Digital Electronics
EE(4th Sem B.Tech)

Module-1

Fundamentals of Digital Systems and logic families

ANALOG SIGNALS

We are very familiar with analog signals. The readings of a moving coil or moving iron voltmeter and ammeter, dynamometer wattmeter etc., are all analog quantities. The trace on a CRO screen is also analog. Analog methods for communication system have long been in use. Frequency division multiplexing is the means of analog communication. An electronic amplifier is an analog circuit. The low level analog signal (audio, video, etc.) is amplified to provide strength to the signal. Analog circuit systems (position control, process control) have been in use for the past many decades. Analog Computers use voltages, resistances and potentiometric rotations to represent the numbers and perform arithmetic operations. Analog differentiation, integration, etc., is also done. Operational amplifier is a very versatile analog electronic circuit used to perform a variety of operations (addition, subtraction, multiplication, division, exponentiation, differentiation, integration etc.). Analog integrated circuits are widely used in electronic industry.

DIGITAL SIGNALS

The term digital is derived from digits. Any device or system which works on digits is a digital device or system. A digital voltmeter indicates the value of voltage in the form of digits, e.g., 230.25. Reading an analog instrument introduces human error and also requires more time. A digital reading is more accurate, eliminates human error and can be read quickly. Communication systems have also gone digital. The initial signal waveform is always analog. To use digital transmission, the signal waveform is sampled and the digital representation transmitted.

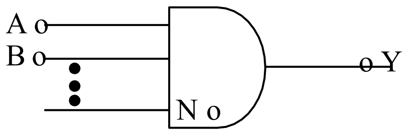
BASIC DIGITAL CIRCUITS

In a digital system there are only a few basic operations performed, irrespective of the complexities of the system. These operations may be required to be performed a number of times in a large digital system like digital computer or a digital control system, etc. The basic operations are AND, OR, NOT, and FLIP-FLOP. The AND, OR, and NOT operations are discussed here and the FLIP-FLOP, which is a basic memory element used to store binary information (one bit is stored in one FLIP-FLOP).

The AND Operation

A circuit which performs an AND operation is shown in Fig. It has N inputs (N = 2) and one output. Digital signals are applied at the input terminals marked A, B, …, N, the other terminal being ground, which is not shown in the diagram. The output is obtained at the output terminal marked Y (the other terminal being ground) and it is also a digital signal. The AND operation is defined as : the output is 1 if and only if all the inputs are 1. Mathematically, it is written as

$$
\begin{aligned}
Y &= A \text{ AND } B \text{ AND } C \ldots \text{ AND } N \\
&= A \cdot B \cdot C \cdot \ldots \cdot N \\
&= ABC \ \ldots N
\end{aligned}
$$

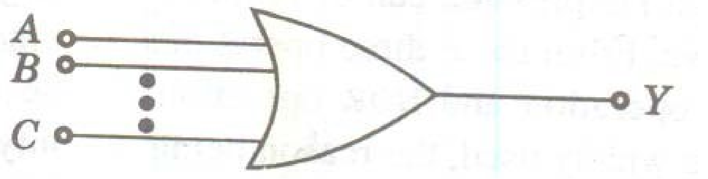


Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The OR Operation

Figure below shows an OR gate with N inputs (N ≥ 2) and one output. The OR operation is defined as: the output of an OR gate is 1 if and only if one or more inputs are 1. Its logical equation is given by

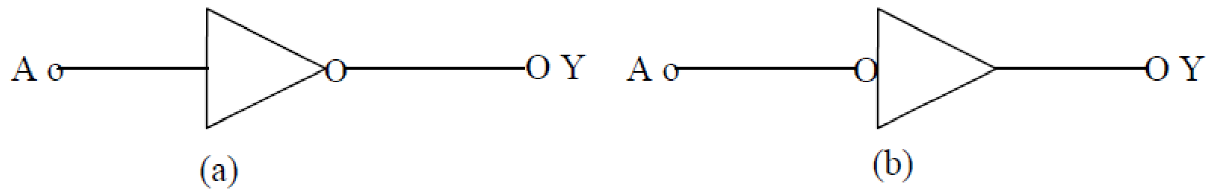
$$Y = A \text{ OR } B \text{ OR } C \ldots \text{ OR } \quad N = A + B + C + \ldots + N$$



Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The NOT Operation

Figure below shows a NOT gate, which is also known as an *inverter*. It has one input (A) and one output (Y). Its logic equation is written as



$$Y = \text{NOT } A$$
$$= \overline{A}$$

Input	Output
A	Y
0	1
1	0

The NAND Operation

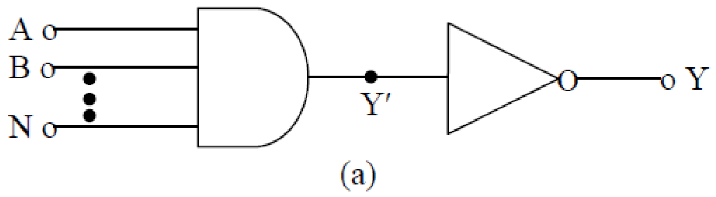
The NOT-AND operation is known as the NAND operation. Figure shows and N input (N ≥ 2) AND gate followed by a NOT gate. The operation of this circuit can be described in the following way:

The output of the AND gate (Y′) can be written using Eq.

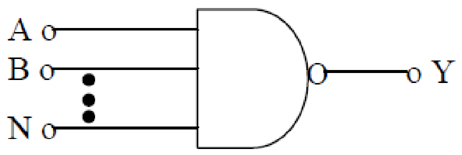
$$Y' = AB \ldots N$$

Now, the output of the NOT gate (Y) can be written using Eq.

$$Y = Y' = (AB...N)$$



(a)



(b)

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The NOR Operation

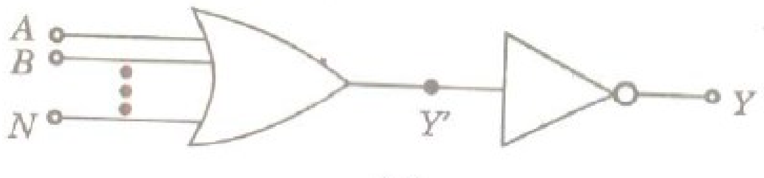
The NOT-OR operation is known as the NOR operation. Figure below shows an N input (N ≥ 2) OR gate followed by a NOT gate. The operation of this circuit can be described in the following way:

The output of the OR gate Y′ can be written using Eq. as

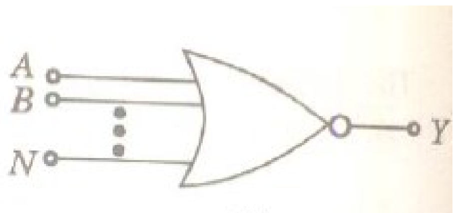
$$Y'. = A + B + \dots + N$$

and the output of the NOT gate (Y) can be written using Eq.

$$Y = Y' = \overline{A + B + \dots + N}$$



(a)

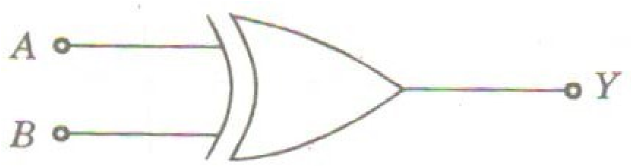


(b)

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

EXCLUSIVE–OR OPERATION

The EXCLUSIVE–OR (EX–OR) operation is widely used in digital circuits. It is not a basic operation and can be performed using the basic gates–AND, OR and NOT or universal gates NAND or NOR. Because of its importance, the standard symbol shown below is used for this operation.



Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

BOOLEAN ALGEBRA

Commutative Law

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

The above two equations mean that inputs can be interchanged in OR gate and AND gate.

Associative Law

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

$$A. (B. C) = (A. B) C$$

Distributive Law

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

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

DE MORGAN'S THEOREMS

First Theorem : DE Morgan's first theorem is

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

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

Second Theorem : De Morgan's second theorem can be written as

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

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

DECIMAL NUMBER SYSTEM

We are all familiar with the decimal number system. It uses ten digits (0, 1, 2, 3, 4, 5, 6, 7, 9) and thus its base is 10. The decimal number system of counting was evolved because we have 8 fingers and 2 thumbs on our two hands so that we can count 10. By using the different digits in different positions we can express any number. For numbers bigger than 9 we use two or more digits. The position of each digit in the number indicates the magnitude that this number represents. In the number 27 the digit 7 represents $7 \times 10°$ or 7 and the digit 2 represents 2×10^1 or 20. The sum of 7 and 20 makes 27. Similarly the number 263 can be expressed as

Decimal 263 $= (2 \times 10^2) + (6 \times 10^1) + (3 \times 10^0) = 200 + 60 + 3 = 263$.Since the base in decimal number system is 10, the number 263 can be written as 263_{10}. The suffix 10 emphasizes the fact that the base is 10.

BINARY NUMBER SYSTEM

The binary number system has only two digits 0 and 1. Thus a binary number is a string of zeros and ones. Since it has only two digits, the base is 2. The abbreviation of binary digit is bit. The binary number 1100 has 4 bits, 101011 has 6 bits and 11001010 has 8 bits. Each bit may represent either 0 or 1. A string of 8 bits is known as a byte. A byte is the basic unit of data in computers. In most computers, the data is processed in strings of 8 bits or some multiples (i.e., 16, 24, 32 etc.). The computer memory also stores data in strings of 8 bits or multiples of 8 bits.

Binary to Decimal Conversion

The procedure to convert a binary number to decimal start with the left hand bit. Multiply this value by 2 and add the next bit. Again multiply by 2 and add the next bit. Stop when the bit on extreme right hand side is reached. Another fast and easy method to convert binary number to decimal number is as under:

1. Write the binary number.
2. Write the weights 2^0, 2^1, 2^2, 2^3 etc., under the binary digits starting with the bit on right hand side.
3. Cross out weights under zeros.
4. Add the remaining weights.

Example Convert 100101_2 to decimal.

Solution : Left hand bit 1

Multiply by 2 and add next bit $2 \times 1 + 0 = 2$

Multiply by 2 and add next bit $2 \times 2 + 0 = 4$

Multiply by 2 and add next bit $2 \times 4 + 1 = 9$

Multiply by 2 and add next bit $2 \times 9 + 0 = 18$

Multiply by 2 and add next bit $2 \times 18 + 1 = 37$

Therefore, $100101_2 = 37_{10}$

Decimal to Binary Conversion

A systematic way to convert a decimal number into equivalent binary number is known as double dabble. This method involves successive division by 2 and recording the remainder (the remainder will be always 0 or 1). The division is stopped when we get a quotient of 0 with a remainder of 1. The remainders when read upwards give the equivalent binary number.

Example 2.4 Convert decimal number 10 into its equivalent binary number.

```
2 | 10
2 | 5    remainder 0
2 | 2    remainder 1
2 | 1    remainder 0
  0      remainder 1
```

The binary number is 1010.

SIGNED BINARY NUMBERS

To represent negative numbers in the binary system, digit 0 is used for the + sign and 1 for the −ve sign. The most significant bit is the sign bit followed by the magnitude bits. Numbers expressed in this manner are known as signed binary numbers.The numbers may be written in 4 bits, 8 bits, 16 bits, etc. In every case, the leading bit represents the sign and the remaining bits represent the magnitude.

Example Express in 16−bit signed binary system : (a) + 8, (b) −8, (c) 165, (d) −165.

Solution : (a)

```
2 | 8
2 | 4    remainder 0
2 | 2    remainder 0
2 | 1    remainder 0
  | 0    remainder 1
```

The binary number is 1000.

For the 16 bit system, we use 16 bits, 0 (which stands for +) in the leading position, 1000 in the last 4 bits and 0 in the remaining 11 positions. So the signed 16 bit binary number is

+ 8 = 0000 0000 0000 1000

(b) In the leading bit we will have 1 (to represent the '–' sign). The rest of the representation is the same s in part (a).

1'S COMPLEMENT

The 1's complement of a binary number is obtained by complementing each bit (i.e., 0 for 1 and 1 for 0).

Thus each bit in the original word is inverted to give the 1's complement. For example, for the number

	1 1 0 0	1 0 0 1
1's complement is	0 0 1 1	0 1 1 0

2'S COMPLEMENT

The signed binary numbers required too much electronic circuitry for addition and subtraction. Therefore, positive decimal numbers are expressed in sign–magnitude form but negative decimal numbers are expressed in 2's complements.

2's complement is defined as the new word obtained by adding 1 to 1's complement*

e.g., Let A = 0 1 0 1 i.e., 5

1′s complement of A is denoted by A and 2′s complement of A is denoted by A′.

2'S COMPLEMENT ADDITION, SUBTRACTION

The use of 2's complement representation has simplified the computer hardware* for arithmetic operations. When A and B are to be added, the B bits are not inverted so that we get

S = A + B

When B is to be subtracted from A, the computer hardware forms the 2's complement of B and then adds it to A. Thus

$$S = A + B' = A + (-B) = A - B$$

Eqns. (2.1 and 2.2) represent algebraic addition and subtraction. A and B may represent either positive or negative numbers. Moreover, the final carry has no significance and is not used.

Conversion of Binary to Decimal

In the binary system, the weights of the binary bits after the binary point, can be written as

$$0.1011 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$$

$$= 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8} + 1 \times \frac{1}{16}$$

$$= 0.5 + 0 + 0.125 + 0.0625 = 0.6875 \text{ (decimal)}$$

Example Express the number 0.6875 into binary equivalent

Solution :

Fraction	Fraction × 2	Remainder new fraction	Integer
0.6875	1.375	0.375	1 (MSB)
0.375	0.75	0.75	0
0.75	1.5	0.5	1
0.5	1	0	1(LSB)

The binary equivalent is 0.1011.

OCTAL NUMEBR SYSTEM

The number system with base (or radix) eight is known as the octal number system. In this system, eight symbols, 0, 1, 2, 3, 4, 5, 6 and 7 are used to represent numbers. Similar to decimal and binary number systems, it is also a positional system and has, in general, two parts: integer and fractional, set apart by a radix (octal) point (·). For example, $(6327.4051)_8$ is an octal number. Using the weights it can be written as

$$(6327.4051)_8 = 6 \times 8^3 + 3 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + .4 \times 8^{-1}$$
$$+ 0 \times 8^{-2} + 5 \times 8^{-3} + 1 \times 8^{-4}$$

$$= 3072 + 192 + 16 + 7 +$$

$$= (3287.5100098)_{10}$$

Thus, $(6327.4051)_8 = (3287.5100098)_{10}$

HEXADECIMAL NUMBER SYSTEM

Hexadecimal number system is very popular in computer uses. The base for hexadecimal number system is 16 which requires 16 distinct symbols to represent the numbers. These are numerals 0 through 9 and alphabets A through F. Since numeric digits and alphabets both are used to represent the digits in the hexadecimal number system, therefore, this is an alphanumeric number system. Table 2.3 gives hexadecimal numbers with their binary equivalents for decimal numbers 0 through 15. From the table, it is observed that there are 16 combinations of 4-bit binary numbers and sets of 4-bit binary numbers can be entered in the computer in the form of hexadecimal (hex.) digits. These numbers are required to be converted into binary representation, using hexadecimal-to-binary converter circuits before these can be processed by the digital circuits.

