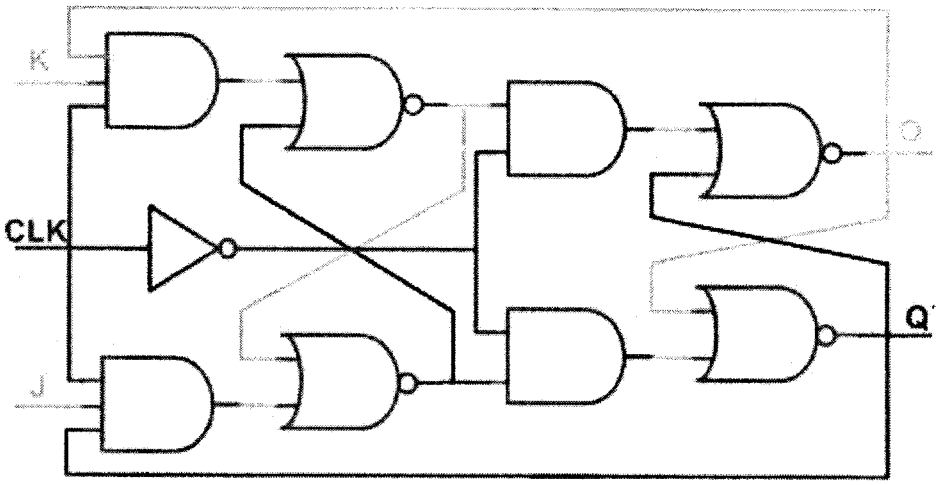
✓ JK Master Slave Flip-Flop

All sequential circuits that we have seen in the last few pages have a problem (All level sensitive sequential circuits have this problem). Before the enable input changes state from HIGH to LOW (assuming HIGH is ON and LOW is OFF state), if inputs changes, then another state transition occurs for the same enable pulse. This sort of multiple transition problem is called **racing.**

If we make the sequential element sensitive to edges, instead of levels, we can overcome this problem, as input is evaluated only during enable/clock edges.



In the figure above there are two latches, the first latch on the left is called master latch and the one on the right is called slave latch. Master latch is positively clocked and slave latch is negatively clocked.

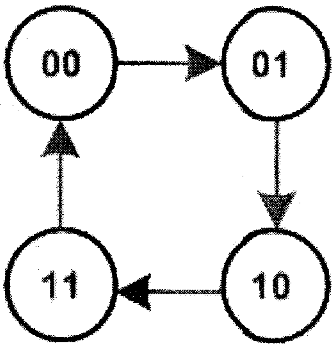


3.3 SEQUENTIAL CIRCUITS DESIGN

We saw in the combinational circuits section how to design a combinational circuit from the given problem. We convert the problem into a truth table, then draw K-map for the truth table, and then finally draw the gate level circuit for the problem. Similarly we have a flow for the sequential circuit design. The steps are given below.

- Draw state diagram.
- Draw the state table (excitation table) for each output.
- Draw the K-map for each output.
- Draw the circuit.

- **State Diagram** -The state diagram is constructed using all the states of the sequential circuit in question. It builds up the relationship between various states and also shows how inputs affect the states.
  Let's consider designing the 2 bit up counter (Binary counter is one which counts a binary sequence) using the T flip-flop.

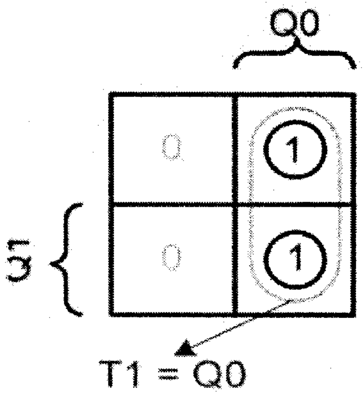


DESIGN

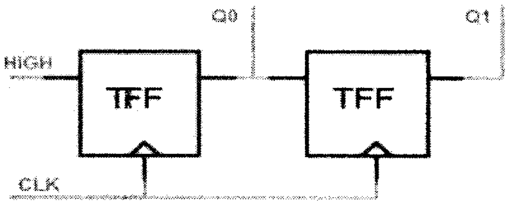
- **State Table** - The state table is the same as the excitation table of a flip-flop, i.e. what inputs need to be applied to get the required output. In other words this table gives the inputs required to produce the specific outputs.

Q1	Q0	Q1+	Q0+	T1	T0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

- **K-map** -The K-map is the same as the combinational circuits K-map. Only difference: we draw K-map for the inputs i.e. T1 and T0 in the above table. From the table we deduct that we don't need to draw K-map for T0, as it is high for all the state combinations. But for T1 we need to draw the K-map as shown below, using SOP.



- **Circuit-** There is nothing special in drawing the circuit, it is the same as any circuit drawing from K-map output. Below is thecircuit of 2-bit up counter using the T flip-flop.



3.4 SHIFT REGISTER

Register:
A set of n flip-flops.
Each flip-flop stores one bit.

Two basic functions: data storage and data movement

Shift Register:
A register that allows each of the flip-flops to pass the stored information to its adjacent neighbor.

A **shift register** is a cascade of Flip flops, sharing the same clock, which has the output of any one but the last flip-flop connected to the "data" input of the next one in the chain, resulting in a circuit that shifts by one position the one- dimensional "bit array" stored in it, *shifting in* the data present at its input and *shifting out* the last bit in the array, when enabled to do so by a transition of the clock input. More generally, a **shift register** may be multidimensional, such that its "data in" input and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Types of shift register

Shift registers can have a combination of serial and parallel inputs and outputs, including serial-in, parallel-out (SIPO) and parallel-in, serial-out (PISO) types. There are also types that have both serial and parallel input and types with serial and parallel output. There are also bi-directional shift registers which allow you to vary the direction of the shift register. The serial input and outputs of a register can also be connected together to create a circular shift register. One could also create multi-dimensional shift registers, which can perform more complex computation.

✓ **Serial-in, serial-out**

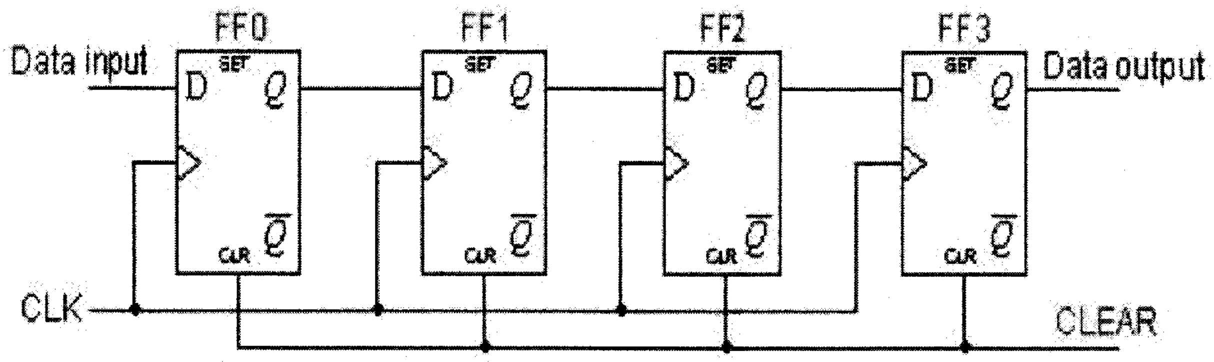
Destructive readout- These are the simplest kind of shift register. The data string is presented at 'Data In', and is shifted right one stage each time 'Data Advance' is brought high. At each advance, the bit on the far left (i.e. 'Data In') is shifted into the first flip-flop's output. The bit on the far right (i.e. 'Data Out') is shifted out and lost.The data are stored after each flip-flop on the 'Q' output, so there are four storage 'slots' available in this arrangement, hence it is a 4-Bit Register. To give an idea of the shifting pattern, imagine that the register holds 0000 (so all storage slots are empty).

As 'Data In' presents 1,1,0,1,0,0,0,0 (in that order, with a pulse at 'Data Advance' each time. This is called clocking or strobing) to the register, this is the result. The left hand column corresponds to the left-most flip-flop's output pin, and so on.So the serial output of the entire register is 11010000 (). As you can see if we were to continue to input data, we would get exactly what was put in, but offset by four 'Data Advance' cycles. This arrangement is the hardware equivalent of a queue. Also, at any time, the whole register can be set to zero by bringing the reset (R) pins high. This arrangement performs destructive readout -each datum is lost once it been shifted out of the right-most bit.

Non-destructive readout- Non-destructive readout can be achieved using the configuration shown

below. Another input line is added - the Read/Write Control. When this is high (i.e. write) then the shift register behaves as normal, advancing the input data one place for every clock cycle, and data can be lost from the end of the register. However, when the R/W control is set low (i.e. read), any data shifted out of the register at the right becomes the next input at the left, and is kept in the system. Therefore, as long as the R/W control is set low, no data can be lost from the system.

Example: Basic four-bit shift register



The operation of the circuit is as follows,

- The register is first cleared, forcing all four outputs to zero.
- The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0).
- During each clock pulse, one bit is transmitted from left to right. Assume a data word to be 1001.
- The least significant bit of the data has to be shifted through the register from FF0 to FF3.

In order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero.

FF0	FF1	FF2	FF3	
0	0	0	0	1001

The data is loaded to the register when the control line is HIGH (ie WRITE). The data can be shifted out of the register when the control line is LOW (ie READ).

Clear	FF0	FF1	FF2	FF3
1001	0	0	0	0

Write

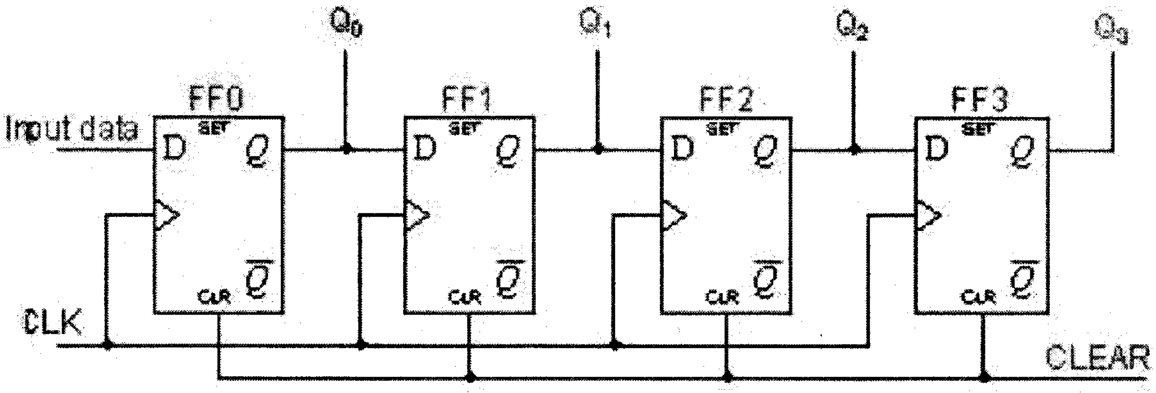
FF0	FF1	FF2	FF3	
1	0	0	1	0000

Read

FF0	FF1	FF2	FF3	
1	0	0	1	1001

✓ **Serial-in, parallel-out**

This configuration allows conversion from serial to parallel format. Data are input serially, as described in the SISO section above. Once the data has been input, it may be either read off at each output simultaneously, or it can be shifted out and replaced.



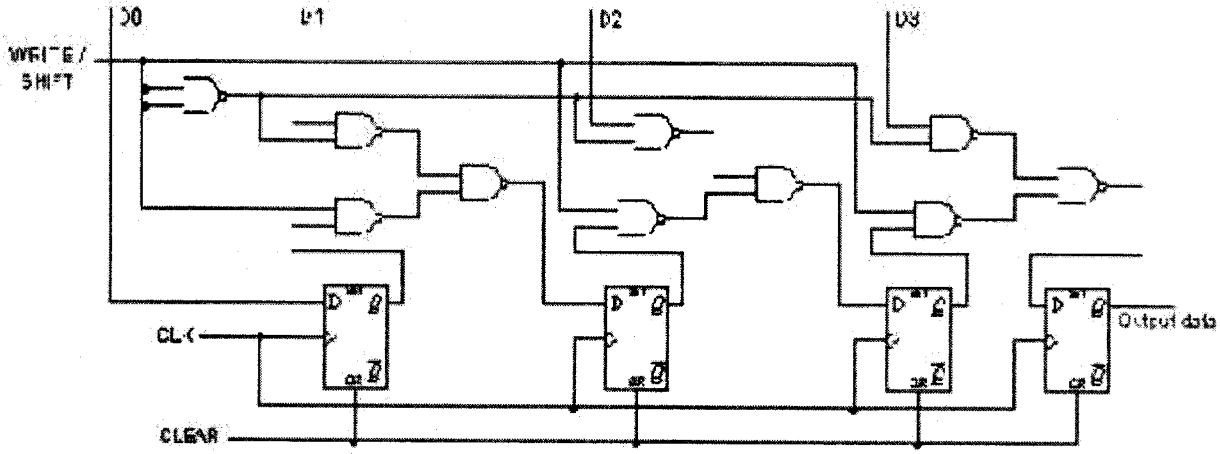
In the table below, we can see how the four-bit binary number 1001 is shifted to the Q outputs of the register.

Clear	FF0	FF1	FF2	FF3
1001	0	0	0	0
-	1	0	0	0
-	0	1	0	0
-	0	0	1	0
-	1	0	0	1

✓ **Parallel-in, serial-out**

This configuration has the data input on lines D1 through D4 in parallel format. To write the data to the register, the Write/Shift control line must be held LOW. To shift the data, the W/S control line is brought HIGH and the registers are clocked. The arrangement now acts as a SISO shift register, with D1 as the Data Input. However, as long as the number of clock cycles is not more than the length of the data-string, the Data Output, Q, will be the parallel data read off in order.

Example: A four-bit parallel in - serial out shift register is shown below. The circuit uses D flip -flops and NAND gates for entering data (ie writing) to the register.