Decimal-to-Hexadecimal Conversion

The conversion from decimal to hexadecimal, the procedure used in binary as well as octal systems is applicable, using 16 as the dividing (for integer part) and multiplying (for fractional part) factor.

Hexadecimal-to-Binary Conversion

Hexadecimal numbers can be converted into equivalent binary numbers by replacing each hex digit by its equivalent 4–bit binary number.

Example Convert $(2F9A)_{16}$ to equivalent binary number.

Solution : Using Table 2.7, find the binary equivalent of each hex digit. $(2F9A)_{16} = (0010\ 1111\ 1001\ 1010)_2$

$= (0010111110011010)_2$

Binary-to-Hexadecimal Conversion

Binary number can be converted into the equivalent hexadecimal numbers by making groups of four bits starting from LSB and moving towards MSB for integer part and then replacing each group of four bits by its hexadecimal representation.

For the fractional part, and above procedure is repeated starting from the bit next to the binary point and moving towards the right.

CODES

Computers and other digital circuits process data in the binary format. Various binary codes are used to represent data which may be numeric, alphabets or special characters. Although, in every code used the information is represented in binary form, the interpretation of this binary information is possible only if the code in which this information is available is known. For example, the binary number 1000001 represents 65 (decimal) in straight binary, 41 (decimal) in BCD and alphabet A in ASCII code. A user must be very careful about the code being used while interpreting information available in the binary format. Codes are also used for error detection and error correction in digital systems.

BINARY CODED DECIMAL (BCD)

Computers work with binary numbers. We work with decimal numbers. A code is needed to represent decimal numbers and binary numbers. A weighted binary code is one in which each number carries a certain weight. A string of 4 bits is known as nibble. Binary coded decimal (BCD) means that each decimal digit is represented by a nibble (binary code of 4 digits). Main BCD codes have been proposed, e.g., 8421, 2421, 5211, X53. Out of these 8421 code is the most predominant BCD code. The designation 8421indicates the weights of the 4 bits (8, 4, 2 and 1 respectively starting from the left most bit). When one refers to a BCD code, it always means 8421 code. Though 16 numbers (2^4) can be represented by 4 bits, only 10 of these are used. Table 2.4 shows the BCD code. The remaining 6 combinations, i.e., 1010, 1011, 1100, 1101, 1110 and 1111 are invalid in 8421 BCD code. To express any number in BCD code, each decimal number is replaced by the appropriate four bit code.

BCD ADDITION

Addition is the most important arithmetic operation. Subtraction, multiplication and division can be done by using addition. The rules for BCD addition are :

1. Add the two numbers using binary addition (section 2.5). If the four bit sum is equal or less than 9(i.e., equal to or less than 1001) it is a valid BCD number.
2. If the four bit sum is more than 9 or a carry is generated from the group of 4 bits, the result is invalid. In such a case add 6(i.e., 0110) to the four bit sum to

skip the 6 invalid states. If a carry is generated when adding 6, add the carry to the next four bit group.

GRAY CODE

It is an unweighted code. The bit positions do not have any specific weights assigned to them. However, the most important characteristic of this code is that only a signal bit change occurs when going from one code number to next. (In binary systems all the 4 bits change when we go from 0111 to 1000. i.e., 7_{10} to 8_{10}). The single bit change property is important in some applications, e.g., shaft position encoders. In these applications the chances of error increase if more than one bit change occurs.

Binary to Gray Conversion

The rules for changing binary number into equivalent Gray code are :

1. The left most bit (most significant bit) in Gray code is the same as the left most bit in binary

```
1   0   1   1   Binary
↓
1               Gray
```

2. Add the left most bit to the adjacent bit

```
1  +  0   1   1
1     1
```

3. Add the next adjacent pair

```
1  0  +  1   1
1  1  1      0
```

4. Add the next adjacent pair and discard carry

```
1  0  1   +  1
1  1  1      0
```

5. Continue the above process till completion.

Gray to Binary Conversion

the method to convert from Gray code to binary is an under:

1. Left most bit in binary is the same as the left most bit in Gray

```
code. 1 1   0   1   1   Gray
↓
```

1 Binary

2. Add the binary MSB to the Gray digit in the adjacent position. Discard

carry 1 1 0 1 1 Gray

↓ ↗

1 0 Binary

3. Add the binary digit generated in step 2 to the next Gray digit. Discard

carry 1 1 0 1 1 Gray

↓ ↗

1 0 0 Binary

4. Continue the above process till all the digits are covered. Discard carry in each case

1 1 0 ↗ 1 1 Gray

↓

1 0 0 1 0 Binar
y

2.10.2 EXCESS 3 CODE

Excess 3 is a digital code obtained by adding 3 to each decimal digit and then converting the result to four bit binary. It is an unweighted code, i.e., no weights can be assigned to any of the four digit positions.

Example Convert the following decimal numbers to excess 3 code (a) 14 (b) 32 (c) 46 (d) 430.

Solution : In each case add 3 to each digit in decimal number and then convert into binary. (a) 1 4 (b) 3 2

+3 + 3 +3 + 3

4 7 6 5

↓ ↓ ↓ ↓

0100 0111 0110 1010

(c) 4 6 (d) 4 3 0

+3 +3 +3 + 3 +3

7 9 7 6 3

↓ ↓ ↓ ↓ ↓

0111 1001 0111 0110 001
1

ERROR DETECTION CODES

Every digit of a digital system must be correct. An error in any digit can cause a problem because the computer may recognize it as something else. The correct ASCII code for A is 1000001. An error in one bit (i.e., 1000011) would mean C. Many methods have been devised to detect such errors.

Digital logic families

The design of a logic system starts with the statement of logic problems. This problem is then translated into a truth table and then into a logic equation. The logic equation yields the logical blocks and their interconnection so that we get the desired output for the given input conditions. From the logical block we realize an actual digital circuit. The basic building block in a digital system is logic gate logic circuits have evolved into families each of which has its own advantages and disadvantages. Generally a digital system is designed with circuits from one family only.

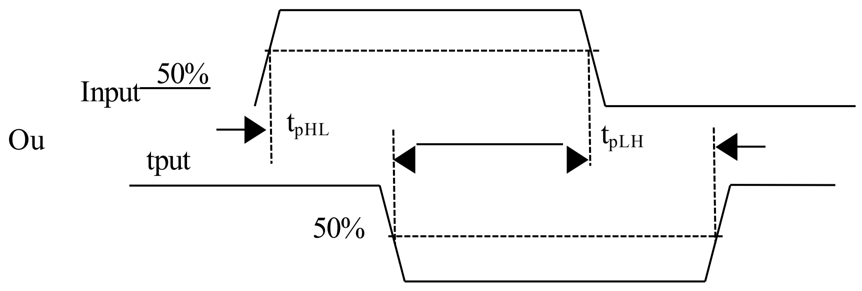
CLASSIFICATION OF LOGIC FAMILIES

Classification as per Level of Integration

As per this classification digital integrated circuits can be classified as small scale integration (SSI), Medium scale integration (MSI), large scale integration (LSI), Very large scale integration (VLSI) and ultra large scale integration (ULSI). This classification is given in SSI are the least complex and include basic gates and flip flops. They are available in dual in package (DIP) or flat package and 14 or 16 pin versions. Small digital sub systems form the MSI category. The complex logic functions, e.g., adders, registers, comparators, code converters, counter, multiplexers etc., are fabricated in MSI. They are also available in dual in package (KIP), flat package and carrier package with 24 or 28 pins.

Classification as per Technology

Digital ICS are manufactured by two technologies viz., Bipolar and MOS. The bipolar family uses transistor fabricated on a chip. This family includes DTL (Diode transistor logic using diodes and transistor), TTL (transistor-transistor logic which propagation delay time of the logic gate is taken as the average of these two delay times.



Input and output voltage waveforms to define propagation delay tines.

Power Dissipation

This is the amount of power dissipated in an IC. It is determined by the current, $1_{CC,}$ that it draws from the V_{CC} supply, and is given by $V_{CC} \times 1_{CC}$. 1_{CC} is the average value of 1_{CC} (0) and 1_{CC} (1). This power is specified in milli watts.

Figure of Merit

The figure of merit of a digital IC is defined as the product of speed and power. The speed is specified in terms of propagation delay time expressed in nanoseconds.

Figure of merit = propagation delay time (ns) × power

(mW) It is specified in pico joules (ns × mW = p^J)

A low value of speed-power product is desirable. In a digital circuit, if it is desired to have high speed, i.e. low propagation delay, then there is a corresponding increase in the power dissipation and vice-versa.

Fan-Out

This is the number of similar gates which can be driven by a gate, High fan-out is advantageous because it reduces the need for additional drivers to drive more gates.

Current and Voltage Parameters

The following currents and voltages are specified which are very useful in the design of digital system.

High-level input voltage, V_{IH}: This is the minimum input voltage which is recognized by the gate as logic 1.

Low-level input voltage, V_{IL}: This is the maximum input voltage which is recognized by the gate as logic 0.

TTL is the most popular family in SSI and MSI category. MOS family includes PMOS (p-channel MOSFET), NMOS (n0channel MOSFET) and CMOS (complementary MOSFET). PMOS is almost obsolete. NMOS is dominating the LSI field. CMOS is the most commonly used technology for digital wrist watches pocket calculators etc. The technologies being used now-a-days are TTL, ECL and CMOS. Before discussing these technologies we discuss the important specifications of digital ICs.

CHARACTERISTICS OF DIGITAL ICs

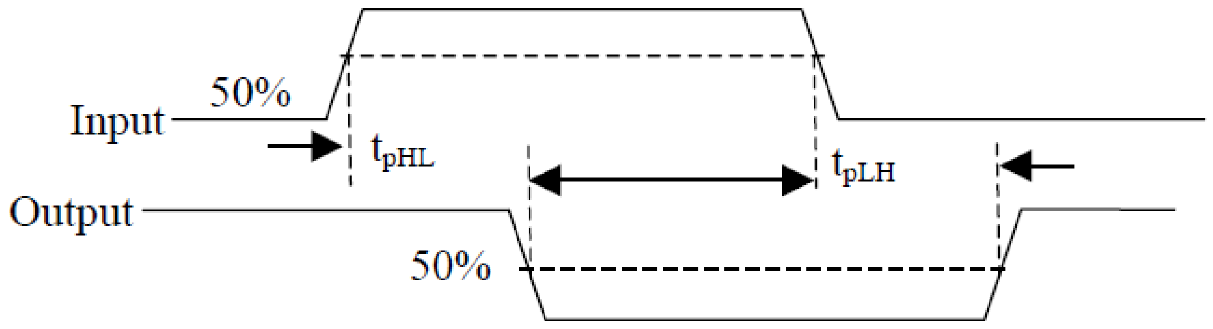
With the widespread use of ICs in digital systems and with the development of various Technologies for the fabrication of ICs, it has become necessary to be familiar with the characteristics of IC logic families and their relative advantages and disadvantages. Digital ICs are classified either according to the complexity of the circuit, as the relative number of individual basic gates (2-input NAND gates) it would require to build the circuit to accomplish the same logic function or the number of components fabricated on the chip.

The various characteristics of digital ICs used to compare their performances are:

1. Speed of operation,
2. Power dissipation,
3. Figure of merit,
4. Fan-out,
5. Current and voltage parameters,
6. Noise immunity,
7. Operating temperature range,
8. Power supply requirements, and
9. Flexibilities available.

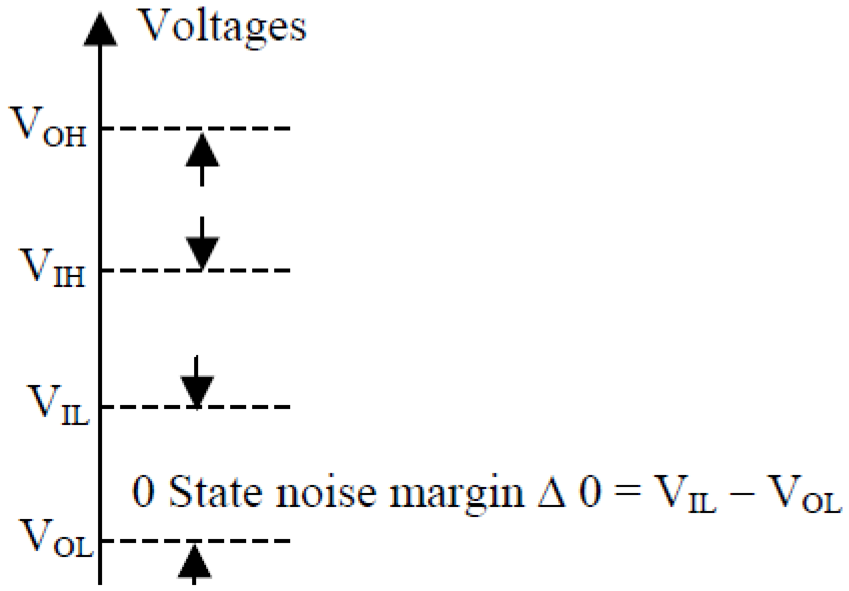
Speed of Operation

The speed of a digital circuit is specified in terms of the propagation delay time. The input and output waveforms of a logic gate are shown. The delay times are measured between the 50 per cent voltage levels of input and output waveforms. There are two delay times: tpLH , when the output goes from the HIGH state to the LOW state and tpLH , corresponding to the output making a transition from the LOW state to the HIGH state. The propagation delay time of the logic gate is taken as the average of these two delay times.



Noise Immunity

The input and output voltage levels defined above are shown. Stray electric and magnetic fields may induce unwanted voltages, known as noise, on the connecting wires between logic circuits. This may cause the voltage at the input to a logic circuit to drop below VIH or rise above VIL and may produce undesired operation. The circuit’s ability to tolerate noise signals is referred to as the no8ise immunity, a quantitative measure of which is called noise margin. Noise margins are illustrated in Fig. The noise margins defined above are referred to as dc noise margins. Strictly speaking, the noise is generally thought of as an a.c. signal with amplitude and pulse width.



Operating Temperature

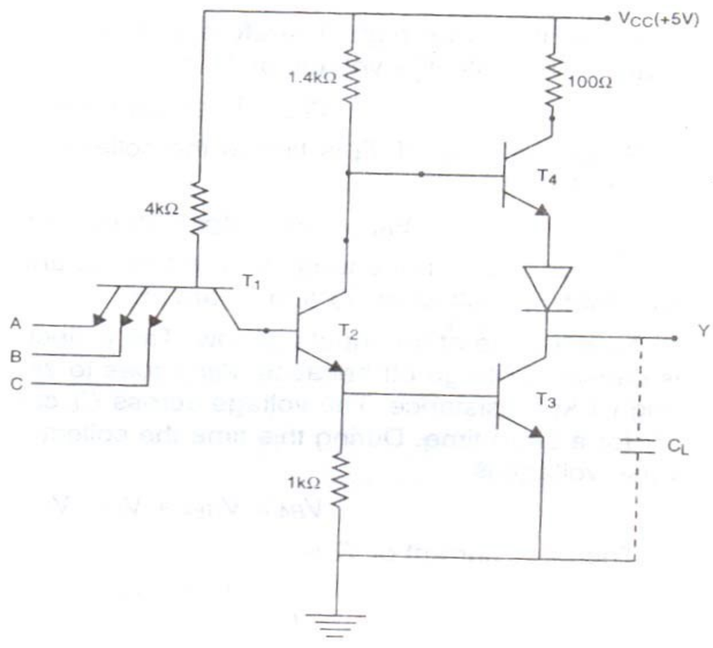
The temperature range in which an IC functions properly must be known. The accepted temperature ranges are: 0 to + 70 ^{0}C for consumer and industrial applications and --55 ^{0}C to + 125 ^{o}C for military purposes.

Power Supply Requirements

The supply voltage (s) and the amount of power required by an IC are important characteristics required to choose the proper power supply.

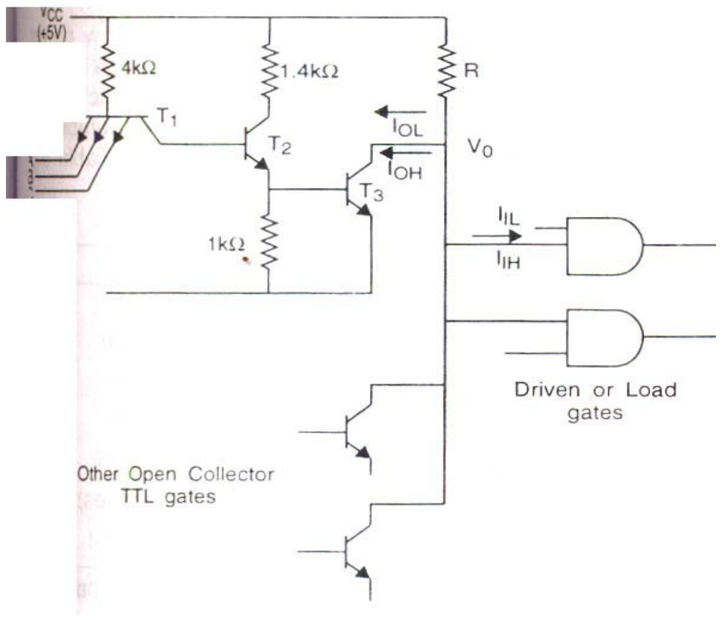
TRANSISTOR-TRANSISTOR LOGIC (TTL)

Fig. below shows a TTL NAND gate with a totem pole output. The totem pole output means that transistor T4 sits atop T3 so as to give low output impedance. The low output impedance implies a short time constant RC so that the output can change quickly from one state to another. T1 is a multiple emitter transistor. This transistor can be thought of as a combination of many transistors with a common base and collector. Multiple emitter transistors with about 60 emitters have been developed. In the T1 has 3 emitters so that there can be three inputs A, B, C. The transistor T2 acts as a phase splitter because the emitter voltage is out of phase with the collector voltage. The transistors T3 and T4 form the totem pole output. The capacitance CL represents the stray capacitance etc. The diode D is added to ensure that T4 is cut off when output is low. The voltage drop of diode D keeps the base emitter junction of T4 reverse biased so that only T3 conducts when output is low. The operation can be summed up as under:



Open Collector Connection

TTL gates without active pull up and with collector open and brought out can be connected for wired AND connection. Fig.below shows one such connection. In this case all the open collector terminals share one common pull up resistance R. The size of this resistance R depends on the number of open collector gates, required noise margin, fan out etc. In any of the input (say A) of TTL gate is not used (and left unconnected or open) the corresponding emitter base junction of T1 will not be forward biased and behave as if logical 1 is applied to this input. Therefore, the unused input terminal of any TTL gate should preferably be TTL gates are used with clamping diodes connected between each input terminal and ground as shown in Fig. These diodes clamp the input to – 0.7 V and minimize undesired negative noise transients.

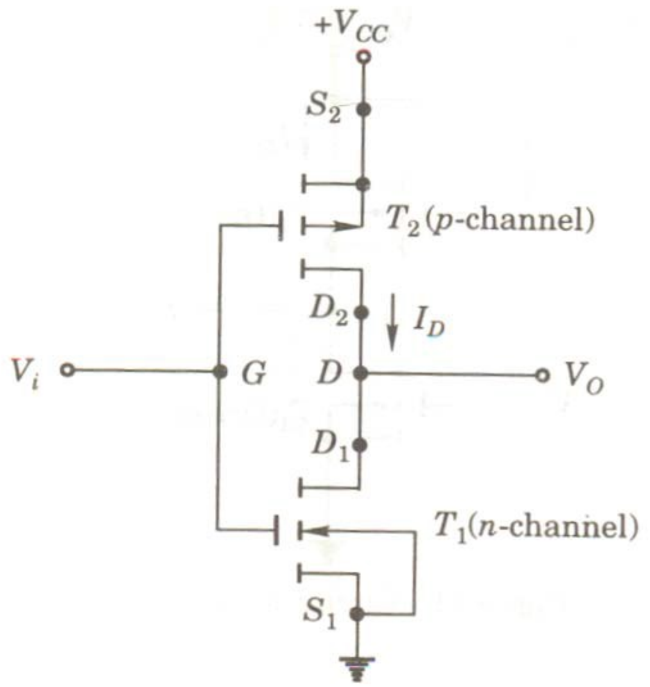


CMOS LOGIC

A complementary MOSFET (CMOS) is obtained by connecting a p-channel and an n-channel MOSFET is series, with drains tied together and the output is taken at the common drain. Input is applied at the common gate formed by connecting the two gates together. In a CMOS, p-channel and n-channel enhancement MOS devices are fabricated on the same chip, which makes its fabrication more complicated and reduces the packing density. But because of negligibly small power consumption, CMOS is ideally suited for battery operated systems. Its speed is limited by substrate capacitances. To reduce the effect of these substrate capacitances, the latest technology known as silicon on sapphire (SOS) is used in microprocessor fabrication which employs an insulating substrate (sapphire). CMOS is becoming very popular in MSI and LSI areas.

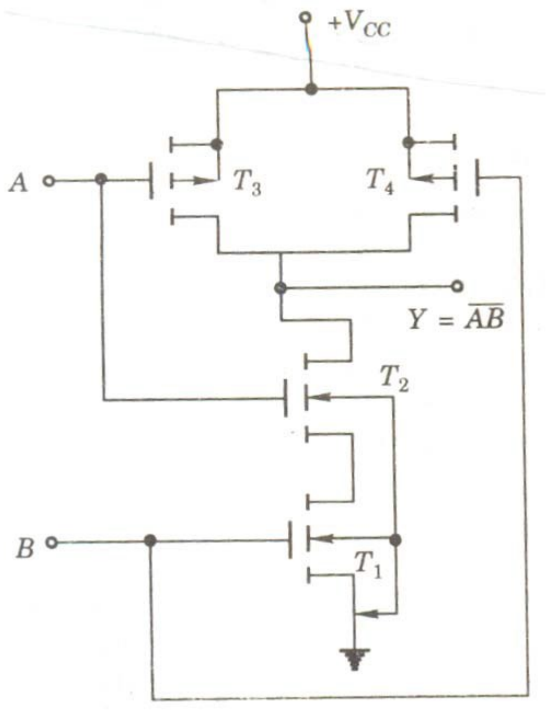
CMOS Inverter

The basic CMOS logic circuit is an inverter shown in Fig. For this circuit the logic levels are 0 V (logic 0) and VCC (logic 1). When Vi = VCC, T1 turns On and T2 turns OFF. Therefore V0 = 0 V, and since the transistors are connected in series the current ID is very small. On the other hand, when Vi = 0 V, T1 turns OFF and T2 turns ON giving an output voltage V0 = VCC and ID is again very small. In either logic state, T1 or T2 is OFF and the quiescent power dissipation which is the product of the OFF leakage current and VCC is very low. More complex functions can be realized by combinations of inverters.

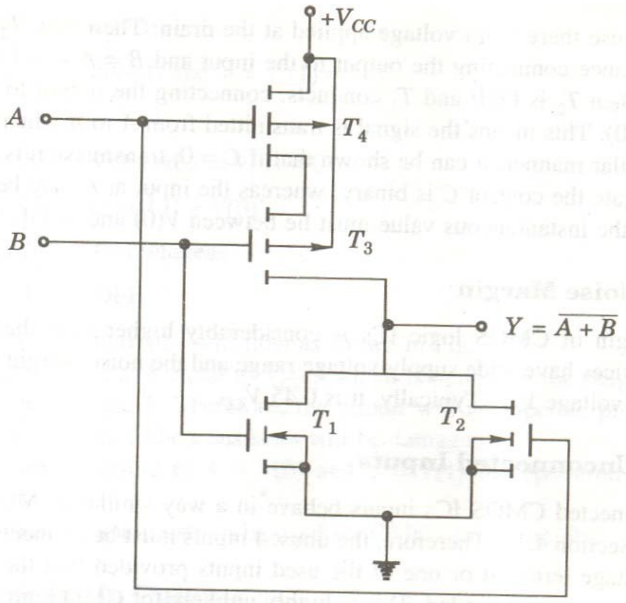


CMOS NAND and NOR Gates

A 2-input CMOS NAND and NOR gate shown in Fig. In the NAND gate, the NMOS drivers are connected in series, whereas the PMOS loads are connected in parallel. On the other hand, the CMOS NOR gate is obtained by connecting the NMOS drivers in parallel and PMOS loads in series. The operation of NAND gate can be understood from Table.



A 2-input CMOS NAND gate



A 2-input CMOS NOR gate

TRI-STATE LOGIC

In normal logic circuits there are two states of the output, LOW and HIGH. If the output is not in the LOW state, it is definitely in the other state (HIGH). Similarly, if the output is not in the HIGH state, it is definitely in the LOW state. In complex digital systems like microcomputers and microprocessors, a number of gate outputs may be required to be connected to a common line which is referred to as a bus which, in turn, may be required to drive a number of gate inputs. When a number of gate outputs are connected to the bus, we encounter some difficulties. These are:

1. Totem-pole outputs cannot be connected together because of very large current drain from the supply and consequent heating of the ICs which may get damaged.
2. Open-collector outputs can connected together with a common collector-resistor connected externally. This causes the problems of loading and speed of operation.

To overcome these difficulties, special circuits have been developed in which there is one more state of the output, referred to as the *third state* or *high-impedance state,* in addition to the LOW and HIGH states. These circuits are known as TRI-STATE, *tri-state logic* (TSL) or *three-state logic.* TRI-STATE, is a registered Trade Mark of National Semiconductor Corporation of USA.

There is a basic functional difference between wired-OR and the TSL. For the wired-OR connection of two functions Y1 and Y2 is

$$Y = Y1 + Y2$$

Whereas for TSL, the result is not a Boolean function but an ability to multiplex many functions economically.

Module-II

Combinational Digital Circuits

Boolean algebraic theorems are used for the manipulations of logical expressions. It has also been demonstrated that a logical expression can be realized using the logic gates. The number of gates and the number of input terminals for the gates required for the realization of a logical expression, in general, get reduced considerably if the expression can be simplified. Therefore, the simplification of logical expression is very important as it saves the hardware required to design a specific system. A large number of functions are available in IC form and therefore, we should be able to make optimum use of these ICs in the design of digital systems. That is, our aim should be to minimize the number of IC packages. Basically, digital circuits are divided into two broad categories:

1. Combinational circuits, and
2. Sequential circuits,

In ***combinational circuits,*** the outputs at any instant of time depend upon the inputs present at that instant of time. This means there is no memory in these circuits. There are other types of circuits in which the outputs at any instant of time depend upon the present inputs as well as past inputs/outputs. This means that there are elements used to store past information. These elements are known as *memory.* Such circuits are known as ***sequential circuits***. A sequential logic system may have combinational logic sub-systems. The design of combinational circuits will be discussed here. Sequential circuit design will be discussed later. The design requirements of combinational circuits may be specified in one of the following ways

1. A set of statements
2. Boolean expression, and
3. Truth table.

The aim is to design a circuit using the gates already discussed or some other circuits which are in fact derived from the basic gates. As is usual in any engineering design, the number of components used should be minimum to ensure low cost, saving in space, power requirements, etc. There can be two different approaches to the design of combinational circuits. One of these is the traditional method, wherein the given Boolean expression or the truth table is simplified by using standard methods and the simplified expression is realized using the gates. The other method normally does not require any simplification of the logical expression or truth table, instead the complex logic functions available in medium scale integrated circuits (MSI) or large scale integrated circuits (LSI) can be directly used. Combinational circuit design using the traditional design methods has been discussed below. The following methods can be used to simplify the Boolean function:

1. Algebraic method,
2. Karnaugh map technique,
3. Variable entered aping (VEM) technique, and

4. Quino-McCluskey method.

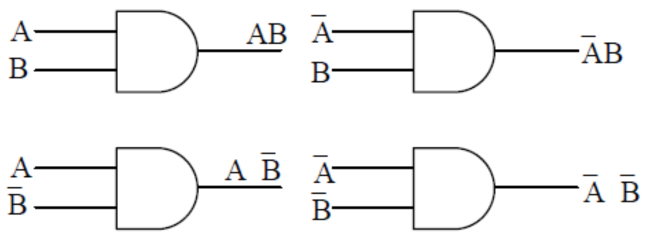
STANDARD FORMS OF LOGIC FUNCTIONS

Design of logic circuits starts with preparation of word equation or truth table for the desired output (0 or 1) for the given input conditions. A logic circuit to implement the above equation/truth table is to be synthesized. The logic expression can be either a sum of products or product of sums.