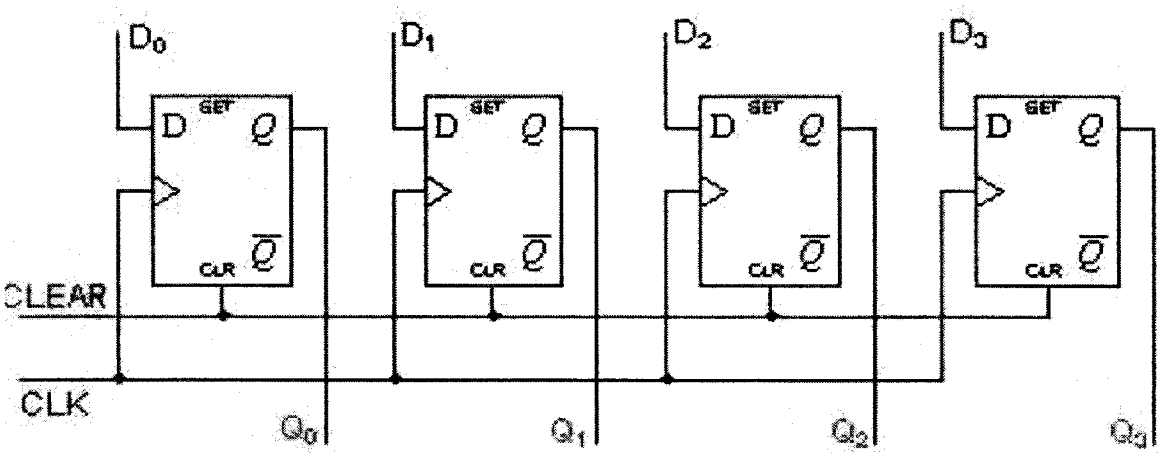
D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data

can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs right shift operation on the application of a clock pulse, as shown in the table below.

✓ **Parallel-in, parallel-out**

This configuration allows conversion from parallel to parallel format. Data input are in parallel, as described in the PISO section above. Once the data has been input, it may be either read off at each output simultaneously, or it can be shifted out and replaced.

	Q0	Q1	Q2	Q3	
Clear	0	0	0	0	
Write	1	0	0	1	
Shift	1	0	0	1	
-	1	1	0	0	1
-	1	1	1	0	01
	1	1	1	1	001
	1	1	1	1	1001

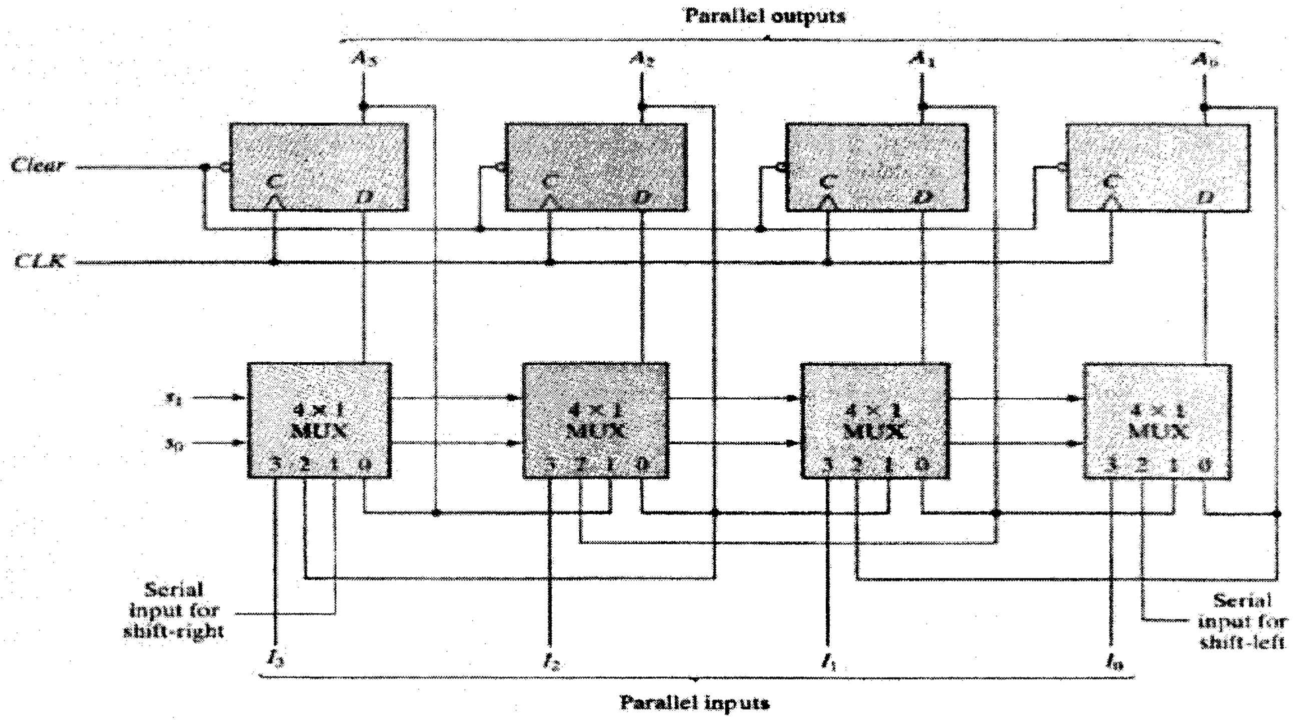


The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously

✓ **Universal shift register**

A register capable of shifting in one direction only is a unidirectional shift register .One that can shift in both directions is a bidirectional shift register. If the register has both shifts and parallel-loads, it is referred as **universal shift register**. The circuit consists of four D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0.

Figure: Block diagram of 4-bit universal shift register.



Mode control		**Register Operation**
s1	**s0**	
0	0	No change
0	1	Shift right
0	0	Shift left
1	1	Parallel load

Applications of shift registers

Shift registers can be found in many applications. Here is a list of a few. 1. To produce time delay

The serial in -serial out shift register can be used as a time delay device. The amount of delay can be controlled by:

1. The number of stages in the register
2. The clock frequency

3. To convert serial data to parallel data
4. To simplify combinational logic.

3.5 COUNTERS

In digital logic and computing, a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship 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,

- ✓ 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, and count in binary, or sometimes binary coded decimal. Many types of counter circuit are available as digital building blocks, for example a number of chips in the 4000 series implement 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 to one 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 alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip flop (remembering to invert the output to the input), you will get another 1 bit counter that counts half as fast. Putting them together yields a two bit counter:

✓ Synchronous counters

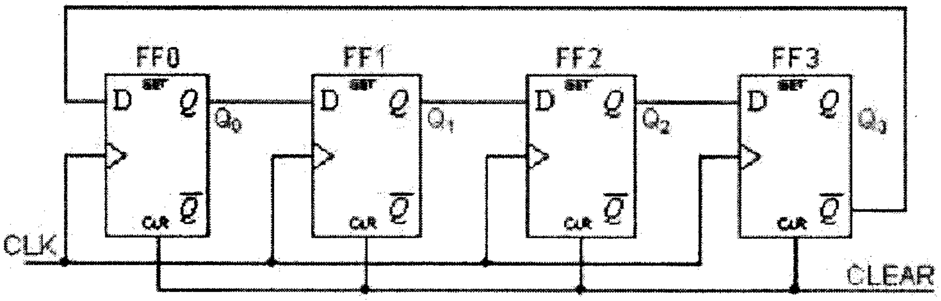
Where a stable count value is important across several bits, which is the case in most counter systems, synchronous counters are used. These also use flip-flops, either the D-type or the more complex J-K type, but here, each stage is clocked simultaneously by a common clock signal. Logic gates between each stage of the circuit control data flow from stage to stage so that the desired count behavior is realized. Synchronous counters can be designed to count up or down, or both according to a direction input, and may be presetable via a set of parallel "jam" inputs. Most types of hardware-based counter are of this type.

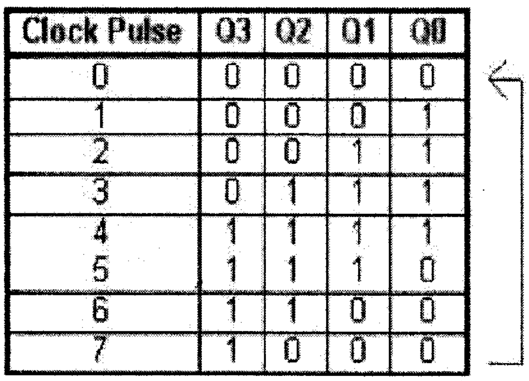
A simple way of implementing the logic for each bit of an ascending counter (which is what is shown in the image to the right) is for each bit to toggle when all of the less significant bits are at a logic high state. For example, bit 1 toggles when bit 0 is logic high; bit 2 toggles when both bit 1 and bit 0 are logic high; bit 3 toggles when bit 2, bit 1 and bit 0 are all high; and so on.

✓ Johnson counters

A Johnson counter is a special case of shift register, where the output from the last stage is inverted and fed back as input to the first stage. A pattern of bits equal in length to the shift register thus circulates

indefinitely. These counters are sometimes called "walking ring" counters, and find specialist applications, including those similar to the decade counter, digital to analogue conversion, etc.





Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0

The apparent disadvantage of this counter is that the maximum available states are not fully utilized. Only eight of the sixteen states are being used.

✓ Decade counters

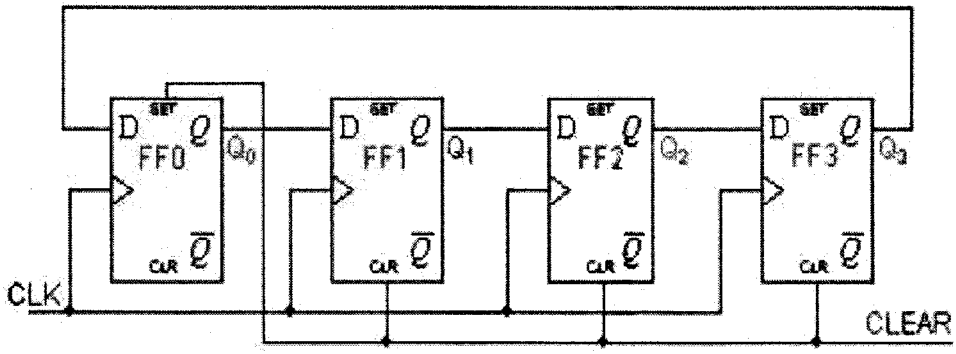
Decade counters are a kind of counter that counts in tens rather than having a binary representation. Each output will go high in turn, starting over after ten outputs have occurred. This type of circuit finds applications in multiplexers and demultiplexers, or wherever a scanning type of behaviour is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

✓ Up-Down Counters

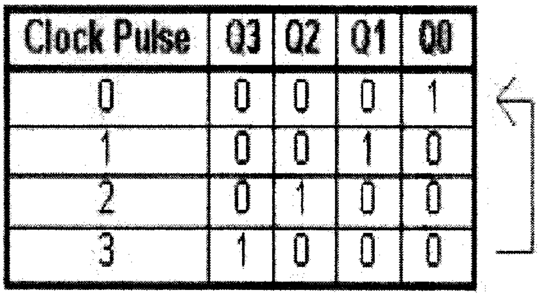
It is a combination of up counter and down counter, counting in straight binary sequence. There is an up-down selector. If this value is kept high, counter increments binary value and if the value is low, then counter starts decrementing the count. The Down counters are made by using the complemented 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 basically a circulating shift register in which the output of the most significant stage is fed back to the input of the least significant stage. The following is a 4-bit ring counter constructed from D flip-flops. The output of each stage is shifted into the next stage on the positive edge of a clock pulse. If the CLEAR signal is high, all the flip -flops except the first one FF0 are reset to 0. FF0 is preset to 1 instead.



Since the count sequence has 4 distinct states, the counter can be considered as a mod-4 counter. Only 4 of the maximum 16 states are used, making ring counters very inefficient in terms of state usage. But the major advantage of a ring counter over a binary counter is that it is self-decoding. No extra decoding circuit is needed to determine what state the counter is in.



Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

Applications of counters:

- **Watches**
- **Clocks**
- **Alarms**
- **Web** browser refresh

UNIT IV

Multivibrators Classification of multiiibrators – Astable, monostable, bistable multivibrators using operational amplifier. D/A and A/D converters: Binary weighted register D/A converter using Op-Amp – R-2R ladder D/A converter with Op-Amp – Analog to Digital converters (ADC) – their characteristics.

Introduction

Systems for generating and processing pulses make extensive use of **multivibrators**; these are circuits which have two states. There are three types of multivibrator: **astable** (free-running), **monostable** (one-shot), and **bistable** (flip-flop). There are many ways of implementing each type, and many variants.

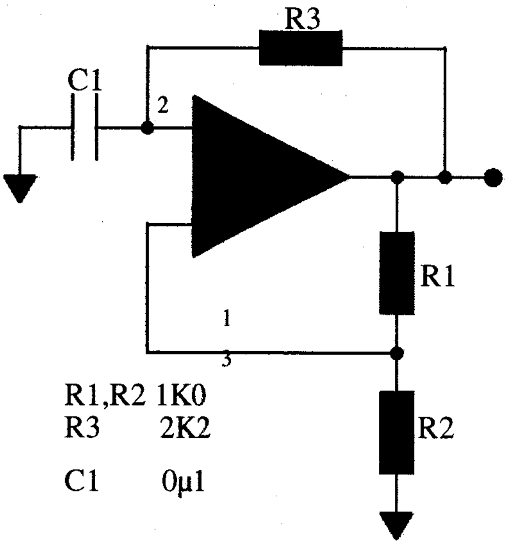
Note: All the circuits in this document operate by using positive feedback to drive the op-amp into saturation, it is therefore not the case that the two inputs of the op-amp can be assumed to be at the same potential. See the comments on Worksheet10 regarding op-amps vs comparators.