Fundamental Products and Sum of Products

Fig. below shows four possible ways for connecting two input variables A, B and their complements A, B to AND gate. These four products AB, AB, AB and AB are known as fundamental products and are listed in Table

A	**B**	**Fundamental Product**
0	0	$\bar{A}\bar{B}$
0	1	$\bar{A}B$
1	0	$A\bar{B}$
1	1	AB



For three variables

A	B	C	Fundamental products
0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	$\bar{A}\bar{B}C$
0	1	0	$\bar{A}B\bar{C}$
0	1	1	$\bar{A}BC$
1	0	0	$A\bar{B}\bar{C}$
1	0	1	$A\bar{B}C$
1	1	0	$AB\bar{C}$
1	1	1	ABC

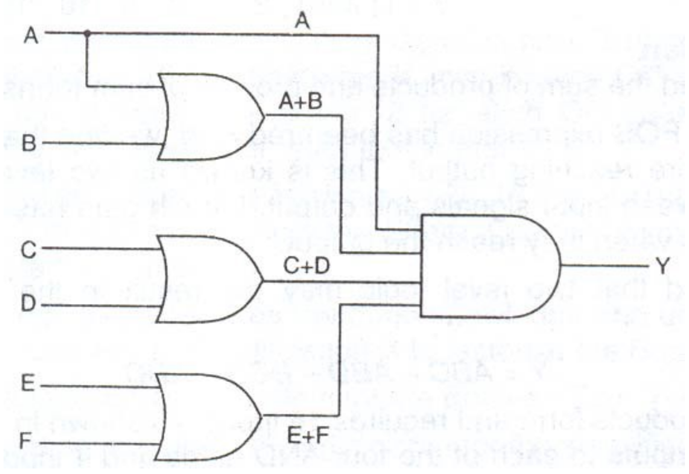
Sum of Product in terms of Minterms

Since the terms ABC must appear in every product, a short hand notation has been developed to save the labour in writing down the letters again and again. TO use this notation substitute 0 for a letter with bar (i.e., NOT ed letter) and 1 for a letter without a br. Express the resultant binary number by a decimal number and write it as a subscript of m. This subscript is used in all truth tables and maps.

Product of Sums

The product of sums, as the name suggests, is an expression involving the product of two or more terms where each term contains sum of a number of variables. The product of sum form is a dual of sum of product form. It may also contain a single variable term. The expression A(A + B) (C + D) is an example of product of sum form. Fig shows a logic circuit corresponding to expression A(A + B) (C + D).

Example Identify the form of the expression
Y = A(A + B) (C + D) (E + F)
Draw logic circuit.
Solution: It is product of sum form. Fig below shows the logic circuit. It has three OR gates and one AND gate.



Product of Sums in terms of Max Terms

Just as the sum of products can be written in short hand using minterms, the product of sums can be written in short hand using max terms. If a system has variables A, B, C then the max terms would be in the form A + B + C . A Boolean expression written in max terms takes the form

$$Y = \prod M(0, 1, 3, 4)$$

Where the capital $\prod$ represents the product and M stands for max terms.

The numbering of max terms is different from numbering of min terms. The unbarred letters represent 0s and the barred (NOT ed) letters represent 1s in forming the maxterms designation.

SIMPLIFICATION OF LOGICAL FUNCTIONS USING K-MAP

Simplification of logical functions with K-map is based on the principle of combining terms in adjacent cells. Two cells are said to be adjacent if they differ in only one variable. For example, in the two-variable K-maps, the top two cells are adjacent and the bottom two cells are adjacent. Also, the left two cells and the right two cells are adjacent. It can be verified that in adjacent cells one of the literals in same, whereas the other literal appears in uncomplemented form in one and in the complemented form in the other cell. Similarly, we observe adjacent cells in the 3-variable and 4-variable K-maps

Grouping Two Adjacent Ones

If there are two adjacent ones on the map, these can be grouped together and the resulting term will have one less literal than the original two terms. It can be verified for each of the groupings of two ones as given in Table

Cell with decimal number	Decimal numbers of adjacent cell		
	2-variable	3-variable	4-variable
0	1, 2	1, 2, 4	1, 2, 4, 8
1	0, 3	0, 3, 5	0, 3, 5, 9
2	0, 3	0, 3, 6	0, 3, 6, 10
3	1, 2	1, 2, 7	1, 2, 7, 11
4		0, 5, 6	0, 5, 6, 12
5		1, 4, 7	1, 4, 7, 13
6		2, 4, 7	2, 4, 7, 14
7		3, 5, 6	3, 5, 6, 15
8			0, 9, 10, 12
9			1, 8, 11, 13
10			2, 8, 11, 14
11			3, 9, 10, 15
12			4, 8, 13, 14
13			5, 9, 12, 15
14			6, 10, 12, 15
15			7, 11, 13, 14

Grouping Four Adjacent Ones

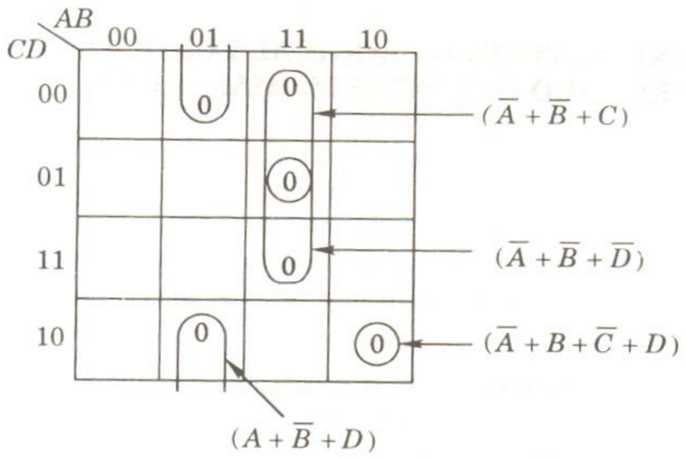
Four cells form a group of four adjacent ones if two of the literals associated with the minterms/maxterms are not same and the other literals are same. Table below gives all possible groups of four adjacent ones for each cell in a 3-variable map. In case of 2-variable map, there is only one possibility corresponding to entry 1 in all the four cells, and the simplified expression will be Y=1. That is, Y always equals 1(independent of the variables). On the basis of groupings of 4 adjacent ones given in Table 3.5, we can find the groupings in K-maps of four or more variables. In the case of a four-variable Kmap, there are six possible groupings of 4-variables involving any cell. It is left to the reader to verify this fact.

Cell with decimal number	Decimal numbers of cells forming groups of adjacent fours		
0	(0, 2, 6, 4),	(0, 1, 2, 3),	(0, 1, 4, 5),
1	(1, 0, 2, 3),	(1, 3, 7, 5),	(1, 0, 4, 5),
2	(2, 0, 6, 4),	(2, 3, 1, 0),	(2, 3, 6, 7),
3	(3, 1, 7, 5),	(3, 2, 1, 0),	(3, 2, 6, 7),
4	(4, 6, 2, 0),	(4, 5, 6, 7),	(4, 5, 0, 1),
5	(5, 1, 3, 7),	(5, 4, 6, 7),	(5, 4, 0, 1),
6	(6, 0, 2, 4),	(6, 7, 4, 5),	(6, 7, 2, 3),
7	(7, 1, 3, 4),	(7, 6, 4, 5),	(7, 6, 2, 3),

Example 3.16 Minimize the logic function of f(A, B, C, D) = ΠM (4, 6, 10, 12, 13, 15) in POS form.

Solution :

$$f(A, B, C, D) = \Pi M\ (4, 6, 10, 12, 13, 15)$$


AB
CD
00
01
11
10
0
$(\overline{A}+\overline{B}+C)$
$(\overline{A}+\overline{B}+\overline{D})$
$(\overline{A}+B+\overline{C}+D)$
$(A+\overline{B}+D)$

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

DON’T-CARE CONDITIONS

We enter 1’s and 0’s in the map corresponding to input variables that make the function equal to 1to 0, respectively. The maps are simplified using either 1’s or 0’s. Therefore, we make the entries in the map for either 1’s or 0’s. The cells which do not contain 1 are assumed to contain 0 and vice-versa. This is not always true since there are cases in which certain combinations of input variables do not occur. Also, for some functions the outputs corresponding to certain combinations of input variables do not matter. In such situations the designer has flexibility and it is left to him whether to assume a 0 or 1 as output for each of these combinations. This condition is known as *don’t-care* condition and can be represented on the K-map as a × mark in the corresponding cell. The × mark in a cell may be assumed to be a 1 or a 0 depending upon which one leads to a simpler expression. The function can be specified in one of the following ways:

1. In terms of minterms and don’t-care conditions,

For example,

$f(A, B, C, D) = \Sigma m\,(1, 3, 7, 11, 15) + d(0, 2, 5)$

Its K-map and the minimized expression are given in Fig.

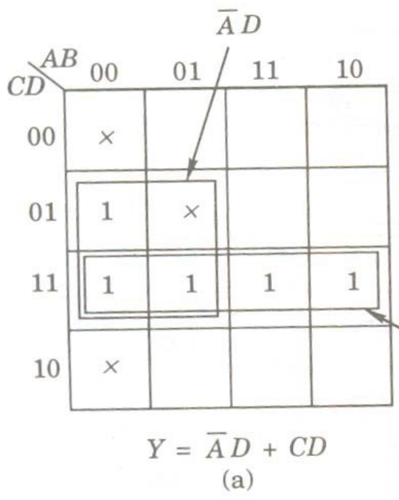
2. In terms of maxterms and don’t-care conditions.

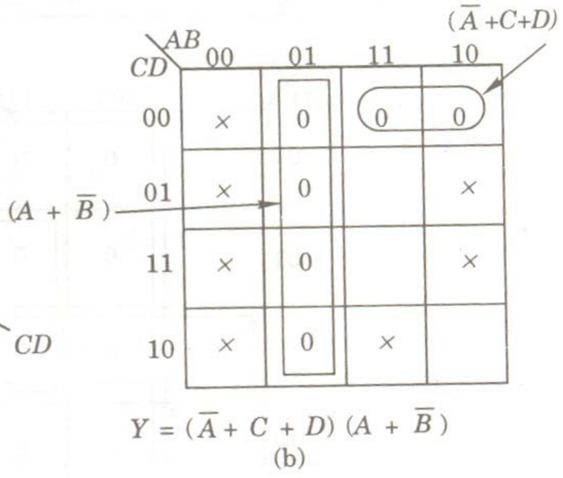
For example,

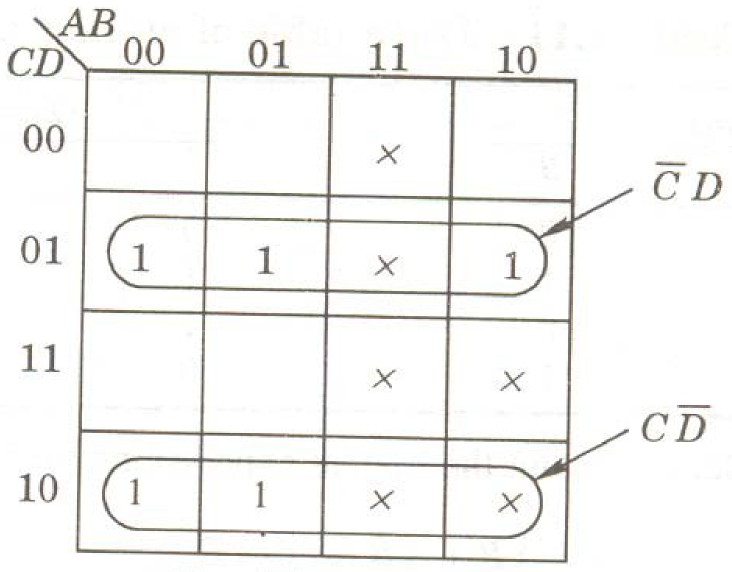
$f(A, B, C, D) = \Pi m\,(4, 5, 6, 7, 8, 12) \,.\, d(1, 2, 3, 9, 11, 14)$

Its K-map and the minimized expression are given in Fig.

3. In terms of truth table. For example, consider the truth table.







$Y = \overline{C}D + C\overline{D}$

(c)

Arithmetic Circuits

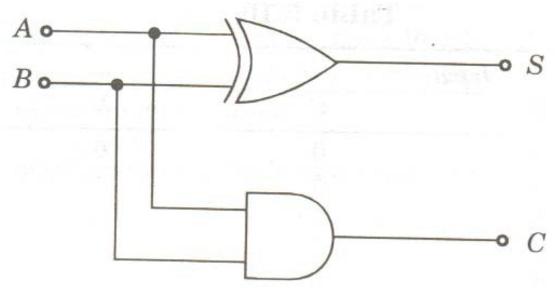
1. Half-adder. A logic circuit for the addition of two one-bit numbers is referred to as an half-adder. The addition process is illustrated in Section 2.5 and is reproduced in truth table. Here, A and B are the two inputs and S (SUM) and C(CARRY) are the two outputs.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$$C = AB$$

The realization of an half-adder using gates is shown in Fig.

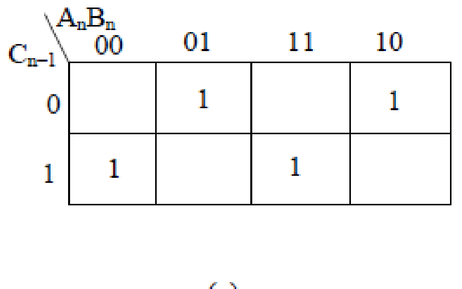


Full-adder

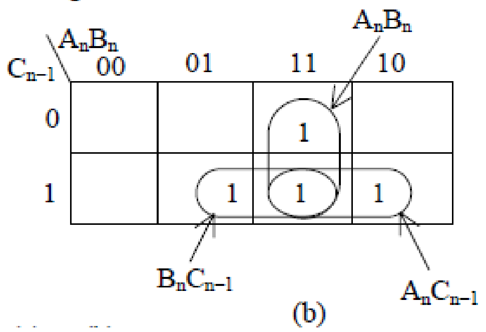
An half-adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multi bit addition is performed. For this purpose, a third input terminal is added and this circuit is used to add An, Bn, and Cn−1, where An and Bn are the nth order bits of the numbers A and B respectively and Cn−1 is the carry generated from the addition of (n−1)th order bits. This circuit is referred to as *full-adder* and its truth table is given in Table

Inputs			Outputs	
A_n	B_n	C_{n-1}	S_n	C_n
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

The K-maps for the outputs S_n and C_n are given in Fig. 3.29.



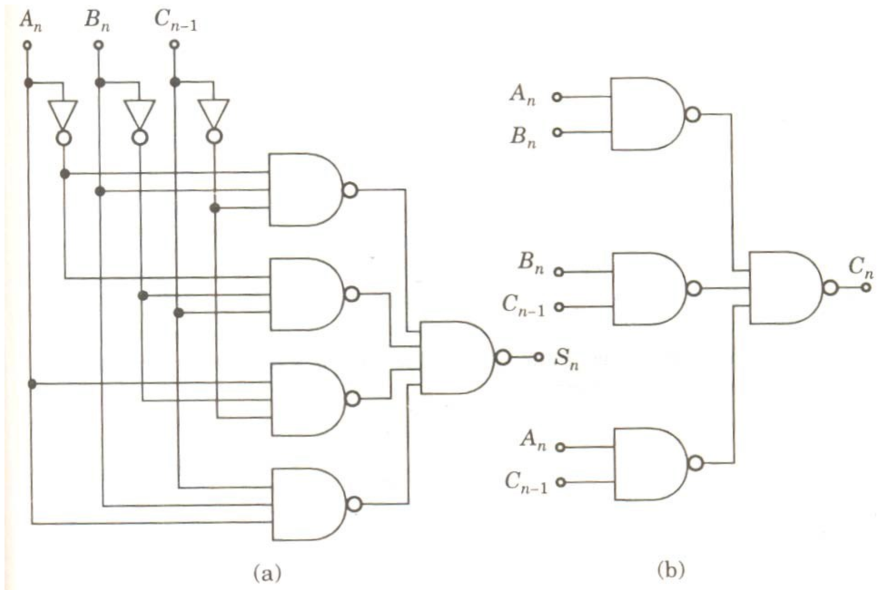
(a)



(b)

$$S_n = \overline{A}_n B_n \overline{C}_{n-1} \overline{A}_n \overline{B}_n C_{n-1} + A_n \overline{B}_n C_{n-1} + A_n B_n C_{n-1}$$

$$C_n = A_n B_n + B_n C_{n-1} + A_n C_{n-1}$$



Half-sub-tractor.

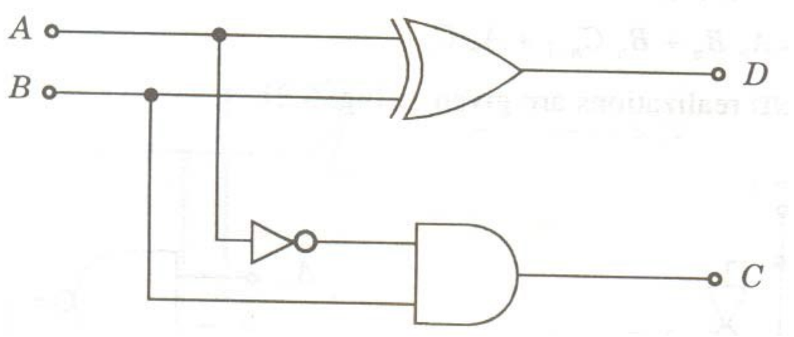
A logic circuit for the subtraction of B (subtrahend) from A (minuend) where A and B are 1-bit numbers is referred to as a *half-sub-tractor.* The subtraction process is reproduced in truth table form in Table 3.13. Here, A and B are the two inputs and D(difference) and C (borrow) are the two outputs.

Inputs		Outputs	
A	B	D	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

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

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

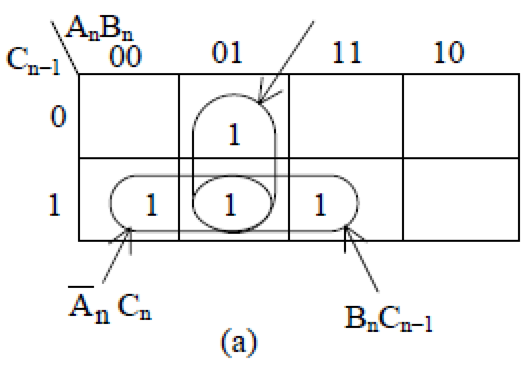
The realization of a half-subtractor using gates is shown in Fig.



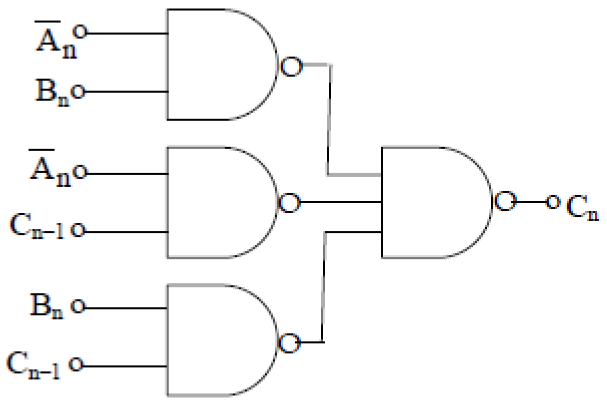
Full-subtractor. Just like a full-adder, we require a *full-subtractor* circuit for performing multibit subtraction wherein a borrow from the previous bit position may also be there. A full subtractor will have three inputs, An (minuend), Bn (sub-trahend) and Cn−1 (borrow from the previous stage) and two outputs, Dn (difference) and Cn (borrow). Its truth table is given in Table. The K-map for the output Dn is exactly same as the K-map for Sn of the adder circuit and therefore, its realization is same as given in Fig.

Inputs			Outputs	
A_n	B_n	C_{n-1}	S_n	C_n
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

$$C_n = \overline{A}_n B_n + \overline{A}_n C_{n-1} + B_n C_{n-1}$$



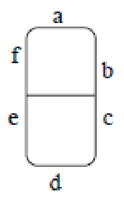
(a)



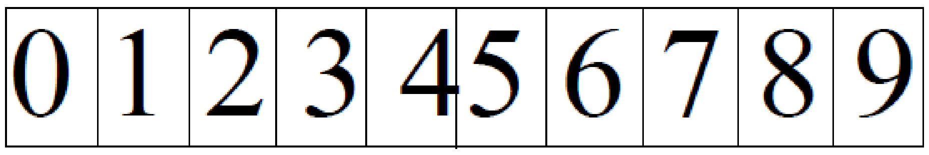
(b)

BCD-to-7-Segment Decoder

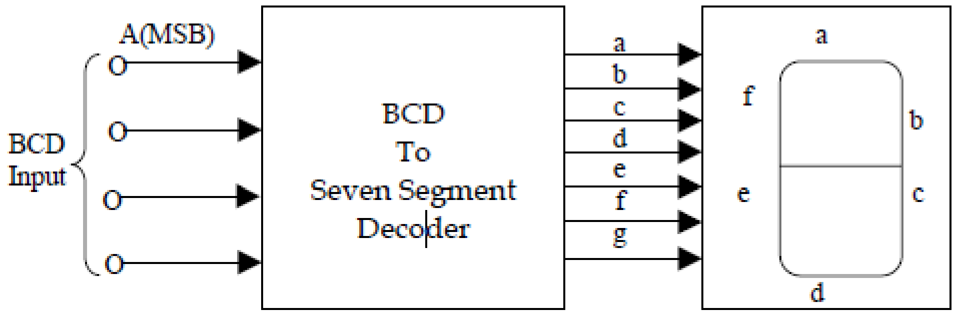
A digital display that consists of seven LED segments is commonly used to display decimal numerals in digital systems. Most familiar examples are electronic calculators and watches where one 7-segment display device is used for displaying one numeral 0 through 9. For using this display device, the data has to be converted from some binary code to the code required for the display. Usually, the binary code used is natural BCD. Figure (a) shows the display device, Fig.(b) shows the segments which must be illuminated for each of the numerals and Fig.(c) gives the display system.



(a)



(b)



(c)

Table below gives the truth table of BCD-to-7-segment decoder. Here ABCD is the natural BCD code for numerals 0 through 9.The entries in the K-map corresponding to six binary combinations not used in the truth table are x − don't-care. The K-maps are simplified and the minimum expressions are given by:

$$a = \overline{B}\,\overline{D} + BD + CD + A$$

$$b = \overline{B} + \overline{C}\,\overline{D} + CD$$

$$c = B + \overline{C} + D = \overline{\overline{B}C\overline{D}}$$

$$d = \overline{B}\,\overline{D} + C\overline{D} + \overline{B}C + B\overline{C}D$$

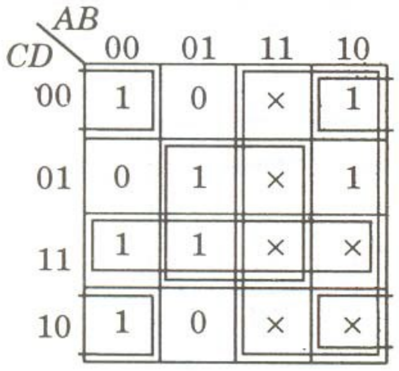
$$e = \overline{B}\,\overline{D} + C\overline{D}$$

$$f = A + \overline{C}\,\overline{D} + B\overline{C} + B\overline{D}$$

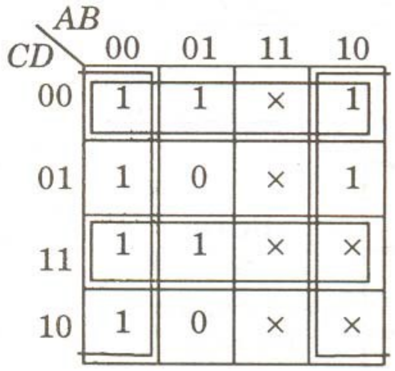
$$g = A + B\overline{C} + \overline{B}C + C\overline{D}$$

Decimal digit displayed	Input				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

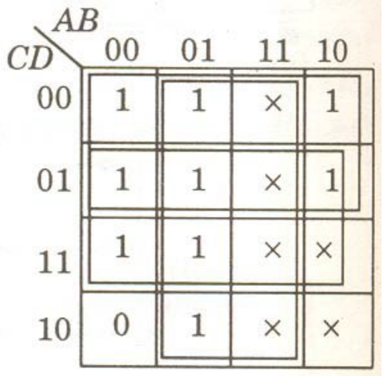
Truth table of BCD-to-7 segment decoder


AB
CD
00 01 11 10
00 1 0 × 1
01 0 1 × 1
11 1 1 × ×
10 1 0 × ×

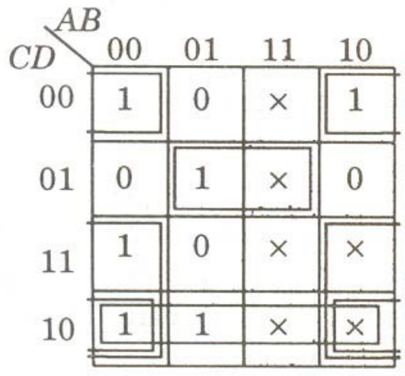
(a)


AB
CD
00 01 11 10
00 1 1 × 1
01 1 0 × 1
11 1 1 × ×
10 1 0 × ×

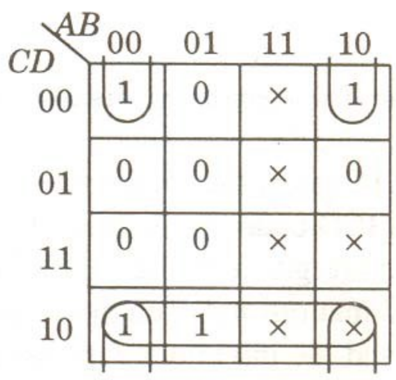
(b)


AB
CD
00 01 11 10
00 1 1 × 1
01 1 1 × 1
11 1 1 × ×
10 0 1 × ×

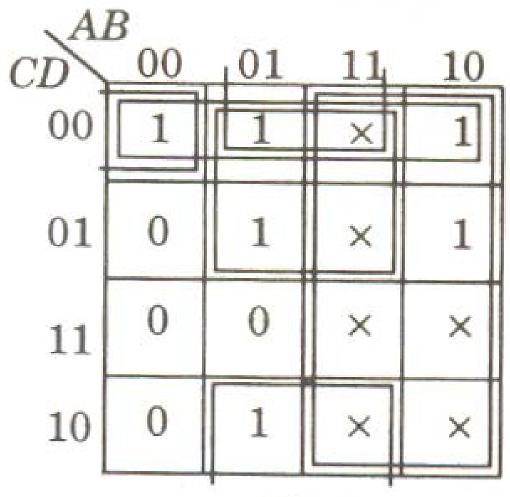
(c)


AB
CD
00 01 11 10
00 1 0 × 1
01 0 1 × 0
11 1 0 × ×
10 1 1 × ×

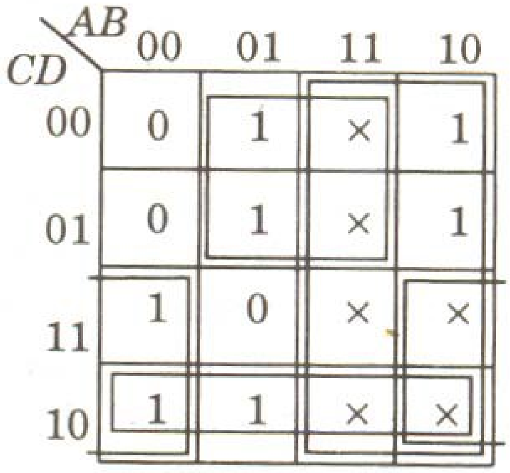
(d)


AB
CD
00 01 11 10
00 1 0 × 1
01 0 0 × 0
11 0 0 × ×
10 1 1 × ×

(e)


AB
CD
00 01 11 10
00 1 1 × 1
01 0 1 × 1
11 0 0 × ×
10 0 1 × ×

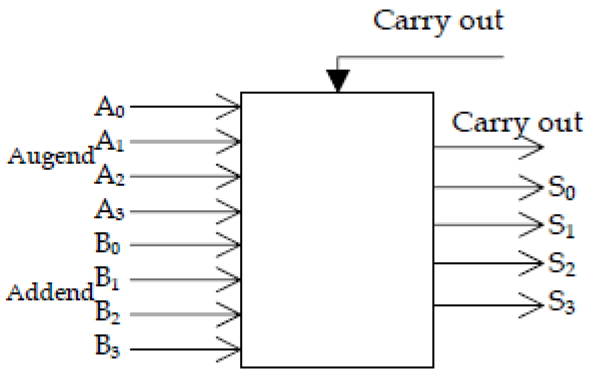
(f)