Astable Multivibrator

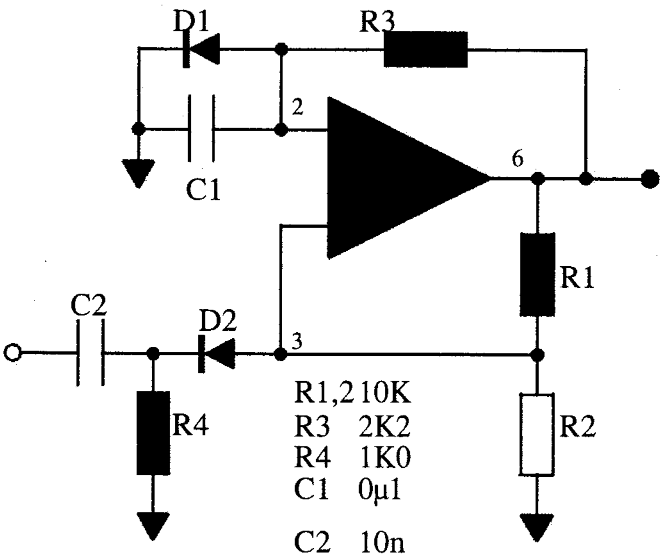
The two states of circuit are only stable for a limited time and the circuit switches between them with the output (node 6) alternating between positive and negative saturation values ±*VS*. Analysis of this circuit starts with the assumption that at time $t = 0$ the output has just switched to state 1 ($V6 = +VS$), and the transition would have occurred when

$$V2 = V6\ (\text{state } 0)\frac{R2}{R1 + R2} \quad \text{where } V6\ (\text{state } 0) = \ VS\ .$$

In state 1, the voltage across the capacitor increases as a result of current flowing through *R*3 from



Astable Multivibrator



Monostable Multivibrator

/

its initial value V2(t=0) = -   Vs R2/(R1+R2)

until V2(t=0)=V3(state 1) = Vs R2/(R1+R2)

when the output from the op-amp switches back to state 0. Then the capacitor discharges until, at time = 0 , the output switches from state 0 back to state 1, and the whole sequence restarts. t is straightforward to show that

$t_0 = C_1R_3\ln(1+2R_2/R_1)$

Monostable Multivibrator

A diode connected in parallel with the timing capacitor of the astable circuit will prevent the inverting input of the amplifier from going positive. The (permanently) stable state of this circuit has

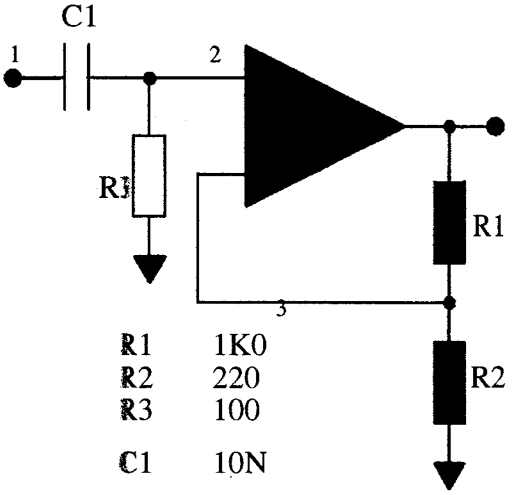
V6=Vs   with node 2 clamped to *0.6 V* by diode *D* 1, and node 3 at

V3(state 1) = 0.6 V +

Vs $R_4/(R_5 + R_4)$

A sufficiently large pulse at node 3, generated by a negative-going edge at the trigger input (node 1), will switch the circuit into its temporary state (6 = S) and, after a delay

$$t0 = C1R3 \ln 1 + \frac{R2}{R1}$$



R1	1K0
R2	220
R3	100
C1	10N

Circuit; Bistable Multivibrator

while $C1$ charges through $R3$, the circuit switches back to its stable state.

Bistable Multivibrator

The above circuit shows an op-amp configured as a bistable multivibrator. The two stable states are $V6 = \pm VSS$ and the circuit is switched between these by a pulse of appropriate polarity applied to the inverting terminal (node 2) of the op-amp.

D/A and A/D Converters

Introduction

The outputs from sensors and communications receivers are analogue signals that have continuously varying amplitudes. In many systems it is convenient to record and/or process these signals within a digital circuit, which may be within a programmable device such as a microcontroller, microprocessor or a computer. In a digital circuit the signal will be represented as a list of binary numbers, with each number representing the amplitude of the signal at a specific time.

Decimal and Binary Numbers

Possibly because we have ten fingers we have developed the decimal number system based upon 10 and powers of 10. Although it is suitable for use by people the decimal number system isn't particularly suitable for use by other physical systems. In particular the digital logic circuits are based upon devices that either conduct or don't conduct. This means that digital logic circuits naturally have two states. This means that in digital logic circuits numbers have to be represented using only two symbols, usually written as 0 and 1. This means that digital circuits use binary numbers.

A good starting point for understanding binary numbers is the decimal numbers that we use everyday. We have all used this decimal number system for so long and so often that we probably don’t think about what numbers

this decimal system actually represent. In this number system each position represents the multiples of the power of 10 associated with that position that form part of the number. To represent these numbers we need 10 symbols (0,1,2,3,4,5,6,7,8,9) to represent the number of multiples. The decimal 1206 then represents $1\times10^3+2\times10^2+0\times10^1+6\times10^0$ (which is one thousand, two hundred and six).

The digital circuits used in programmable devices have only two states and by convention these two states are denoted using the symbols 0 and 1. With only two states/symbols available numbers have to be represented as powers of 2 rather than powers of 10. As in the decimal system counting in binary starts with 0 and this is followed by 1. In the decimal system the next number is representing by the symbol 2. However in binary there are only two symbols and so the next number has to be represented by increasing the power of 2 and so when counting in binary 1 is followed by 10 (following the example of decimal this is equivalent to the decimal number $1\times2^1+0\times2^0$). More generally as in decimal numbers any binary number can be understood using the associated powers of 2. For example

$$1010 = 1 \times 2^3+0 \times 2^2+1 \times 2^1+0 \times 2^0$$

Converting each of the powers of 2 to its decimal equivalent means that

Decimal equivalent of 1010 = $1 \times 2^3+0 \times 2^2+1 \times 2^1+0 \times 2^0 = 1 \times 8+0 \times 4+1 \times 2+0 \times 1 =10$

Each symbol in a binary number is known as a bit and a binary number is therefore a list or string of bits. The first bit in a string is known as the most-significant bit and the last one is known as the least significant bit. One convention is to label each bit with a subscript corresponding to the equivalent power of 2, so that for example the least significant bit (which represents the multiple of 2^0 within a binary number) is b_0. In this convention a four bit number is therefore b_3 b_2 b_1 b_0 and the equivalent number is

$$\text{Decimal} = b_3 \text{ x } 2^3 + b_2 \text{ x } 2^2 + b_1 \text{ x } 2^1 + b_0 \text{ x } 2^0$$

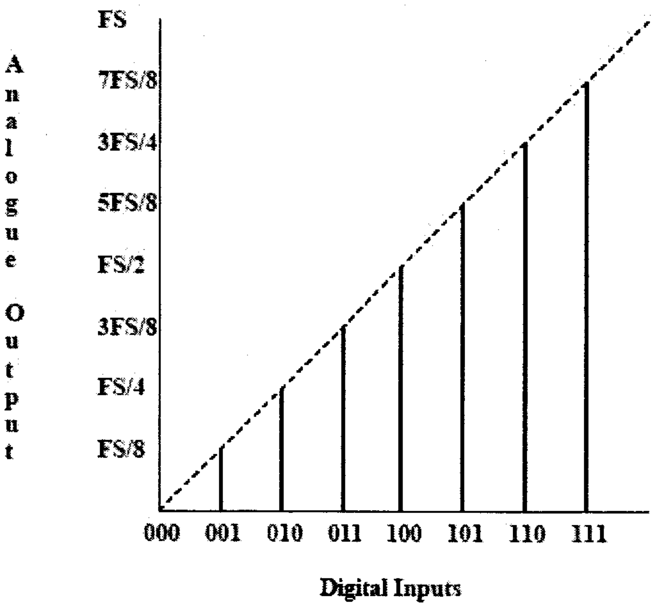
One limitation of this simple representation is that an n-bit binary number can only represent 2^n different values.

Data Converters

Conversion from an analogue signal to a digital number is performed by an <u>analogue-to-digital converter</u> (ADC). There are several different types of ADCs, some of which contain a digital-to-analogue converter (DAC) that converts a digital number to the equivalent analogue signal. When taken together with their independent role in creating analogue output signals to drive parts such as heaters and motors, this makes DACs a critical part of many systems.

It is therefore important to understand the operation of both DACs and ADCs.

Specification of D/A converters (DACs)



The ideal response of a 3-bit DAC, showing the analogue output voltage as a fraction of the full scale output FS. Each bar represents the output for a particular input and the dashed line shows the line connecting the ideal outputs.

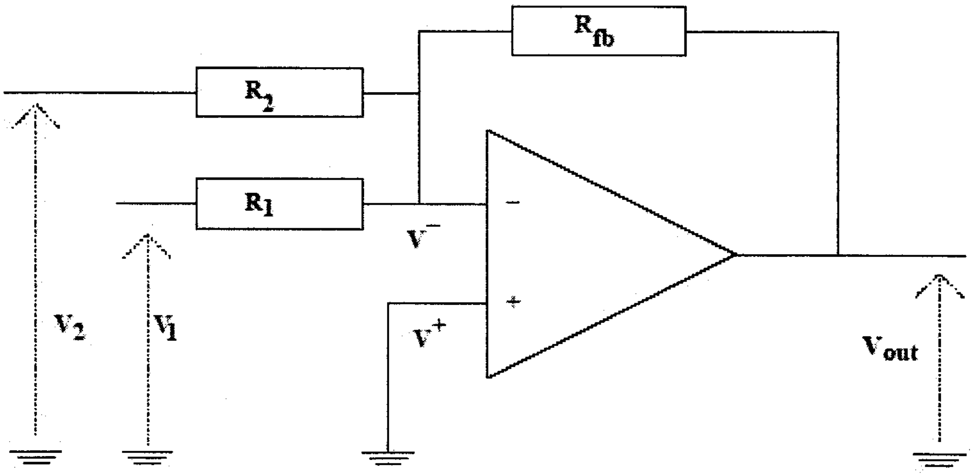
A digital to analogue converter (DAC) converters a digital input represented as a binary number to an analogue voltage (or current) that is proportional to the value of this input. The **ideal** relationship between the analogue output and digital input for a 3-bit converter is shown in Figure (34)[1].

[1] In this diagram 3 bits have been shown for clarity. However, in real instrumentation systems DACs with 8, 10, 12 and 14 bits are often used.

D/A Converter Architectures (DAC Architectures)

The Summing Amplifier

The basis operation required to create a DAC is the ability to add inputs that will eventually correspond to the contributions of the various bits of the digital input. In the voltage domain, that is if the input signals are voltages, addition can be achieved using the inverting summing amplifier shown in Figure (35).



An inverting summing amplifier

To understand how this circuit operates assume that the op-amp is ideal. Since the op-amp is ideal $V^- = V^+$, but, in this circuit $V^+ = 0$ and so the current flowing into this node from the two inputs is

$$I_{in} = \frac{V_1}{R_1} + \frac{V_2}{R_2}$$

Since no current flows into the inverting input of the ideal op-amp all this current must flow around the feedback loop through resistor R_{fb}.

This will only happen when the op-amp output voltage is

$$V_{out} = -I_{in}\, R_{fb}$$

which becomes

$$V_{out} = -\frac{V_1\, R_{fb}}{R_1} - \frac{V_2\, R_{fb}}{R_2}$$

Now if we assume that

$$R_{fb} = R_2 = 2R_1$$

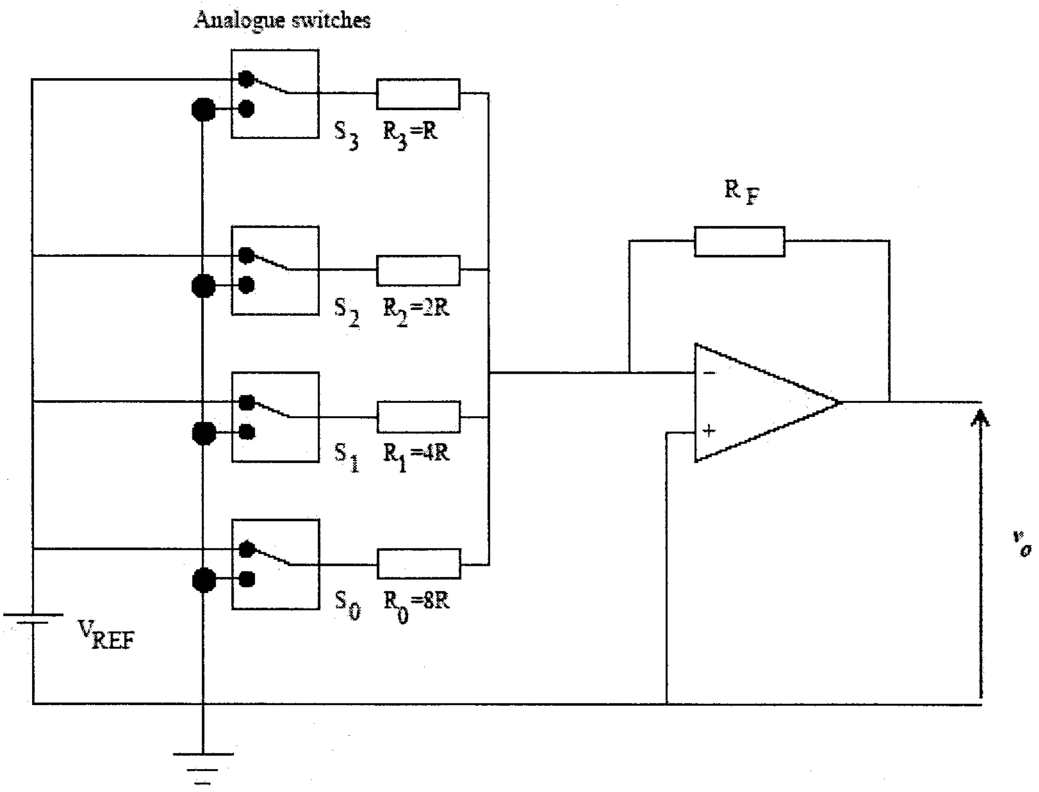
then

$$V_{out} = -(2V_1 + V_2)$$

This weighted combination of inputs is the principle behind the operation of many digital-to-analogue converters.

D/A Converters

The simplest way of convert a digital input word into a corresponding analogue voltage is to use an op-amp as a summing amplifier with a weighted resistor ``ladder", as shown in Figure (36).



A 4-bit DAC based upon summing the current through weighted resistors.