AB
CD
00 01 11 10
00 0 1 × 1
01 0 1 × 1
11 1 0 × ×
10 1 1 × ×

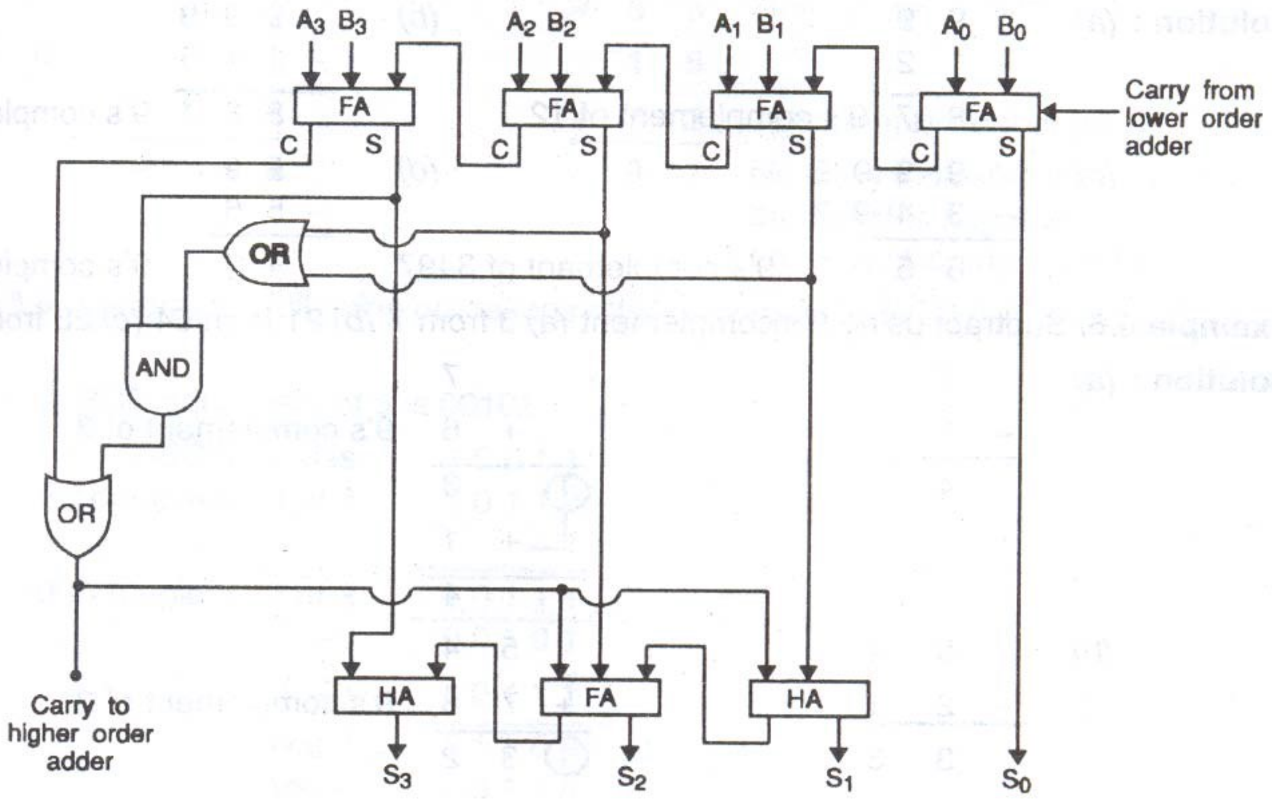
(g)

BCD ADDER

We have discussed the BCD number representation earlier. In BCD system each decimal digit is represented by its equivalent 4 bit binary number, e.g., 63 is written as 01100011 where 0110 represents 6 and 0011 represents 3. Since the highest decimal number is 9 (i.e., 1001) the binary combinations 1010, 1011, 1100, 1101, 1110 and 1111 are not valid in BCD system. In BCD addition, as and when the sum exceeds 9, we add 6 to the sum. A block diagram of BCD adder has been shown in Fig.. It has 9 bits of inputs. 4 bits of augend, 4 bits of addend and one bit for carry in from the lower stage. The output has 5 bits, i.e., 4 bits of SUM and one bit for carry out. Fig below shows the logic diagram of BCD adder made up from full and half adders,



OR gates and AND gate. It has provision for taking care of invalid combinations in BCD



BCD adder
6 + 7 = 13 or 0110 + 0111 = 1101
+ 0110
1 0011 = 3

↓

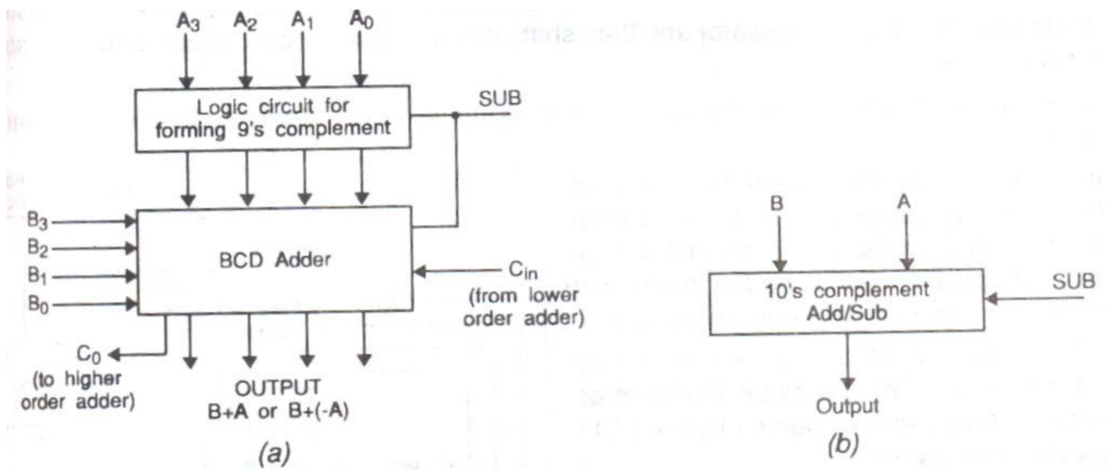
Carry
8 + 6 = 14 or 1000 + 0110 = 1110
+ 0110
1 0011 = 3

↓

Carry

BCD ADDER/SUBTRACTOR

BCD addition and subtraction is very easy is 10's complement and is used in calculators. The logic circuits of Fig.(a) and (b) are connected in tandem. A block diagram of BCD adder/subtractor is shown in Fig. 3.39 (a). Let B3 B2 B1 B0 be the first BCD number and let A3 A2 A1 A0 be the number to be added or subtracted from the first number. The number B3 B2 B1 B0 is fed directly to the BCD adder/subtractor while



BCD adder/subtractor (a) block diagram (b) symbol

When SUB is low, the A bits pass through the logic circuit as it is and BCD adder gives the sum.* S = A + B When SUB is high, A bits are converted to 9's complement by the logic circuit. Moreover, a High SUB adds 0001 to the output of the logic circuit. This is equivalent to forming 10's complement of A bits. Thus we get the result as −A + B or B + (−A), i.e., number A is subtracted from number B. A symbol for BCD adder/subtractor is shown in Fig(b).

BINARY MULTIPLIER

We have discussed the rules for binary multiplication in chapter 2. These are $0 \times 0 = 0$, $0 \times 1 = 0$, $1 \times 0 = 0$ and $1 \times 1 = 1$. To understand the working of a binary multiplier, let us multiply binary number 111 by 101. The method proceeds as under

			1	1	1	Multiplicand
			1	0	1	Multiplier
			1	1	1	First partial product
		0	0	0		Second partial product
		0	1	1	1	Sum of first and second partial products
	1	1	1			Third partial product
1	0	0	0	1	1	Final result

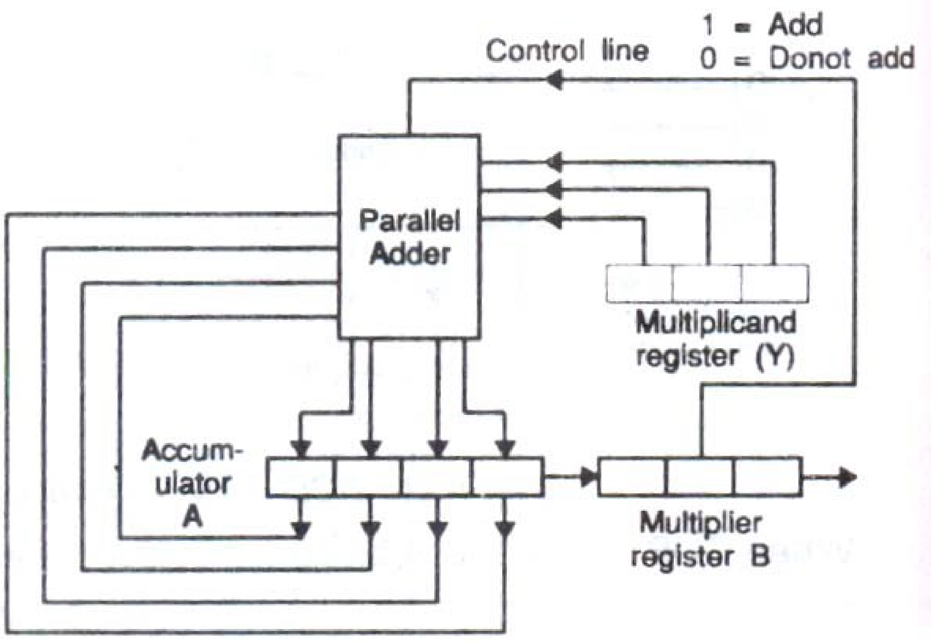
From the above multiplication we can write the following rules for binary multiplication. The logic circuit for binary multiplier must follow these rules.

1. The partial product is 000, if the multiplier bit is 0 and is equal to multiplicand if multiplier bit is 1.
2. The product register* needs twice as many bits as the multiplicand register.
3. The first partial product is shifted one place to the right (relative to second partial product) when adding.

From the above rules and procedure it is evident that logic circuit for binary multiplication does the following.

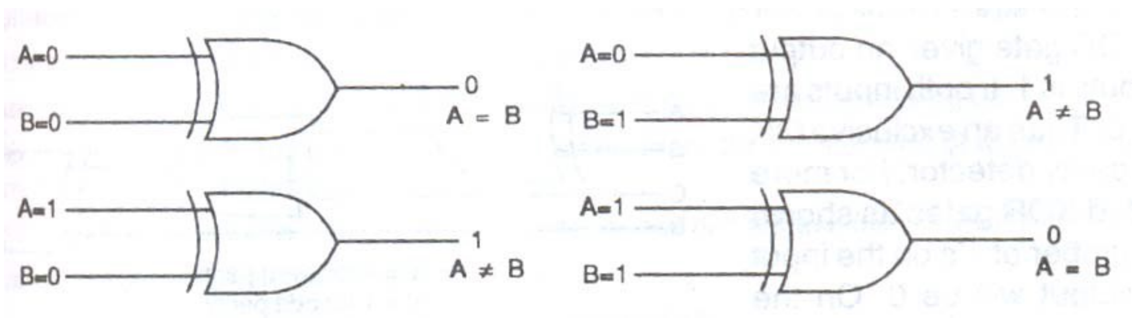
1. It stores the multiplicand and multiplier in separate registers say Y and B respectively.
2. It recognizes a bit as 0 or 1.
3. It stores partial products in accumulator (i.e., B).
4. If the right most bit in multiplier register is 0, it shifts the combined accumulator and B register right one place. If the right most bit in the B register is 1, it adds the contents of Y register (i.e., multiplicand) to the accumulator and then shifts the contents of accumulator and B register right by one place.
5. After each step it examines the right most bit in multiplier register (i.e., B register) whether the bit is 0 or 1.

6. In addition to above it examines the sign bits of multiplicand and multiplier. If both are the same, it stores the sign bit of final result as positive. If the two are different, it stores the sign bit of final result as negative.



MAGNITUDE COMPARATOR

The function of a magnitude comparator is to compare the magnitudes of two numbers and indicate which one is bigger of the two. The simplest form of a magnitude comparator indicates whether the two numbers are equal or not. An exclusive OR is a comparator. If the two input bits are equal its output is 0, if not the output is 1. Thus a 1 bit comparator is just an exclusive OR gate as shown in Fig. below.

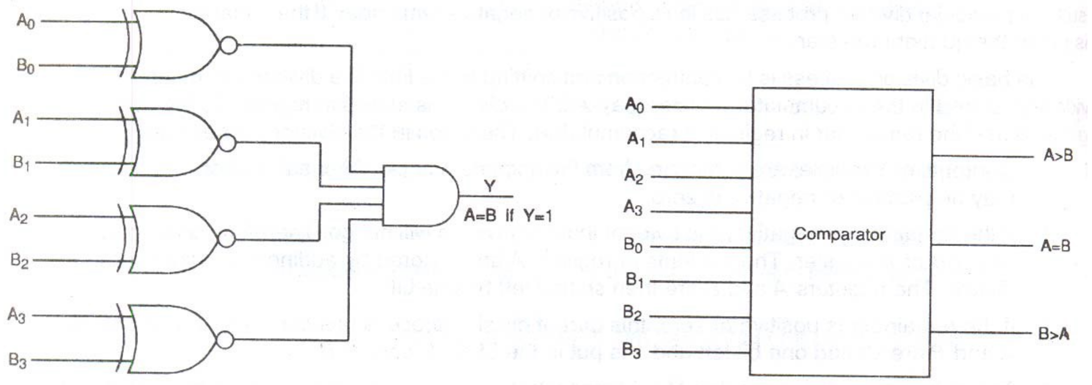


One bit magnitude comparator

When he numbers to be compared are of two bits we need the circuit of Fig.below.

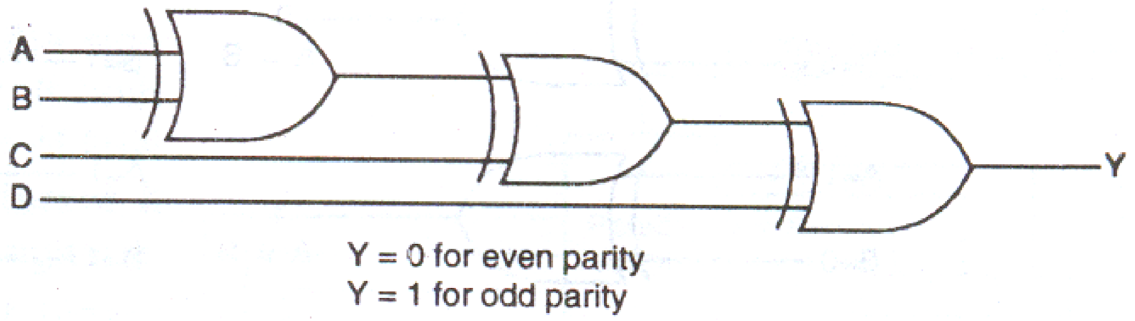
LSBs of the two numbers are A0 and B0 and are fed to XOR gate 1. The MSB of the numbers are A1 and B1 and are fed to XOR gate 2. If the two numbers are equal, outputs of both XOR gates are 0 each and the outputs of NOT gates are 1 each. Then the output of AND gate is 1.