At the start of the conversion process, a **4-bit input code,** $B_0 - B_3$ **is applied to control the corresponding switches** $S_0 - S_3$ .Each switch S_n connects the resistor R_n to the voltage source V_{REF} when the corresponding

bit B_n is high. In contrast when B_n is low the resistor R_n is grounded. The other end of each resistor is connected to the summing junction of the op-amp. For a four bit converter in which the resistors are in the ratio
8 : 4 : 2 : 1, as shown in Figure (16), the total current flowing onto the inverting input of the op-amp is

$$I_{in} = V_{REF}(\frac{B_3}{R} + \frac{B_2}{2R} + \frac{B_1}{4R} + \frac{B_0}{8R})$$

this current then flows through R_F to generate the output voltage and hence

$$V_o = -\frac{R_F}{R} V_{REF}(B_3 + \frac{B_2}{2} + \frac{B_1}{4} + \frac{B_0}{8})$$

The output voltage V_o therefore represents a weighted sum of the input bits. If $R = 2R_F$ then the following relationship between digital inputs and the
analogue output voltage will be obtained:

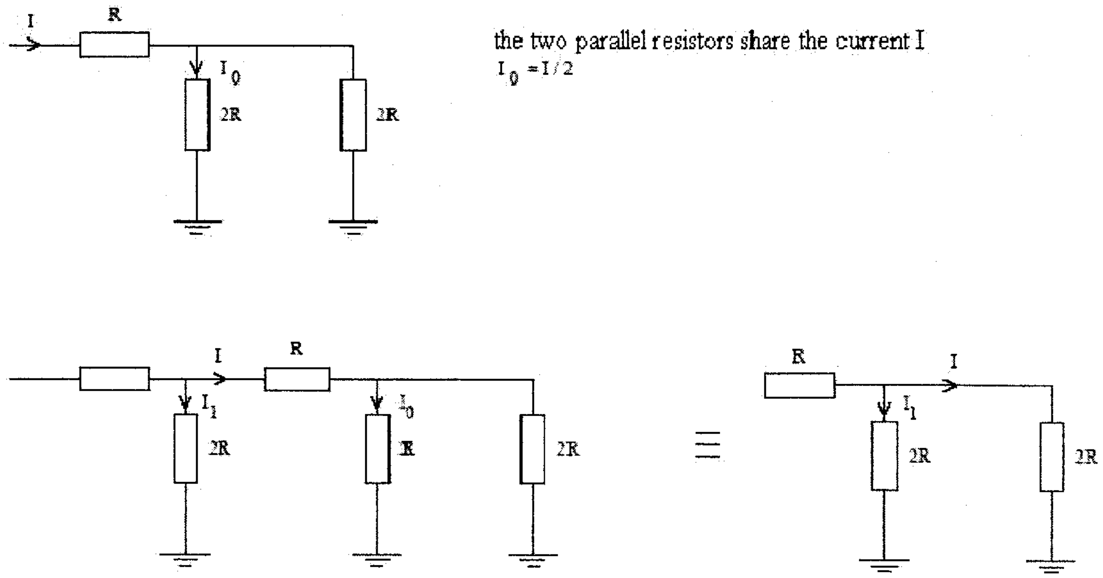
Digital input	V_o
0000	0
0001	$-1/16\ V_{REF}$
0010	$-2/16\ V_{REF}$
.	.
1110	$-14/16\ V_{REF}$
1111	$-15/16\ V_{REF}$

The circuit therefore achieves the desired functions. However, there are two main problems with this circuit:

(i) the output voltage from the reference voltage source must stay constant even when its output current is changing, i.e. its source resistance must be zero.

(ii) the resistor values must be very accurate and in the correct ratio to one another. Although this requirement can be achieved in an integrated circuit, the range of values required for, say, a 12-bit D/A converter (for example, 10 kΩ to 20.48 MΩ) makes it impractical.

For these reasons, a different type of resistor network is normally used, the "**R-2R ladder**", which can be constructed, as its name indicates, out of two values of resistors. This network, shown at the top of Figure (38), therefore avoids the need to create different resistance values.



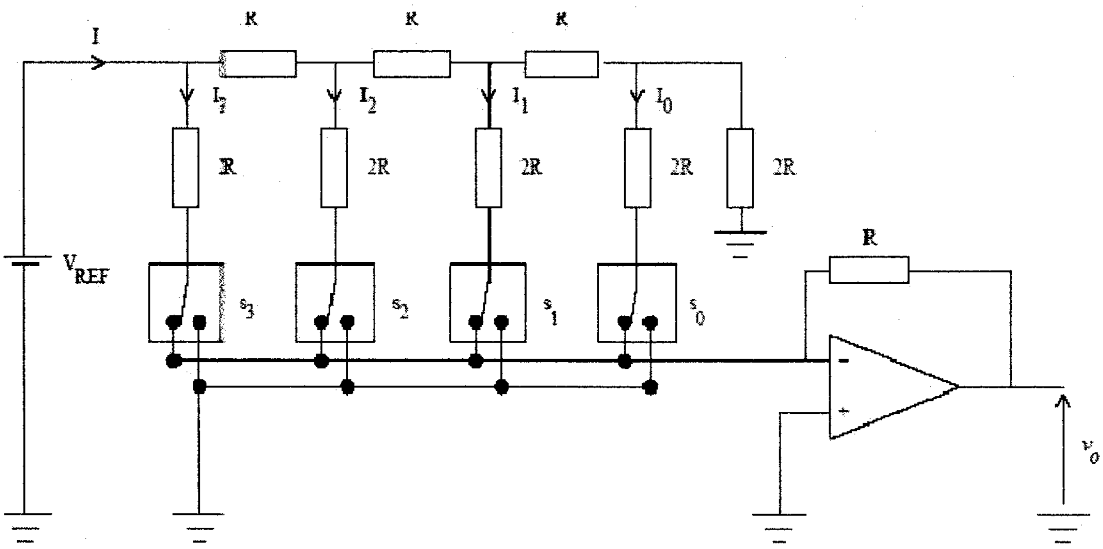
the two parallel resistors share the current I
$I_0 = I/2$

2R in parallel with 2R is equivalent to R

$I_1 = I = 2I_0$

The analysis of an R-2R ladder network

The trick to analysing the R-2R ladder network is to start from its right-hand end: As shown in Figure (37) at this end of the system there are two 2R resistors acting in parallel which combine to form an effective resistance R. This effective resistance then appears in series with another resistance R to form a resistance of 2R. However, this effective resistance of 2R is in parallel with another resistance 2R. Thus at each stage of the analysis of the ladder network, all elements to the right of a particular node are equivalent to a resistance of 2R.



An R-2R ladder 4-bit D/A converter.

The analysis of the ladder network means that for the network in Figure (38) the incoming current splits into two at each node and thus

$$I_2 = 2I_1 = 4I_0$$

and

$$I_3 = V_{REF}/2R = 2I_2 = 4I_1 = 8I_0$$

As with the previous circuit, each bit B_n of the digital code controls a switch S_n. When $B_n = 1$, the switch S_n directs current I_n towards the summing junction; otherwise the current flows straight down to ground. The DAC output voltage is therefore determined by a current that is proportional to the weighted sum of the input bits as required.

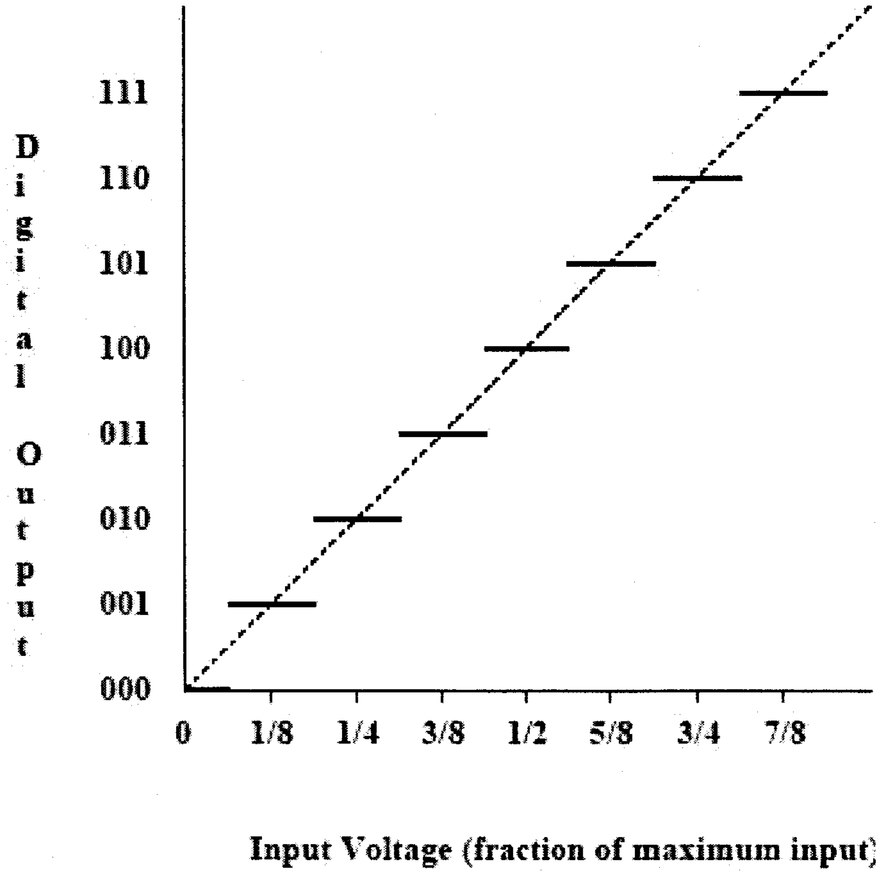
The other advantage of this architecture is that the inverting input of the op-amp is a virtual earth and hence one end of each the 2R resistors is always connected to 'earth'. This means that the current flowing through each branch of the ladder network is independent of the switch

conditions and hence the digital input. The significance of this is that it means that the total current supplied by the voltage source is constant and the circuit performance is independent of the output impedance of thevoltage source.

One potentially useful modification to this basic architecture is to use a variable voltage source, possibly formed by a second DAC, to create a variable reference voltage. The analogue output signal is then proportional to the product of the variable reference voltage and the input binary number; this type of device is usually known as a **multiplying** DAC or MDAC.

A/D converters (ADCs)

Analogue to Digital (A/D) conversion is the process whereby an analogue signal is converted into a corresponding binary number, the digital output. The ideal relationship between the analogue input and the digital output for a 3-bit A/D converter is shown in Figure (39). The input analogue values are **quantised** by dividing the continuous analogue input range into 8 discrete steps or code ranges.



The ideal response of a 3-bit ADC.

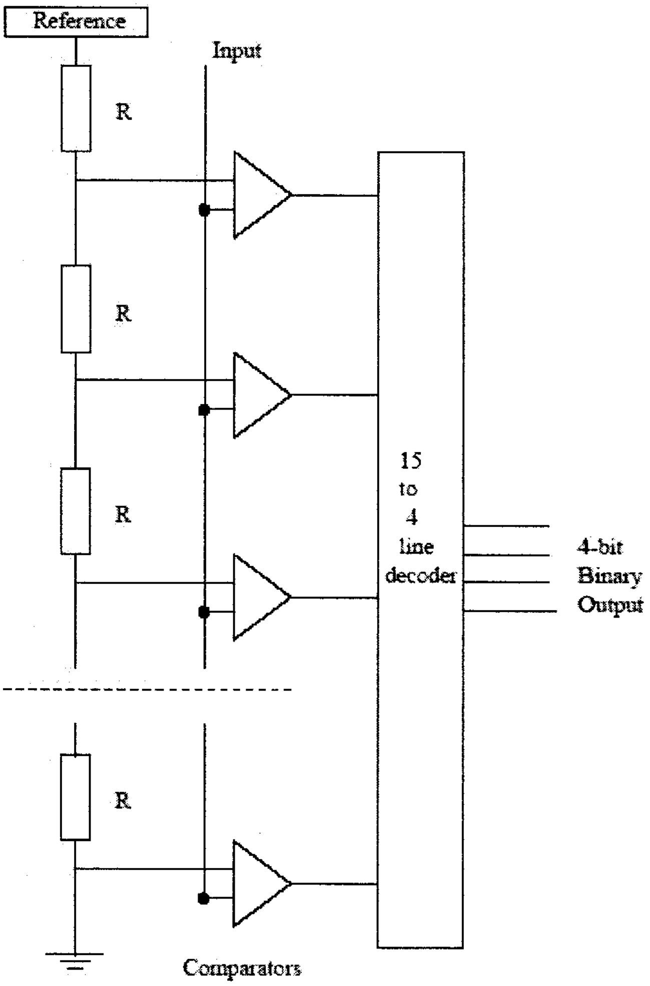
Since the ADC is unable to distinguish among different values in the same code range the output can have an error as large as 1/2LSB. This **quantisation error** is an intrinsic limitation of representing a continuous input by a finite set of output numbers. The first approach to minimising the effects of quantisation errors is to ensure that the maximum expected amplitude of the input signal matches the input range of the ADC. This usually means that amplifiers are needed between the signal source and the ADC. By using an active low-pass filter this amplification function can be performed by the anti-aliasing filter.

The other method of reducing quantisation error is to increase the number of output bits. For example 2^{12} 4096 and hence a 12-bit A/D converter can resolve a signal to 1 part in 4096, or 0.024% of the maximum input.

The two types of A/D converter that we will discuss are: Parallel
converters
Successive-Approximation converters

The choice between the types of converter is made on the grounds of the cost, resolution and speed required for a particular application.

Parallel ADCs



A schematic diagram of a flash A/D converter.

Parallel encoding (sometimes known as ``flash" encoding) is the fastest (but also the most expensive) method of A/D conversion. In this architecture, shown in Figure (40) an n-bit conversion is achieved by simultaneously comparing the analogue input with $2^n - 1$ reference levels. These reference levels are usually generated by a chain of identical resistors connected in series. **Each of these references is compared to the input by**

a circuit known as a comparator. This is a circuit with a very high differential gain so that the output saturates to a maximum value when the voltage on the non-inverting input is higher than the voltage on the inverting input. Otherwise the comparator output saturates to a minimum value.

A comparator array can therefore be designed so that the outputs from all comparators whose reference voltage is below the common input saturate to a maximum output value. The output of all the other comparators will saturate at the minimum value whilst all those with references above the input have the minimum output voltage.

The maximum output voltage can then be interpreted as a logical 1, whilst any low output is interpreted as logical 0 so that the $2^n - 1$ outputs represent the analogue input value by the position of the transition between ones and zeros. The position of the transition between the ones and the zeros moves as the analogue input voltage changes. This representation is therefore often referred to as a thermometer code. The final stage of the conversion process is to use a digital circuit, known as an encoder, to convert this unusual representation of the input to a more conventional binary number.

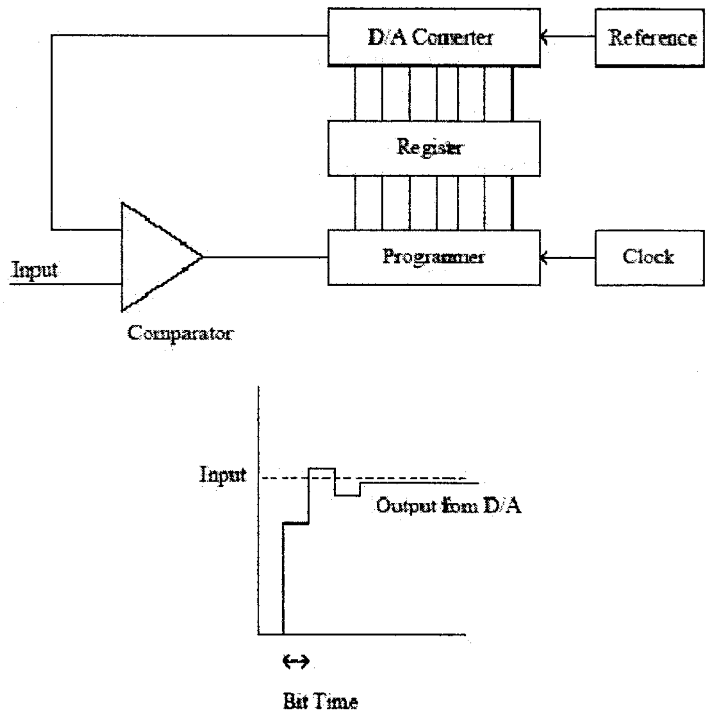
One advantage of the flash converter is that it is conceptually simple. However, its main advantage is the speed at which conversion can be achieved. Since the input is compared to all the reference values simultaneously the time required to perform a conversion, a parameter known as the **conversion time**, is simply the response time for the comparators and the encoder. This time is significantly shorter than the fastest alternative architectures. **The flash converter is therefore the fastest type of converter, the disadvantage of the flash converter is the large number of comparators and resistors required. This means that these converters will be expensive.** Furthermore, as the number of comparators increases the voltage difference between the reference inputs of two adjacent comparators reduces and the errors between reference levels caused by variations between the values of individual resistors must therefore be reduced. Since these variations are caused by slight differences in the sizes of different resistors any reduction in errors will only be achieved by using larger area resistors. Unfortunately, this simply further increases the cost of the final component.

Overall, flash converters are therefore fast, but, expensive.

Successive-Approximation ADCs

The successive-approximation converter shown in Figure (41) operates by approximating the analogue input signal with a binary code. This binary code is successively revising by changing each bit in the code until the best approximation is achieved. At each step in the approximation, the present estimate of the binary value corresponding to the analogue input signal is saved in the successive approximation register. The contents of this register are converted to an analogue signal by a DAC so that a single comparator can determine whether the approximation is larger or smaller than the input signal.

As shown at the bottom of Figure (41) the first approximation sets the most significant bit, the MSB, of the successive approximation register and resets all the other bits (i.e. makes them zero). If the DAC output (which is therefore equal, at this point, to half full-scale) is smaller than the analogue input, the MSB is left on; if the DAC output is too large, then the MSB is turned off. In the next clock cycle, the **next** most significant bit is set (i.e. at the DAC output is now equal to either 3/4 or 1/4 of full-scale, depending on whether the most significant bit was left on or not) and this new approximation is compared with the analogue input. Each successive bit is similarly tested. After the least significant bit has been tested, the conversion is complete and the output register contains the binary code.



A schematic diagram of the architecture of a successive-approximation ADC and the internally generated analogue signal (solid line) which is compared to the input (dashed line).

If the accuracy of conversion is to equal the resolution of the converter, **the input signal must remain constant within the analogue value of 1/2 LSB during the conversion time**

To quantify the limitation this places on the input signals that can be converted accurately assume that the input signal is a sinusoidal wave of frequency *f* and peak-to-peak amplitude V_{REF},i.e.

$$V_{in} = \tfrac{1}{2} V_{REF} \sin 2\pi f t$$

For an n-bit converter, 1/2 LSB (a simple estimate of the quantisation error) is equivalent to a voltage of

$$\tfrac{1}{2}\, V_{REF}/2^n$$

The rate of change of the input signal is:

$$\frac{dV_{in}}{dt} = \pi\, f\, V_{REF} \cos 2\pi f t$$

The maximum rate of change occurs when the input is zero and is given by:

$$|\, dV_{in}/dt \,|_{max} = \pi\, f\, V_{REF}$$

If the conversion time is t_c, then we must have:

$$|\, dV_{in}/dt \,|_{max} \,.\, t_c \leq \frac{1}{2} \,.\, \frac{V_{REF}}{2^n}$$

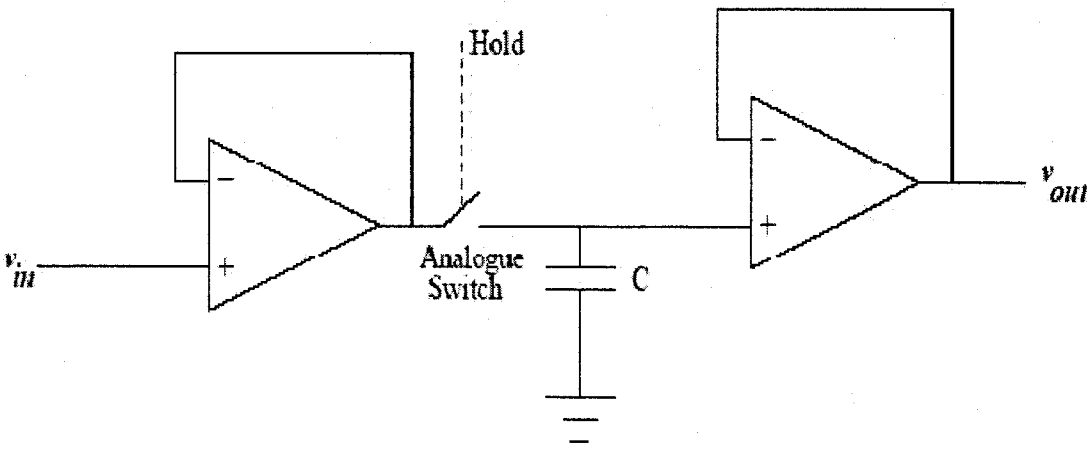
this can be re-written as:

$$\pi\, f\, V_{REF} \,.\, t_c \leq \frac{V_{REF}}{2^{n+1}}$$

which is equivalent to a maximum input frequency of

$$f_{max} = \frac{1}{\pi\, t_c\, 2^{n+1}}$$

For an 8-bit ADC with conversion time of 10 μs, this gives a maximum frequency of 62 Hz! This is obviously much too low for most applications. The problem that limits the maximum frequency that can be converted arises from the changes in the input signal during the conversion process. These changes can be avoided by using a **sample-and-hold** circuit just before the ADC input. As its name suggests this type of circuit **samples** the signal and then **holds** the sampled value until the conversion process is completed and a new sample is acquired.



A sample-and-hold circuit.

The basic sample-and-hold circuit consists of an analogue switch and a storage capacitor, as in the centre of Figure (42). The analogue switch is controlled by a signal, labelled Hold, which allows the input signal to pass through to the capacitor during the **aperture time** and disconnects it during the **hold time**. The value of the input signal v_{in} is therefore stored on the capacitor during the hold time. The choice of a value for this capacitor is a compromise between the need to minimise voltage changes caused by leakage currents during the hold interval (i.e. make C as large as possible) and the need to follow high-frequency input signals without them being low-pass filtered by the combination of the capacitor and the finite on-resistance of the switch (i.e. make C as small as possible). **In order to reduce leakage currents during the hold time, to prevent voltage changes, the voltage on the capacitor is sensed using an op-amp configured as a voltage follower. Similarly, the speed of the circuit is increased by detecting the input signal via a second op-amp acting as unity gain buffer that**

reduces the source impedance driving the capacitor during the aperture time.

With a sample-and-hold circuit on the input to a successive-approximation A/D converter the maximum operating frequency of the converter is now given by

$$1/\pi t_a 2^{n+1}$$

where t_a is the **aperture** time which can be just a few tens of ns; hence input signals whose frequency is several tens of kHz can now be converted to binary format with this type of A/D converter.

Summary

The outputs from sensors and communications receivers are analogue signals that have continuously varying amplitudes. In many systems it is convenient to record and/or process these signals within a digital circuit, which may be a microcontroller, microprocessor or a computer. In a digital circuit the signal will be represented as a list of binary numbers, with each number representing the amplitude of the signal at a specific time.

In the digital circuits used in microcontrollers, microprocessors and computers numbers are represented as a series of bits. Each bit can only have a value of either zero or one which means that the number is in base 2.

Conversion from an analogue signal to a digital number is performed by an analogue-to-digital converter (ADC). A digital to analogue converter (DAC) converters a digital input represented as a binary number to an analogue voltage (or current) that is proportional to the value of this input.

A DAC can be created using an R-2R ladder and an op-amp.

Analogue to Digital (A/D) conversion (ADC) is the process whereby an analogue signal is converted into a corresponding binary number, the digital output. The input analogue values are **quantised** by dividing the continuous analogue input range into 2^N discrete steps or code ranges. This rounding error gives rise to quantisation noise, which can be estimated using its maximum value $V_{max}/2^{N+1}$

In a flash ADC the input voltage is compared in parallel with many different reference voltages. The resulting system is conceptually simple, fast but expensive.

A successive-approximation converter operates by approximating the analogue input signal with a binary code. This binary code is successively revising by changing each bit in the code until the best approximation is achieved. The result is only valid if the input remains approximately constant during the conversion time. This means that the maximum input frequency has to be very small or a sample and hold circuit is used to sample the input voltage before it is converted.

UNIT V

Semiconductor Memories memory cell unit – ROM, RAM – Their classifications – ROM, PROM, EPROM, EEPROM, RAM,Static RAM, dynamic RAM, Memory read and memory write operations – Flash memory - Charge coupled Device (CCD).

Read only memory devices are a special case of memory where, in normal system operation, the memory is read but not changed. Read only memories are non-volatile, that is, stored informa-tion is retained when the power is removed. The main read only memory devices are listed below:

ROM (Mask Programmable ROM—also called "MROMs")

EPROM (UV Erasable Programmable ROM)

OTP (One Time Programmable EPROM)

EEPROM (Electrically Erasable and Programmable ROM)

Flash Memory - This device is covered in Section 10.

HOW THE DEVICE WORKS

The read only memory cell usually consists of a single transistor (ROM and EPROM cells consist of one transistor, EEPROM cells consist of one, one-and-a-half, or two transistors). The threshold voltage of the transistor determines whether it is a "1" or "0." During the read cycle, a voltage is placed on the gate of the cell. Depending on the programmed threshold voltage, the transistor will or will not drive a current. The sense amplifier will transform this current, or lack of current, into a "1" or "0." Figure 9-1 shows the basic principle of how a Read Only Memory works.

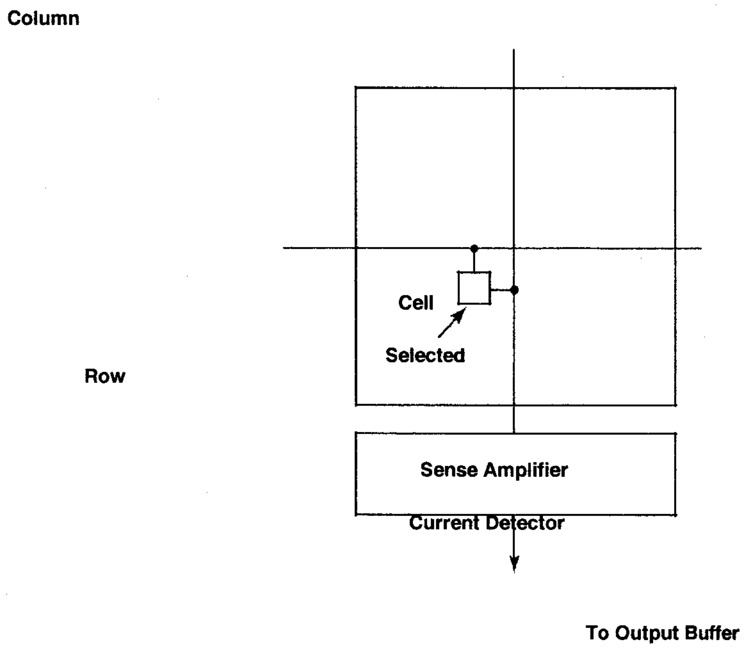


Figure Read Only Memory Schematic

MASK PROGRAMMABLE ROMs

Mask programmable read-only memories (ROMs) are the least expensive type of solid state memory. They are primarily used for storing video game software and fixed data for electronic equipment, such as fonts for laser printers, dictionary data in word processors, and sound data in electronic musical instruments.

ROM programming is performedduring IC fabrication. Several process methods can be used to program a ROM. These include

- Metal contact to connect a transistor to the bit line.
- Channel implant to create either an enhancement-mode transistor or a depletion-mode transistor.
- Thin or thick gate oxide,which creates either a standard transistor or a high threshold transistor, respectively.

The choice of these is a trade-offbetween process complexity, chip size, and manufacturing cycle time. A ROM programmed at the metal contact level will have the shortest manufacturing cycle time, as metallization is one of the last process steps. However, the size of the cell will be larger.

Figure 2 shows a ROM array programmed by channel implant. The transistor cell will have either a normal threshold (enhancement-mode device) or a very high threshold (higher than V_{CC} to assure the transistor will always be off). The cell array architecture is NOR. The different types of ROM architectures (NOR, NAND, etc.) are detailed in the flash memory section (Section 10) as they use the same principle.

Figure 3 shows an array of storage cells (NAND architecture). This array consists of single tran-sistors noted as devices 1 through 8 and 11 through 18 that is programmed with either a normal threshold (enhancement-mode device) or a negative threshold (depletion-mode device).

ROM Cell Size and Die Size

The cell size for the ROM is potentially the smallest of any type of memory device, as it is a single transistor. A typical 8Mbit ROM would have a cell size ofabout 4.5µm^2 for a 0.7µm feature size process, and a chip area of about 76mm^2. An announced 64Mbit ROM, manufactured with a 0.6µm feature size, has a 1.23µm^2 cell on a 200mm^2 die.