Thus a High output of AND gate indicates A = B. If A ≠ B, one of the XOR gates gives High output and the final output of And gate is low. The above process can be expanded for numbers with more bits. Fig shows a magnitude comparator for 4 bit numbers. It is seen that the combination of XOR and NOT gate in Fig. 3.42 is an exclusive NOR gate. Therefore, the circuit of (a) shows Magnitude comparator for two bit numbers A_1 A_0 and B_1 B_0 exclusive NOR gates. As in the case of two bit numbers, a High output of AND gate indicates that A = B.



PARITY DETECTOR

An exclusive OR gate gives an output of 1 if either of its inputs is 1. If both inputs are 0 or 1, the output is 0. Thus an exclusive OR gate can work as a parity detector. For more bits we use cascaded XOR gates as shown in Fig. below. If the number of 1's on the input is even, the final output will be 0. On the other hand if the number of 1's on the input is odd, the final output will be 1.

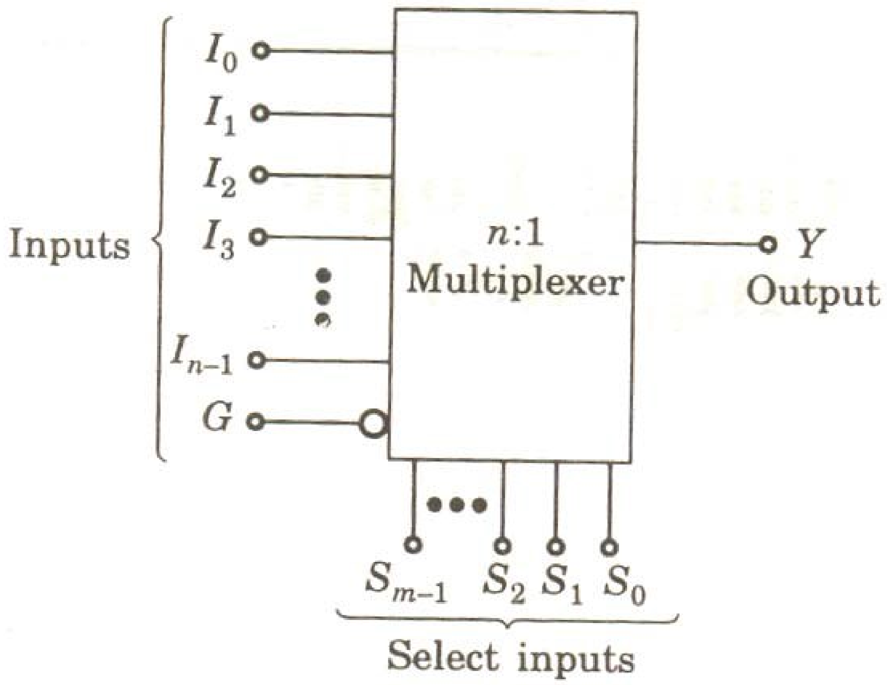


A parity bit can be added to the code group either at the beginning or at the end depending on the system design. However, the total number of 1's, including parity bit is even for even parity and odd for odd parity. Table below lists the parity bits for various numbers on BCD code.

Even Parity					Odd Parity				
P	8	4	2	1	P	8	4	2	1
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	0	1	1
1	0	1	0	0	0	0	1	0	0
0	0	1	0	1	1	0	1	0	1
0	0	1	1	0	1	0	1	1	0
1	0	1	1	1	0	0	1	1	1
1	1	0	0	0	0	1	0	0	0
0	1	0	0	1	1	1	0	0	1

Multiplexer

The multiplexer is a special combinational circuit that is one of the most widely used standard circuits in digital design. The multiplexer (or data selector) is a logic circuit that gates one out of several inputs to a single output. The input selected is controlled by a set of select inputs. Figure shows the block diagram of a multiplexer with n input lines and one output line. For selecting one out of n inputs for connection to the output, a set of m select inputs is required, where 2m = n. Depending upon the digital code applied at the select inputs one out of n data sources is selected and transmitted to a single output channel. Normally, a strobe (or enable) input (G) is incorporated which helps in cascading and it is generally active-low, which means it performs its intended operation when it is LOW.

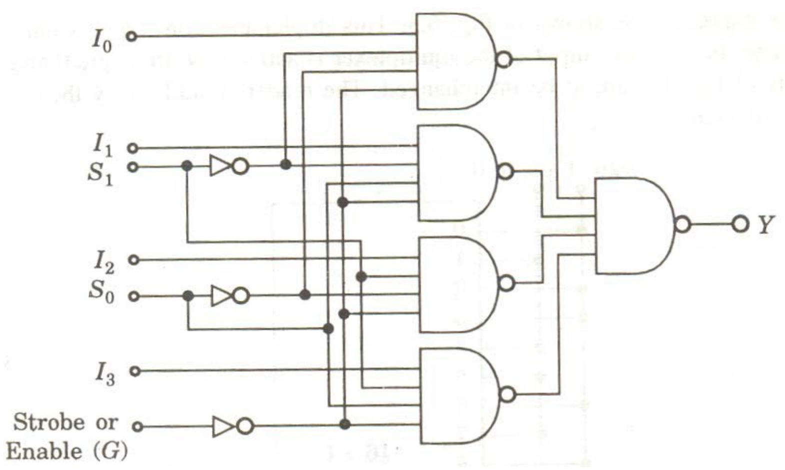


Select inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = \overline{S_1}\,\overline{S_0}\,I_0 + \overline{S_1}\,S_0\,I_1 + S_1\,\overline{S_0}\,I_2 + S_1\,S_0\,I_3$$

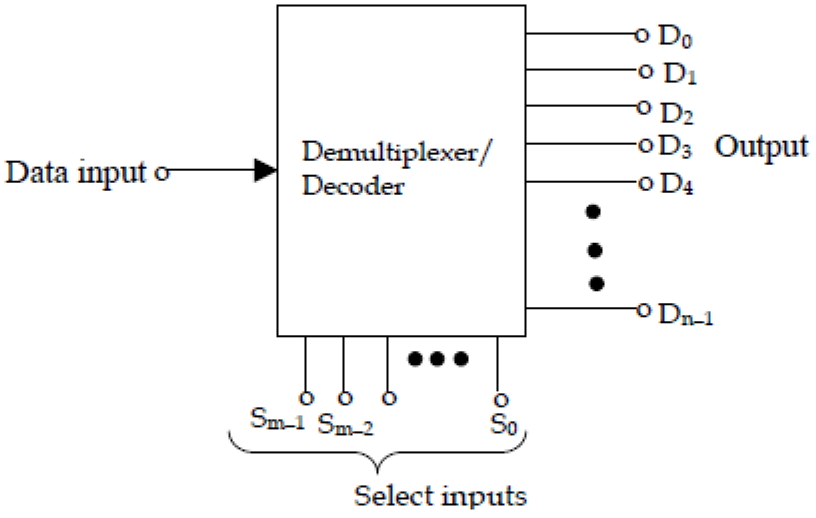
Combinational Logic Design Using Multiplexers

The multiplexing function discussed above can conveniently be used as a logic element in the design of combinational circuits. Standard ICs are available for 2 : 1, 4 : 1, 8 : 1, and 16 : 1 multiplexers.



Demultiplexer

The demultiplexer performs the reverse operation of a multiplexer. It accepts a single input and distributes it over several outputs. Figure below gives the block diagram of a demultiplexer. The select input code determines to which output the data input will be transmitted.



The number of output lines in n and the number of select lines is m, where n = 2m. This circuit can also be used as binary-to-decimal decoder with binary inputs applied at the select input lines and the output will be obtained on the corresponding line. The data input line is to be connected to logic 1 level. This circuit can be designed using gates and it is left as an exercise for the reader. However, this device is available as an MSI IC and can conveniently be used for the design of combinational circuits. The device is very useful if multiple-output combinational circuit is to be designed, because this needs minimum package count. These devices are available as 2-line-to-line, 3-line-to-8-line, and 4-lineto- 16-line decoders.

ENCODERS

Decimal to Binary Encoder

Digital computers operate on binary system. However, we work on decimal numbers and alphabets. The decimal numbers and alphabets are known s alphanumeric characters. An encoder is a device which converts alphanumeric characters to binary codes. An encoder may be decimal to binary, hexadecimal to binary, octal to BCD etc. Fig. shows a decimal to binary encoder. When push button 0 is pressed, none of the OR gates is energized and all outputs are low so that the output word is Y3 Y2 Y1 Y0 = 0 0 0 0

If push button 4 is pressed, Y2 OR gate has high input and the output word is

Y3 Y2 Y1 Y0 = 0 1 0 0

When switch 7 is pressed, OR gates Y2, Y1 and Y0 have high inputs and the output word is Y3 Y2 Y1 Y0 = 0 1 1 1

In a computer, it is required that a binary code be transmitted for every stroke of alphanumeric keyboard (a type writer or teletype). The keyboard of a computer has 26 capital alphabets, 26 lower case alphabets, 10 numerals (0 to 9) and about 22 special characters (+, − etc.). Thus, the total number of input codes is about 84. Encoder for generating an output binary word for these 84 inputs requires 7 bits (because 27 = 128 and 26 = 64). The block diagram of such an encoder is shown in Fig.(b). Whenever a key is pressed, the + 5 V supply is connected to one of the 84 input lines. Inside the box is an array of wires interconnected to generate the required code.

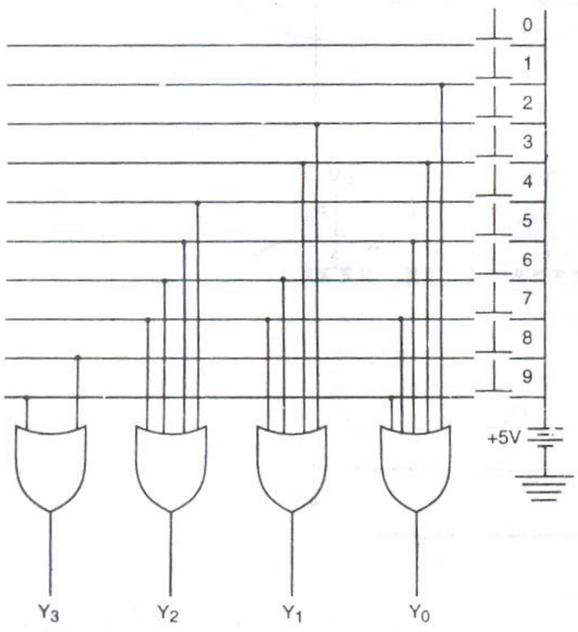


Fig. (a) Decimal to binary encoder

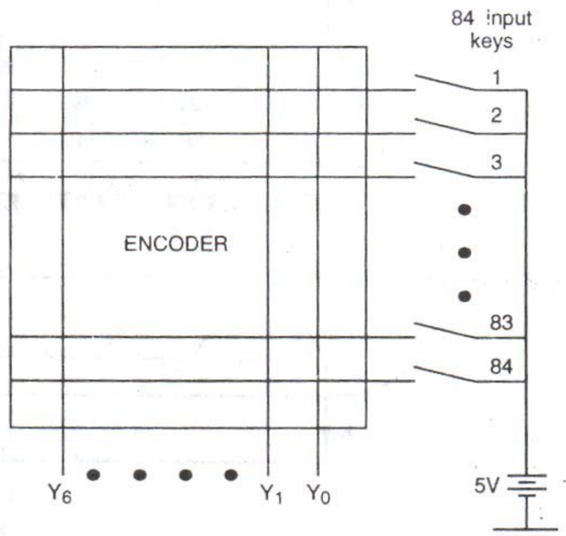


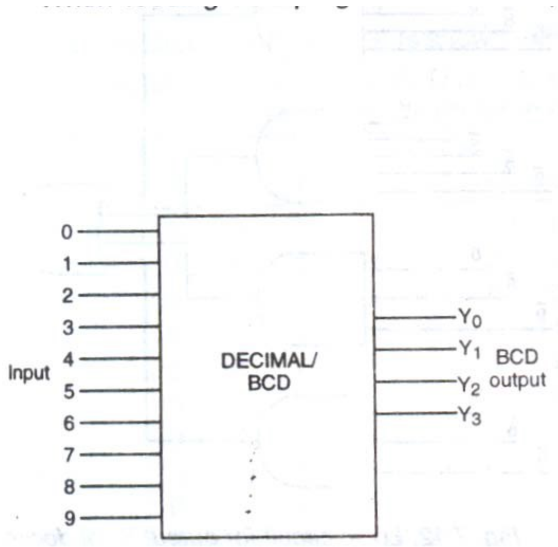
Fig.(b) Block diagram of decimal

Decimal digit	BCD code			
	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0

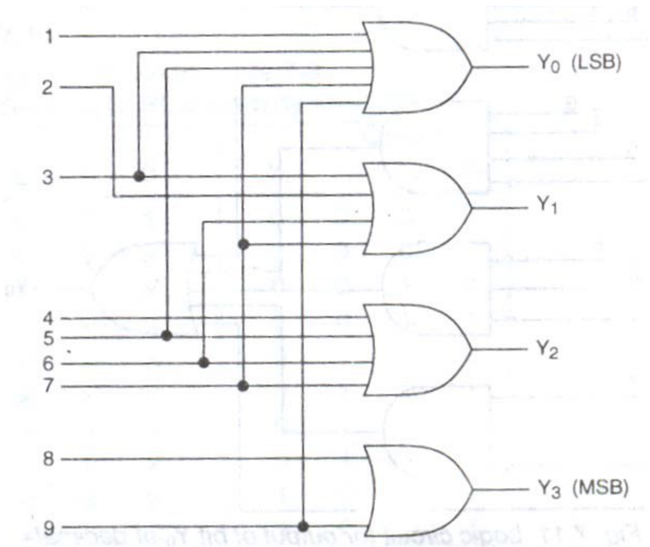
From Table, we can find the relationship between decimal digit and BCD bit.
MSB of BCD bit is Y_3. For decimal digits 8 or 9, $Y_3 = 1$. Thus we can write OR expression for Y_3 bit as
$Y_3 = 8 + 9$
Bit Y_2 is 1 for decimal digits 4, 5, 6, and 7. Thus we can write OR expression
$Y_2 = 4 + 5 + 6 + 7$
Similarly, $Y_1 = 2 + 3 + 6 + 7$
$Y_0 = 1 + 3 + 5 + 7 + 9$