The ROM process is the simplest of all memory processes, usually requiring only one layer of polysilicon and one layer of metal. There are no special film deposition or etch requirements, so yields are the highest among all the equivalent-density memory chips

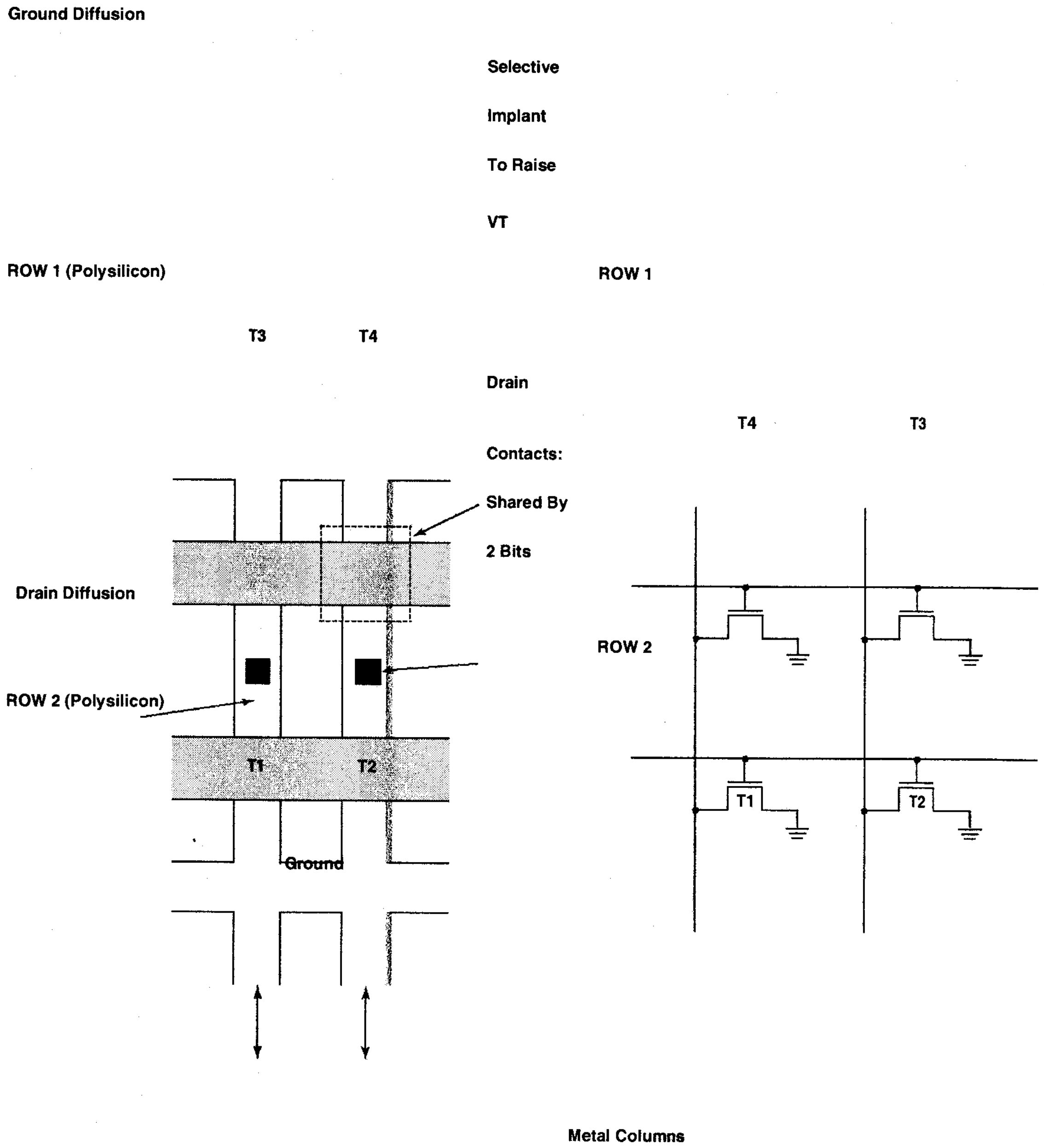
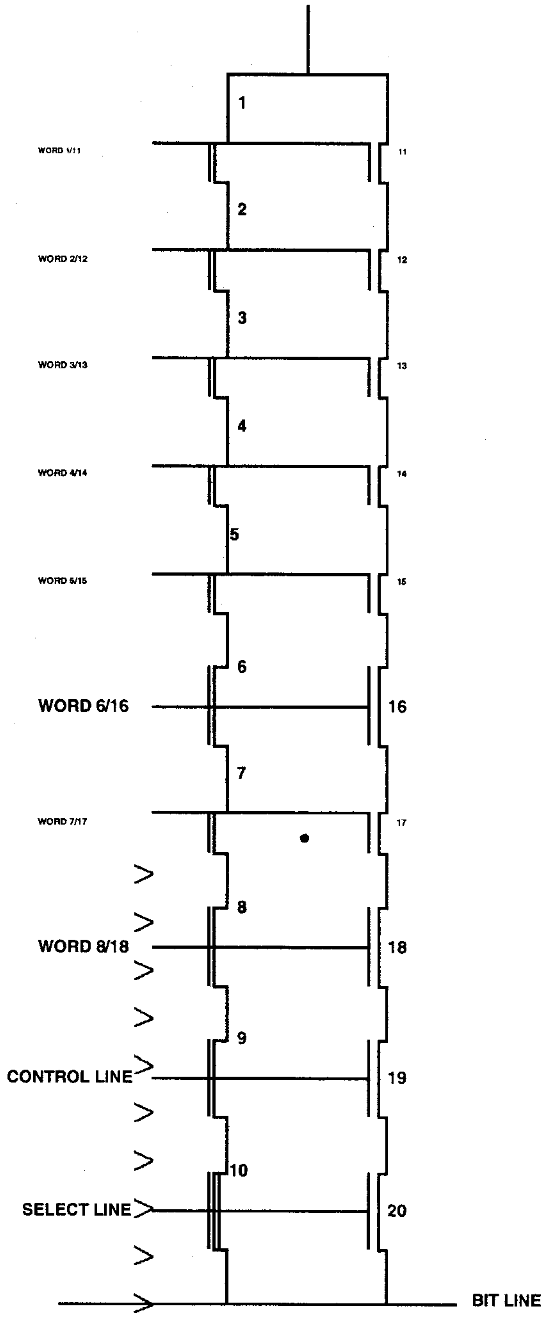


Figure ROM Programmed by Channel Implant



Source: ICE, "Memory 1997"

19050

Figure . Memory Cell Schematic

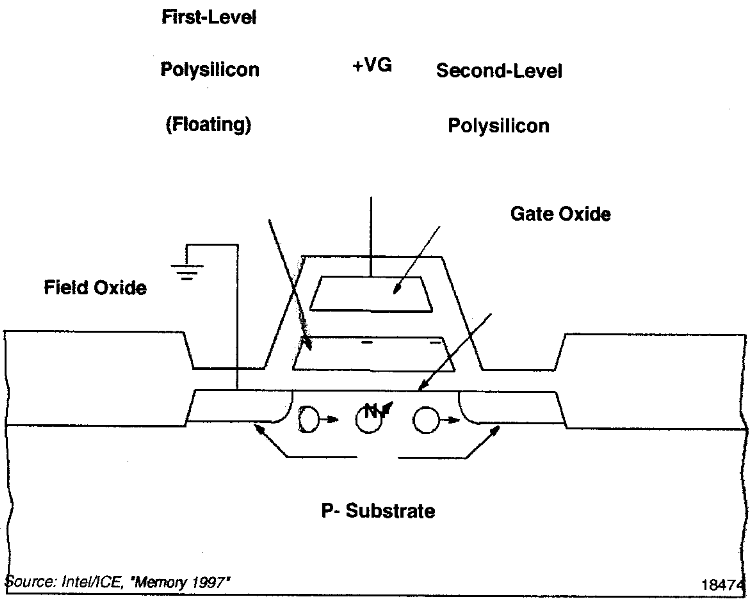
Multimedia Card

In 1996, Siemens announced the introductionof a new solid-state memory chip technology that enables the creation of a multimedia card that is sized37mm x 45mm x 1.4mm, or roughly 40 per-cent the size of a credit card. It is offered with either 16Mbit or 64Mbit of ROM.

EPROM

EPROM (UV Erasable Programmable Read Only Memory) is a special type of ROM that is pro-grammed electrically and yet is erasable under UV light.

The EPROM device is programmed by forcingan electrical charge on a small piece of polysilicon material (called the floating gate) located in the memory cell. When this charge is present on this gate, the cell is "programmed," usually a logic "0," and whenthis charge is not present, it is a logic "1." Figure 9-4 shows the cell used in a typical EPROM. The floatinggate is where the electrical charge is stored.



Source: Intel/ICE, "Memory 1997"

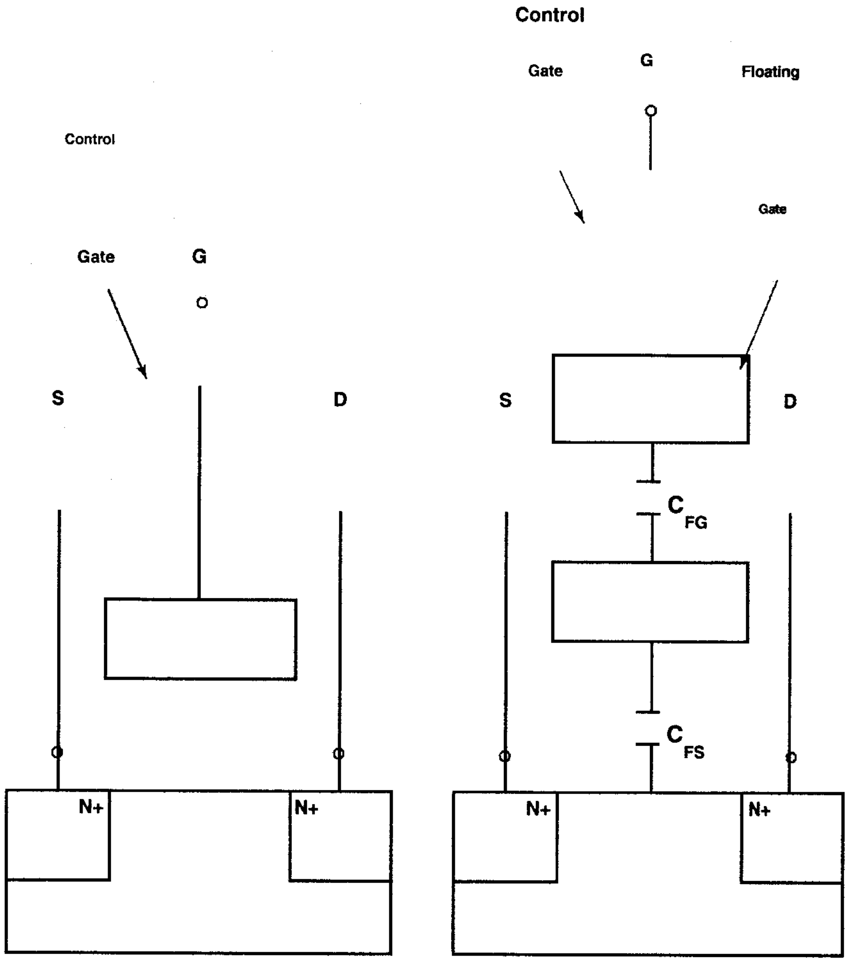
Figure . Double-Poly Structure (EPROM/Flash Memory Cell)

Prior to being programmed, an EPROM has to be erased. To erase the EPROM, it is exposed to an ultraviolet light for approximately 20 minutes through a quartz window in its ceramic package. After erasure, new information can be programmed to the EPROM. After writing the data to the EPROM, an opaque label has to be placed over the quartz window to prevent accidental erasure.

Programming is accomplished through a phenomenon called hot electron injection. High voltages are applied to the select gate and drain connections of the cell transistor. The select gate of the transistor is pulsed “on” causing a large drain current to flow. The large bias voltage on the gate connection attracts electrons that penetrate the thin gate oxide and are stored on the floating gate.

EPROM floating Gate Transistor Characteristic Theory

The following explanation of EPROM floating gate transistor characteristic theory also applies to EEPROM and flash devices. Figures 9-5 (a) and (b) show the cross section of a conventional MOS transistor and a floating gate transistor, respectively. The upper gate in Figure 9-5 (b) is the con-trol gate and the lower gate, completely isolated within the gate oxide, is the floating gate.



(a) Conventional MOS

(b) Floating-Gate MOS

Figure 9-5. Cross Section of a Conventional MOS Transistor and a Floating-Gate MOS Transistor

C_{FG} and C_{FS} are the capacitances between the floating gate and the control gate and substrate, respectively. V_G and V_F are the voltages of the control gate and the floating gate, respectively. $-Q_F$ is the charge in the floating gate. (As electrons have a negative charge, a negative sign was added). In an equilibrium state, the sum of the charges equals zero.

$$(V_G - V_F)\, C_{FG} + (0 - V_F)\, C_{FS} - Q_F = 0$$

$$V_F = \left(\frac{C_{FG}}{C_{FG} + C_{FS}}\right) V_G - \frac{Q_F}{C_{FG} + C_{FS}}$$

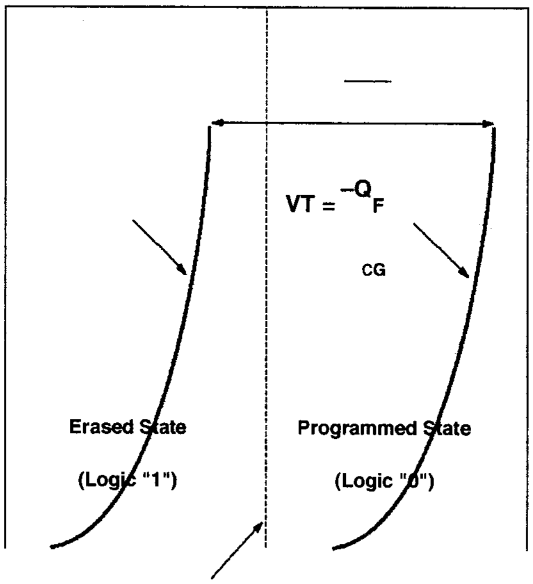
V_{TC} is the threshold voltage of the conventional transistor, and V_{TCG} is the threshold voltage of the floating gate transistor.

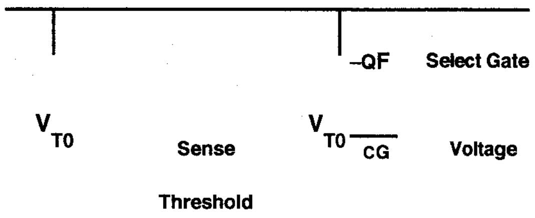
$$V_{TCG} = \left(\frac{C_{FG}}{C_{FG} + C_{FS}}\right) V_{TC} - \frac{Q_F}{C_{FG} + C_{FS}}$$

$$V_{TCG} = V_{TO} - \frac{Q_F}{C_G}$$

Where $V_{TO} = \left(\frac{C_{FG}}{C_{FG} + C_{FS}}\right) V_{TC}$ and $C_G = C_{FG} + C_{FS}$

The threshold voltage of the floating gate transistor (V_{TCG}) will be V_{TO} (around 1V) plus a term depending on the charge trapped in the floating gate. If no electrons are in the floating gate, then $V_{TCG} = V_{TO}$ (around 1V). If electrons have been trapped in the floating gate, then $V_{TCG} = V_{TO} - Q_F/C_G$ (around 8V for a 5V part). This voltage is process and design dependent. Figure 9-6 shows the threshold voltage shift of an EPROM cell before and after programming.





Source: ICE, 'Memory 1997'

17548A

Figure; Electrical Characteristics of an EPROM

The programming (write cycle) of an EPROM takes several hundred milliseconds. Usually a byte—eight bits—is addressed with each write cycle. The read time is comparable to that of fast ROMs and DRAMs (i.e., several tens of nanoseconds). In those applications where programs are stored in EPROMs, the CPU can run at normal speeds.

Field programmability is the EPROM's main advantage over the ROM. It allows the user to buy mass-produced devices and program each device for a specific need. This characteristic also makes the EPROM ideal for small-volume applications, as the devices are usually programmed in very small quantities. Also, the systems supplier can program any last minute upgrades to the program just before shipment. EPROM cells may be configured in the NAND structure shown previously, or, more commonly, in the NOR configuration shown in Figure 9-7.

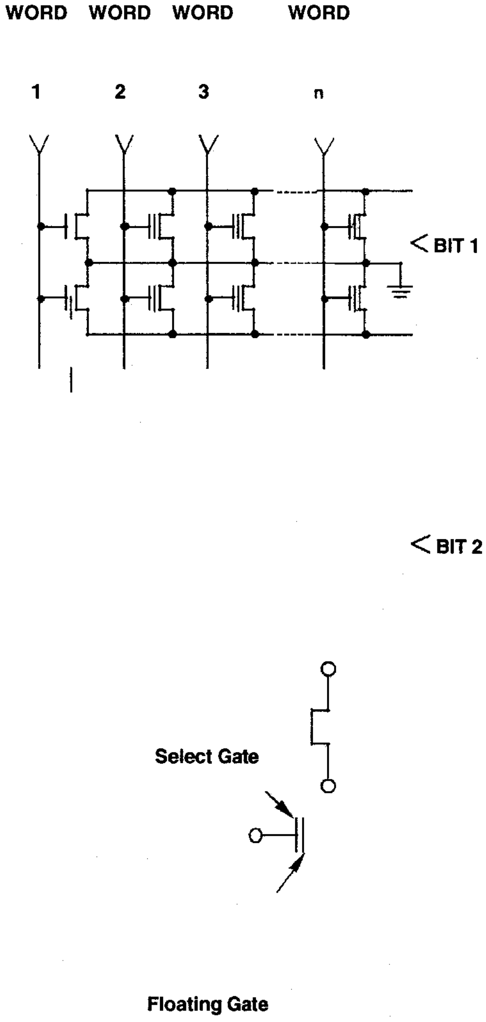


Figure . NOR EPROM Configuration

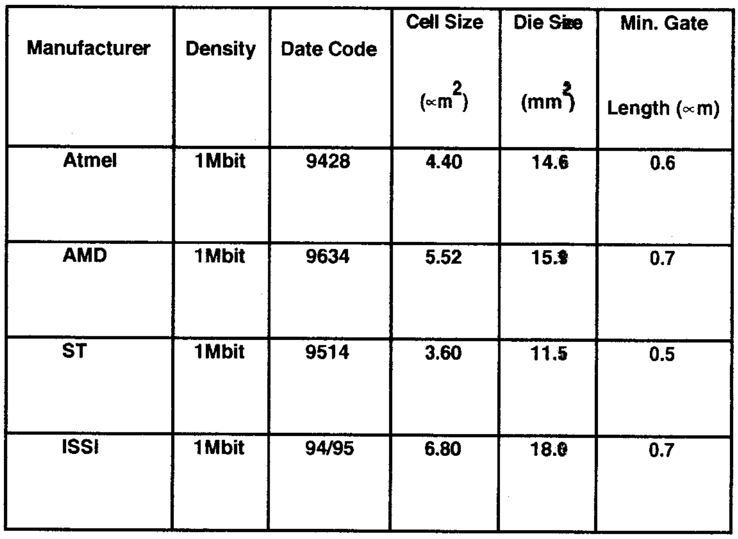
EPROMs were created in the 1970s and have long been the cornerstone of the non-volatile memory market. But the development of flash memory devices (see Section 10) will lead to a loss of EPROM marketshare. EPROM uses a mature technology and design and is on the decline part of its lifecycle. For this reason there is not a lot of R&D expenditure made for EPROM devices. Figure 9-8 shows a cross section of a 1Mbit EPROM cell from two different manufacturers. The main difference between the processes is the polysilicon gate. One manufacturer uses a polycide to improve the speed.

EPROM Cell Size and Die Size

The cell size of the EPROM is also relatively small. The EPROM requires one additional polysili-con layer, and will usually have slightly lower yields due to the requirement for nearly perfect (and thin) gate oxides.

Figure; Typical 1Mbit EPROM Cells

These factors, plus the fact that an EPROM is encased in a ceramic package with a quartz window, make the EPROM average selling price three to five times the price of the mask ROM. Figure 9-9 shows the main feature sizes of 1Mbit EPROM analyzed by ICEÕs laboratory.



Manufacturer	Density	Date Code	Cell Size (∝m^2)	Die Size (mm^2)	Min. Gate Length (∝m)
Atmel	1Mbit	9428	4.40	14.6	0.6
AMD	1Mbit	9634	5.52	15.9	0.7
ST	1Mbit	9514	3.60	11.5	0.5
ISSI	1Mbit	94/95	6.80	18.0	0.7

Source: ICE, "Memory 1997"

22453

Figure; EPROM Feature Sizes

OTP(One Time Programmable) EPROM

In most applications, EPROMs are programmed one time and will never have to be erased. To reduce the cost for these applications, EPROMs may be manufactured in opaque plastic packages since the standard ceramic package of an EPROM is expensive. EPROMs that are programmed one time for a specific use and cannot be erased are referred to as One Time Programmable (OTP) devices.

EEPROM

EEPROM (Electrically Erasable Programmable ROM) offer users excellent capabilities and per-formance. Only one external power supply is required since the high voltage for program/erase is internally generated. Write and erase operations are performed on a byte per byte basis.

The EEPROM uses the same principle as the UV-EPROM. Electrons trapped in a floating gate will modify the characteristics of the cell, and so a logic "0" or a logic "1" will be stored.

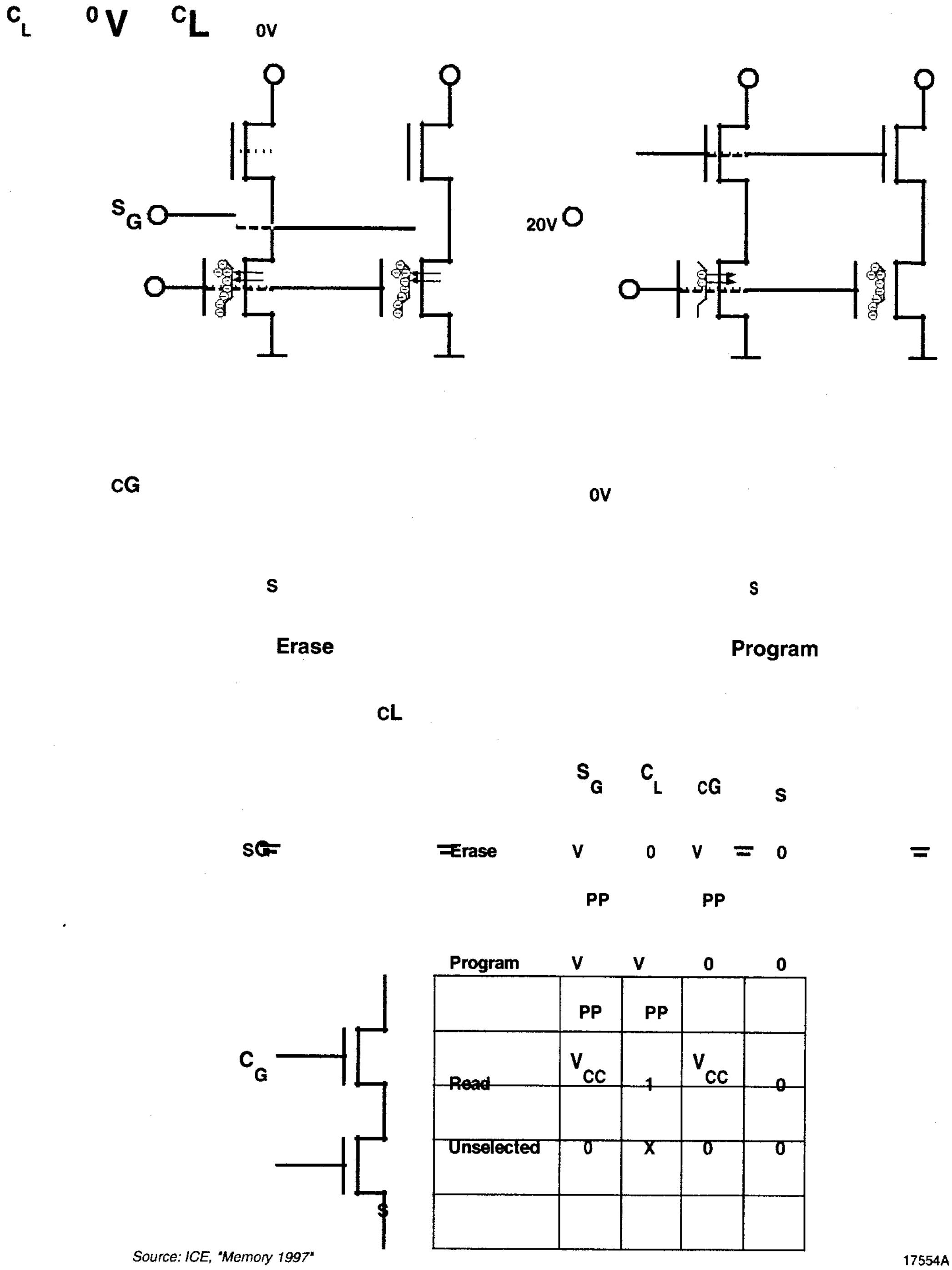
The EEPROM is the memory device that implements the fewest standards in cell design. The more common cell is composed of two transistors. Thestorage transistor has a floating gate (sim-ilar to the EPROM storage transistor) that will trap electrons. In addition, there is an access tran-sistor, which is required for operations. Figure 9-10 shows the voltages applied on the memory cell to program/erase a cell. Note that an EPROM cell is erased when electrons are removed from the floating gate and that the EEPROM cell is erased when the electrons are trapped in the float-ing cell. To have products electrically compatible, the logic path of both types of product will give a "1" for erase state and a "0" for a programmed state. Figure 9-11 shows the electricaldifferences between EPROM and EEPROM cells.

Parallel EEPROM

There are two distinct EEPROM families: serial and parallel access. The serial access represents 90 percent of the overall EEPROM market, and parallel EEPROMs about 10 percent. Parallel devices are available in higher densities (≥256Kbit), are generally faster, offer high endurance and reliability, and are found mostly in the military market. They are pin compatible with EPROMs and flash memory devices. Figure 9-12 shows feature sizes of three 1Mbit parallel EEPROM from different manufacturers, analyzed by ICE's laboratory. Figures 9-13 to 9-15 show photographs and schematics of the respective cells. It is interesting to see the wide differences in these cells.

Serial EEPROM

Serial EEPROMs are less dense (typically from 256 bit to 256Kbit) and are slower than parallel devices. They are much cheaper and used in more "commodity" applications.

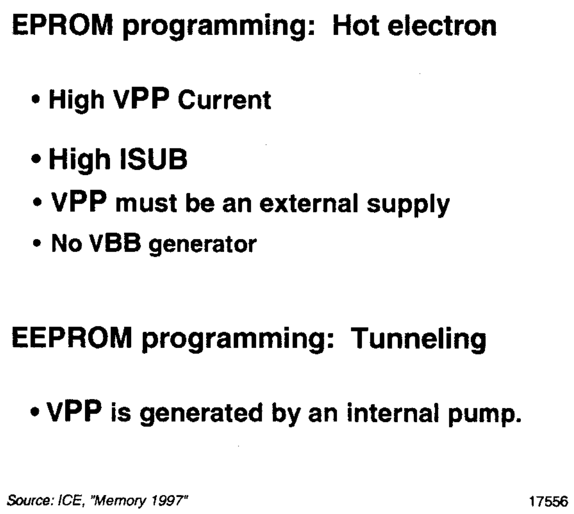


	S_G	C_L	CG	S
Erase	V_{PP}	0	V_{PP}	0
Program	V_{PP}	V_{PP}	0	0
Read	V_{CC}	1	V_{CC}	0
Unselected	0	X	0	0

Source: ICE, "Memory 1997"

17554A

Figure ;. EEPROM Cell Program/Erase



EPROM programming: Hot electron

- High VPP Current
- High ISUB
- VPP must be an external supply
- No VBB generator

EEPROM programming: Tunneling

- VPP is generated by an internal pump.

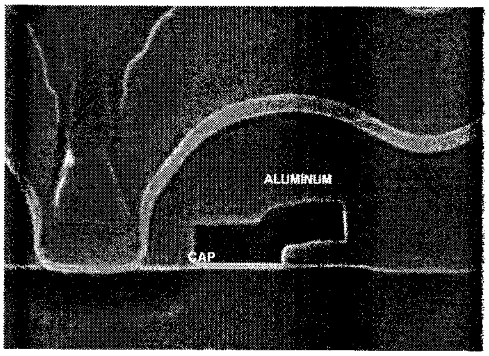
Source: ICE, "Memory 1997" 17556

Figure ;. Vpp EPROM Versus Vpp EEPROM

Manufacturer	Density	Date Code	Cell Size $(\propto m^{2})$	Die Size (mm^{2})	Min Gate Length $(\propto m)$
Winbond	1Mbit	9432	7.8	22.6	0.9
Xicor	1Mbit	9443	21.0	51.0	1.3
Hitachi	1Mbit	94/95	22.5	51.0	0.6

Source: ICE,"Memory 1997" 22463

Figure ; 1Mbit Parallel EEPROM Feature Sizes



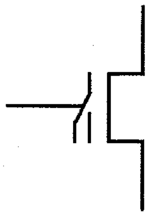


Figure ; Winbond 1Mbit EEPROM Cell

Serial access EEPROMs feature low pin count.Typically they are packaged in an 8-pin package. As illustrated in Figure 9-16, XicorÕs 128Kbit serial EEPROM uses the 8 pins in the following manner:

¥ V_{CC} and V_{SS} for supply voltage
¥ SCL (Serial Clock) to clock the data

¥ SDA (Serial Data) is a bi-directional pinused to transfer data into and out of the device

¥ S0, S1, S2 are select inputs used to setthe first three bits of the 8-bit slave address

¥ WP (Write Protection) controls Write Protection features.

Serial EEPROMs use data transfer interface protocols for embedded control applications. These protocols include the Microwave bus, the I^{2}C bus, theXI^{2}C (Extended I^{2}C) or the SPI (Serial Peripheral Interface) bus interfaces.

There continues to be an ongoing effort to reduce the size of serial EEPROMs. Microchip Technology, for example, introduced a 128bit serial EEPROM in a five-lead SOT-23 package.

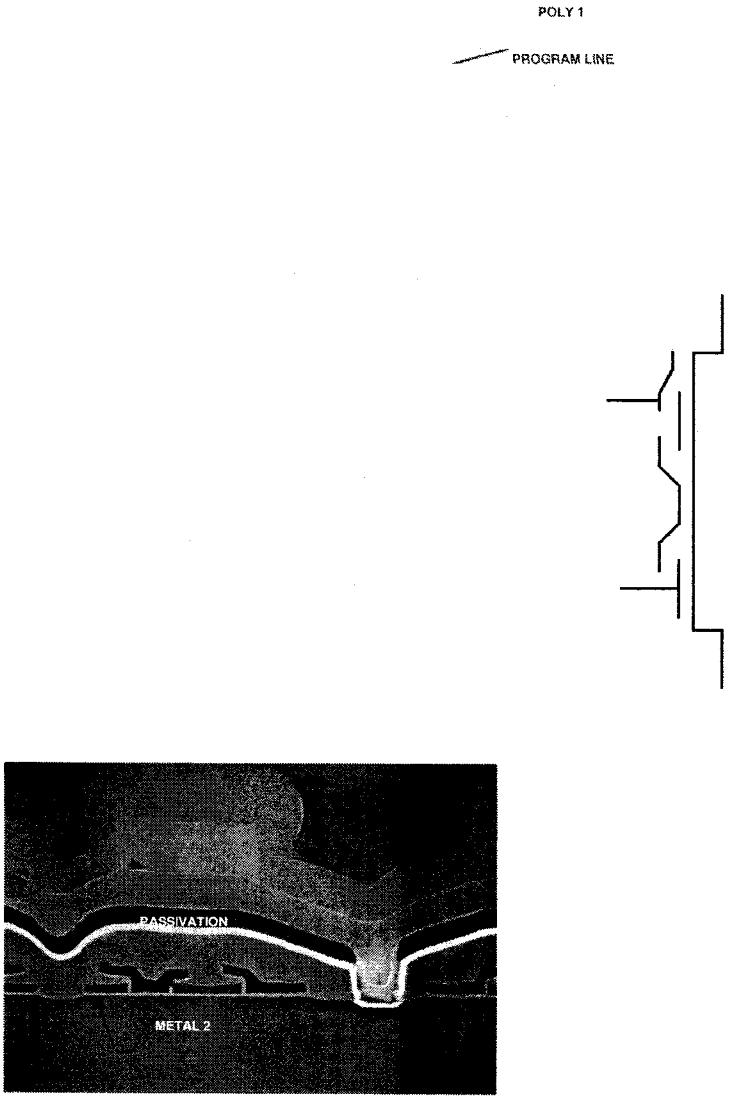


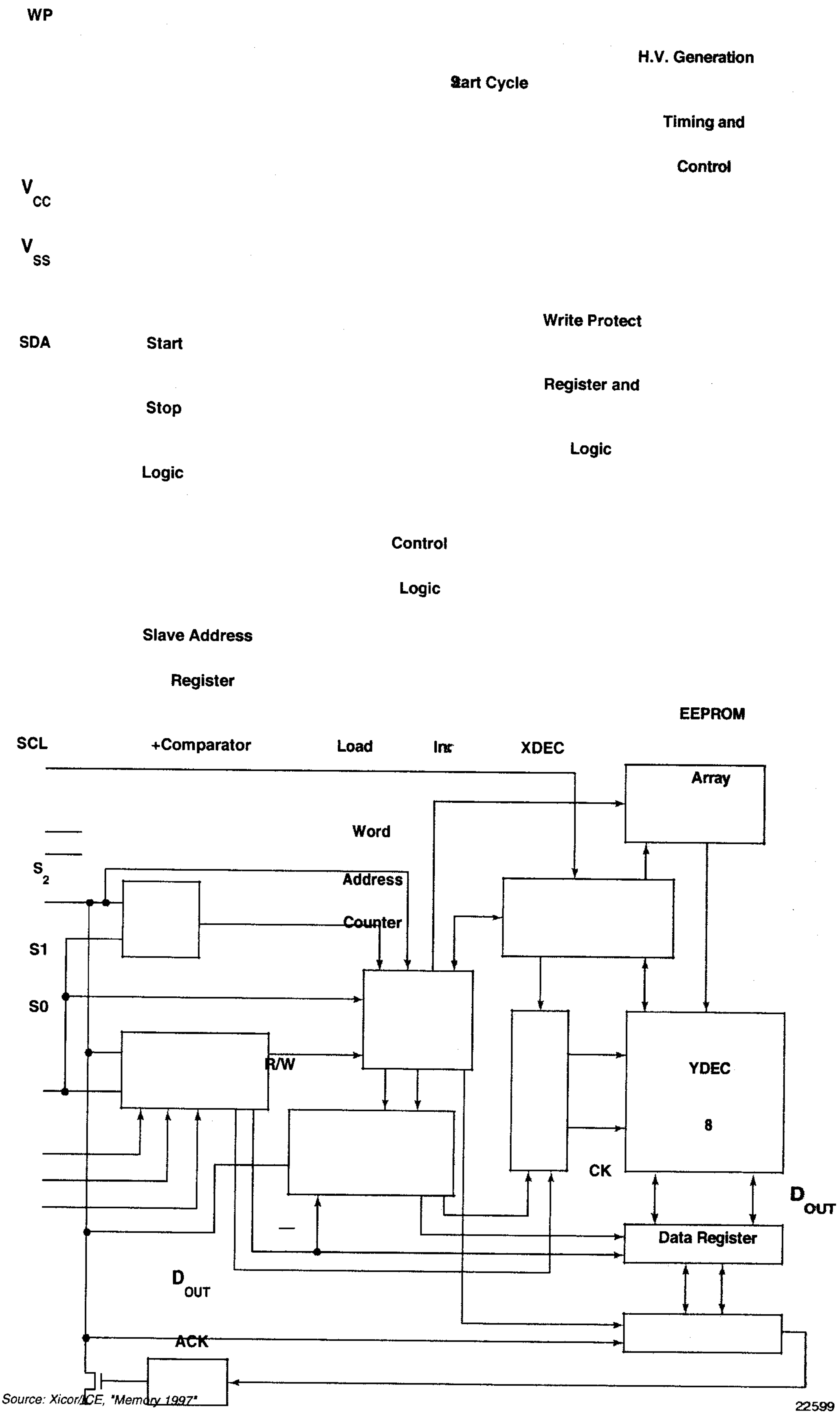
Figure . Xicor 1Mbit EEPROM Cell

Silicon Nitride

Source: ICE, "Memory 1997" 22467

Figure . Hitachi1Mbit EEPROM Cell

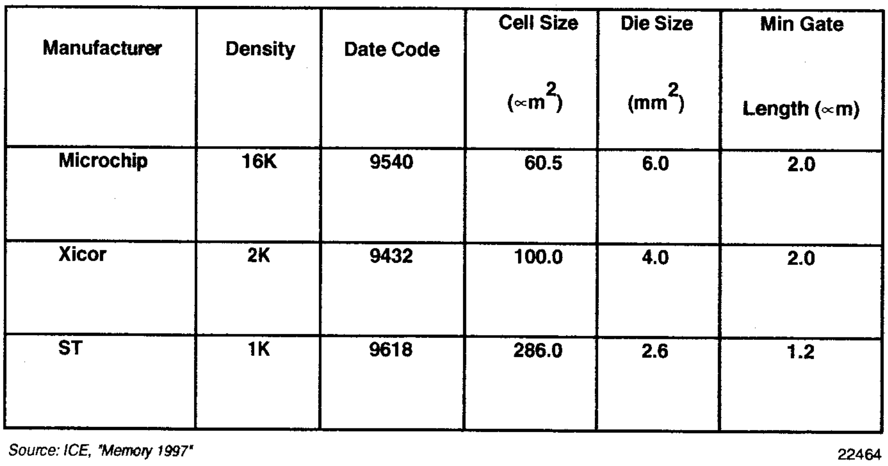
Figure 9-17 shows feature sizes of three serial EEPROMs from different manufacturers that were analyzed by ICEÕs laboratory. Note that larger cell sizes accompany low-density EEPROM devices. When building an EEPROM chip that contains sense amplifiers, controllers, and other peripheral circuitry, cell size is not as great a factor at low (1Kbit, 2Kbit) densities. At larger den-sities, the size of the cell array is more critical. It becomes a larger portion of the chip. Therefore, greater consideration must be given to the size of the cell.



Source: Xicor/ICE, "Memory 1997"

22599

Figure. Xicor 128Kbit Serial EEPROM Functional Diagram



Manufacturer	Density	Date Code	Cell Size (∝m^2)	Die Size (mm^2)	Min Gate Length (∝m)
Microchip	16K	9540	60.5	6.0	2.0
Xicor	2K	9432	100.0	4.0	2.0
ST	1K	9618	286.0	2.6	1.2

Source: ICE, "Memory 1997"

22464

Figure . EEPROM Serial Configuration Feature Sizes

This size impact is illustrated in Figure using a 1Kbit serial EEPROM example from SGS-Thomson. The cell array represents only 11 percent of the total surface of the chip.

Figures show additional EEPROM cells.As noted, there is no design standard for this type of cell. In laying out the EEPROM cell, the designer must take into consideration the ele-ments of size, performance, and process complexity.

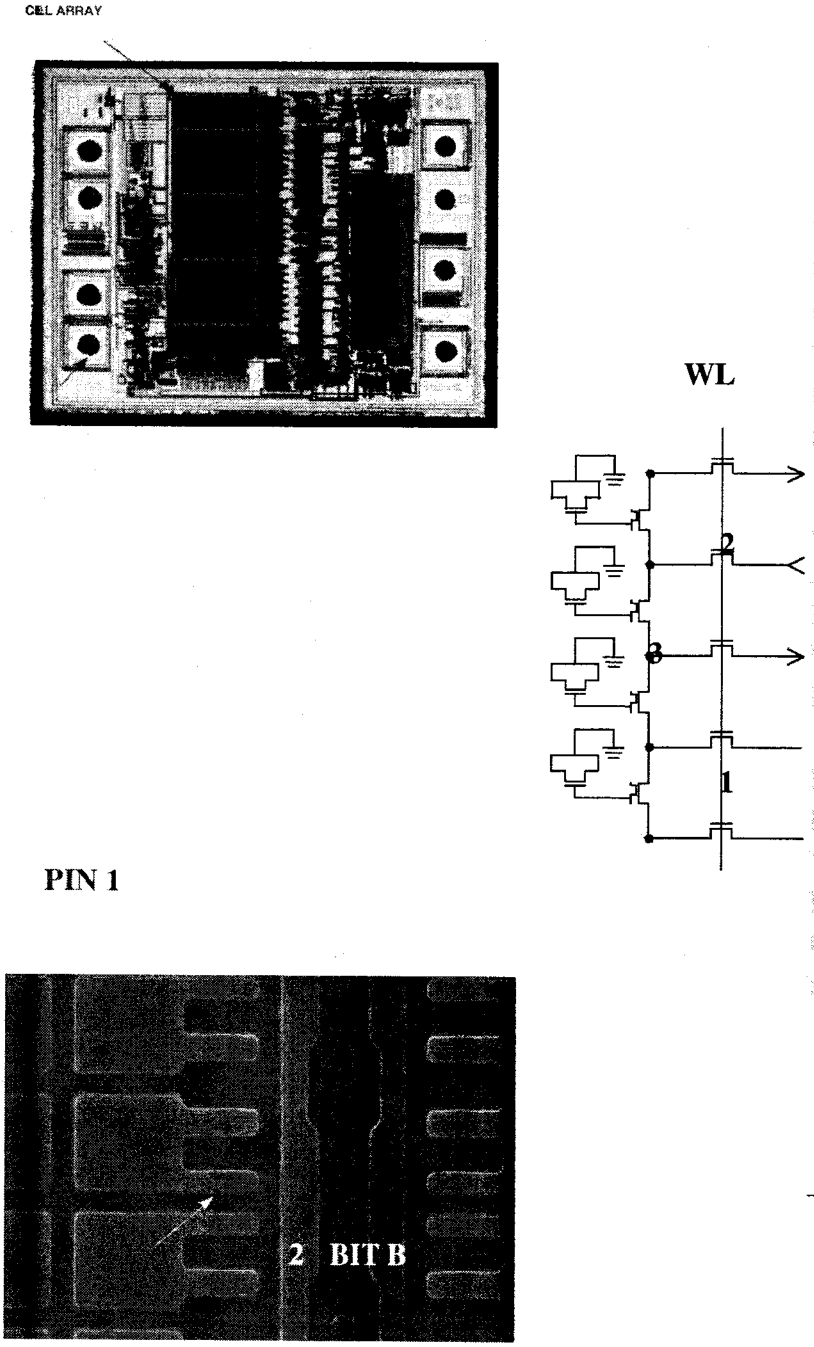
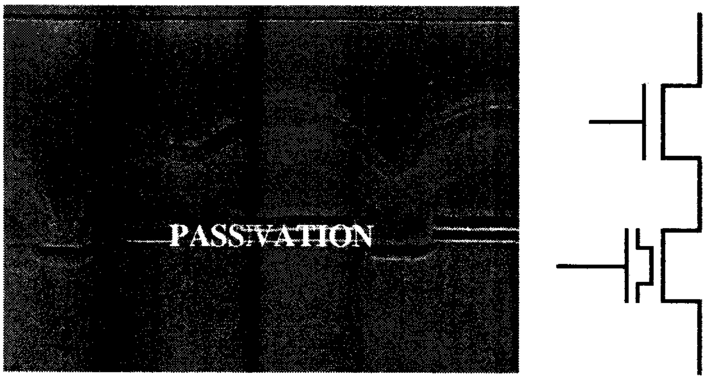


Figure . SGS-Thomson 1Kbit Serial EEPROM



Figure; Microchip 16Kbit Serial EEPROM Cell

Multi-Level Analog Storage EEPROM

The goal of multi-level cell (MLC) is to store more than one bit of information in a single cell. Much work has already been done regarding MLC as applied to flash memory devices. The typ-ical development for digital flash memories is to store four different levels in the same cell, and thus divide the number of cells by two (four data are given by two bits : 00, 01, 10, and 11).

However, for several years now, Information Storage Devices (ISD), a San Jose based company, has proposed multi-level analog storage EEPROMs for analog storage. ISD presented a 480Kbit EEPROM at the 1996 ISSCC conference. The multi-level storage cell is able to store 256 different levels of charge between 0V and 2V. This means the cell needs to have a 7.5mV resolution. The 256 different levels in one cell corresponds to eight bits of information. A comparable digital implementation requires 3.84Mbit memory elements to store the same amount of information. The information stored will not be 100 percent accurate but is good enough for audio applications, which allows some errors.

Course Material Prepared by
Dr. S. MEENAKSHI SUNDARAM
HOD of Physics, Sri Paramakalyani College
Alwarkurichi - 627 412.