Priority Encoder

When feeding data/program into a computer it is opposite that more than one key is pressed simultaneously. A priority function means tht the encoder will give priority to the



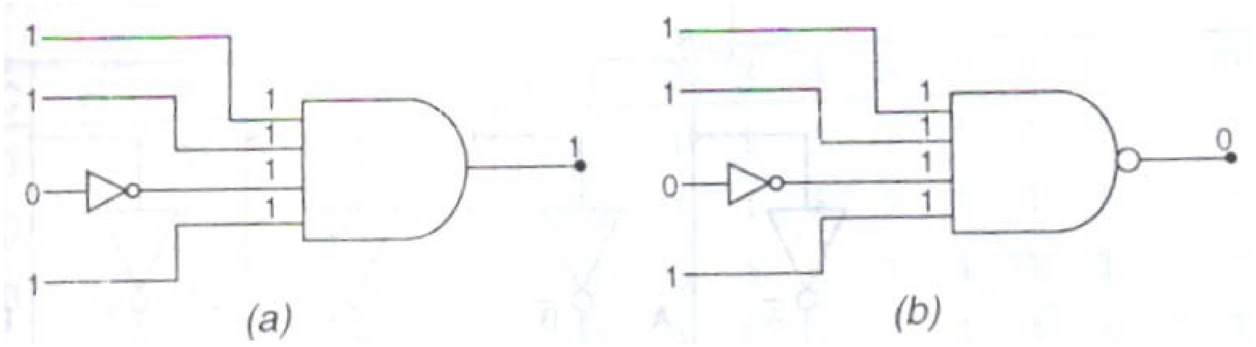
Block diagram of decimal to BCD encoder



Logic circuit for decimal to BCD encoder

Decoder

Let the input be 1101. We need an AND gate as the basic decoding element because an AND gate gives high output only when all its inputs are high. In addition we need a NOT gate to convert 0 in the 2s bit position to 1. Thus the circuit will have one AND gate and one NOT gate as shown in Fig.(a).



Binary to Decimal Decoder
Fig shows a binary to decimal decoder. The input is from a 4 bit register. The inputs would be of the form 0101, 1001 etc. A particular input would cause one of the 10 output lines to be energized so that the corresponding decimal number would be the output, if the input is 0101, only the Y5 AND gate has all high inputs and all other gates have at least one low input so that only Y5 gate would give an output and number 5 would be displayed. If the input is 0 1 1 1, only AND gate Y7 has all high input (all other AND gates have at least one low input) and number 7 is the output. The Boolean equations for different Y outputs are as under

$$Y_0 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

$$Y_1 = \overline{A}\,\overline{B}\,\overline{C}\,D$$

$$Y_2 = \overline{A}\,\overline{B}\,C\,\overline{D}$$

$$Y_3 = \overline{A}\,\overline{B}\,C\,D$$

$$Y_4 = \overline{A}\,B\,\overline{C}\,\overline{D}$$

$$Y_5 = \overline{A}\,B\,\overline{C}\,D$$

$$Y_6 = \overline{A}\,B\,C\,\overline{D}$$

$$Y_7 = \overline{A}\,B\,C\,D$$

$$Y_8 = A\,\overline{B}\,\overline{C}\,\overline{D}$$

$$Y_9 = A\,\overline{B}\,\overline{C}\,D$$

Procedure of Quine-McCluskey Tabular Method

Follow these steps for simplifying Boolean functions using Quine-McClukey tabular method.

Step 1 − Arrange the given min terms in an ascending order and make the groups based on the number of ones present in their binary representations. So, there will be at most ‘n+1’ groups if there are ‘n’ Boolean variables in a Boolean function or ‘n’ bits in the binary equivalent of min terms.

Step 2 − Compare the min terms present in successive groups. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol ‘_’ in the differed bit position and keep the remaining bits as it is.

Step 3 − Repeat step2 with newly formed terms till we get all prime implicants.

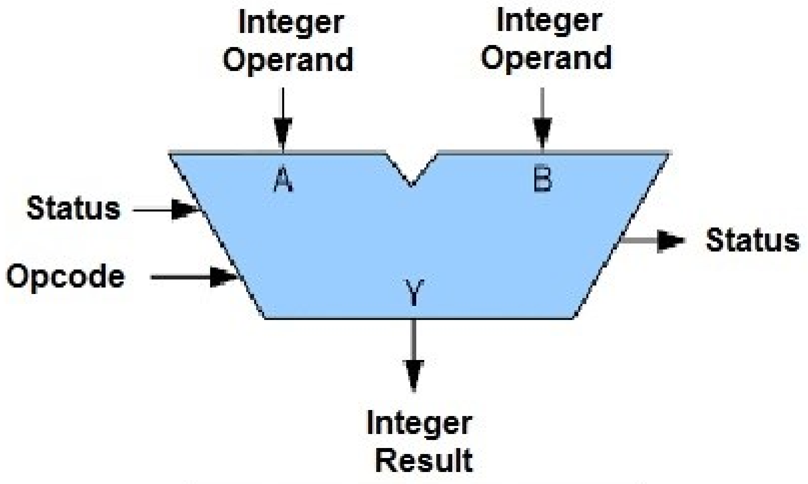
Step 4 − Formulate the prime implicant table. It consists of set of rows and columns. Prime implicants can be placed in row wise and min terms can be placed in column wise. Place ‘1’ in the cells corresponding to the min terms that are covered in each prime implicant.

Step 5 − Find the essential prime implicants by observing each column. If the min term is covered only by one prime implicant, then it is essential prime implicant. Those essential prime implicants will be part of the simplified Boolean function.

Step 6 − Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for Reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

ALU

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU). Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.



Arithmetic Logic Unit (ALU)

ALU is also known as an Integer Unit (IU). The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need. Most of these operations are logical in nature. Depending on how the ALU is designed, it can make the CPU more powerful, but it also consumes more energy and creates more heat. Therefore, there must be a balance between how powerful and complex the ALU is and how expensive the whole unit becomes. This is why faster CPUs are more expensive, consume more power and dissipate more heat.

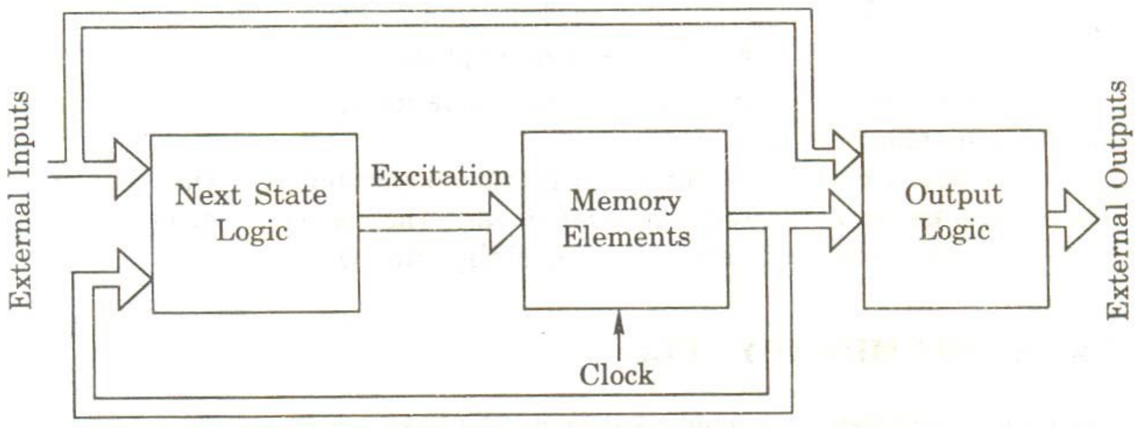
Different operation as carried out by ALU can be categorized as follows –

- logical operations: These include operations like AND, OR, NOT, XOR, NOR, NAND, etc.
- Bit-Shifting Operations: This pertains to shifting the positions of the bits by a certain number of places either towards the right or left, which is considered a multiplication or division operations.
- Arithmetic operations: This refers to bit addition and subtraction. Although multiplication and division are sometimes used, these operations are more expensive to make. Multiplication and subtraction can also be done by repetitive additions and subtractions respectively.

Module-III

Sequential circuits and systems

So far we have directed our studies towards the analysis and design of combinational digital circuits. Though very important, it constitutes only a part of digital systems. The other major aspect of digital systems is analysis and design of sequential circuits. However, sequential circuit design depends, to a large extent, on the combinational circuit design discussed earlier.
There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement cannot be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback form the output back to the input.
A block diagram of a sequential circuit is shown in Fig. It consists of combinational circuits which accept digital signals from external inputs and from outputs of



memory elements and generates signals for external outputs and for inputs to memory elements referred to as excitation.

A memory element is some medium in which one bit of information (1 or 0) can be stored or retained until necessary, and thereafter its contents can be replaced by a new value. The contents of memory elements in Fig. can be changed by the outputs of the combinational circuit which are connected to its input.

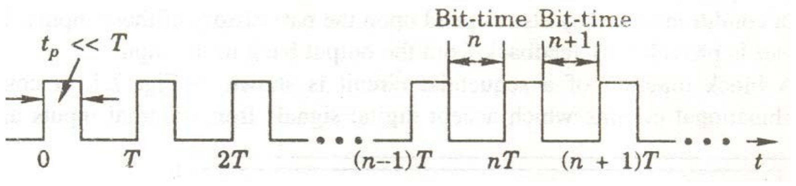
The combinational circuit performs certain operations, some of which are used to determine the digital signals to be stored in memory elements. The other operations are performed on external inputs and memory outputs to generate the external outputs.

The above process demonstrates the dependence of the external outputs of a sequential circuit on the external inputs and the present contents of the memory elements (referred to as the *present state* of memory elements). The new contents of the memory elements, referred to as the *next state;* depend on the external inputs and the present state, hence, the output of a sequential circuit is a function of the time sequence of inputs and the internal states.

Sequential circuits are classified in two main categories, known as *Sychronous* and *Asynchronous* sequential circuits depending on timing of their signals.

A sequential circuit whose behaviour depends upon the sequence in which the input signals change is referred to as an asynchronous sequential circuit. The outputs will be affected whenever the inputs change. The commonly used memory elements in these circuits are time delay devices. These can be regarded as combinational circuits with feedback.

A sequential circuit whose behaviour can be defined from the knowledge of its signal at discrete instants of time is referred to as a synchronous sequential circuit. In these systems, the memory element are affected only at discrete instants of time. The synchronization is achieved by a timing device known as a system clock which generates a periodic train of clock pulses as shown in Fig.below. The outputs are affected only with the application of a clock pulse.



A train of pulses.

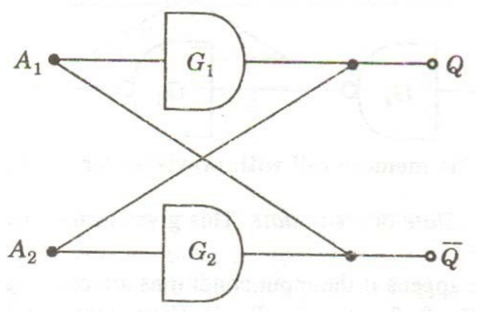
Since the design of asynchronous circuits is more-tedious and difficult, therefore their uses are rather limited.

Synchronous circuits have gained considerable domination and wide popularity and are also known as *clocked-sequential circuits.* The memory elements used are FLIP-FLOPs

which are capable of storing binary information.

4.1 1-BIT MEMORY CELL

The basic digital memory circuit is known as FLIP-FLOP. It has two stable states which are known as the 1 state and the 0 state. It can be obtained by using NAND or NOR gates. We shall be systematically developing a FLIP-FLOP circuit starting from the fundamental circuit shown in Fig. 7.3. It consists of two inverters G_1 and G_2 (NAND gates used as inverters). The output of G_1 is connected to the input of G_2 (A_2) and the output of G_2 is connected to the input of G_1 (A_1).



Cross- coupled inverters as a memory element.

Let us assume the output of G_1 to be Q = 1, which is also the input of G_2 (A_2 = 1). Therefore, the output of G_2 will be $\overline{Q}$= 00, which makes A_1= 0 and consequently Q = 1 which confirms our assumption.

In a similar manner, it can be demonstrated that if Q = 0, then $\overline{Q}$ =1 and this is also

consistent with the circuit connections.

From the above discussion we note the following

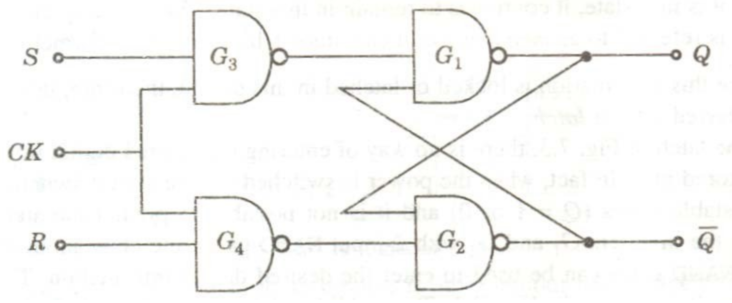
1. The outputs Q and $\overline{Q}$ are always complementary.
2. The circuit has two stable states; in one of the stable state Q = 1 which is referred to as the 1 state (or set state) whereas in the other stable state Q = 0 which is referred to as the 0 state (or reset state).

If the circuit is in 1 state, it continues to remain in this state and similarly if it is in 0

4.1 CLOCKED S-R FLIP-FLOP

It is often required to set or reset the memory cell in synchronism with a train of pulses (Fig. known as clock (abbreviated as CK). Such a circuit is shown in Fig., and is referred to as a *clocked set-reset* (S-R) FLIP- FLOP.

A clocked S-R FOIP-FLOP.



In this circuit, if a clock pulse is present (CK = 1), its operation is exactly the same as above. On the other hand, when the clock pulse is not present (CK =0), the gates G_3 and G_4 are inhibited, i.e. their outputs are 1 irrespective of the values of S or R. In other words, the circuit responds to the inputs S and R only when the clock is present.

Assuming that the inputs do not change during the presence of the clock pulse, we can express the operation of a FLIP-FLOP in the form of the truth table in Table 4.1 for the S- R FLIP-FLOP. Here S_n and R_n denote the inputs and Q_n the output during the bit time Q_{n+1} denotes the output Q after the pulse passes, i.e. in the bit time n + 1.

Inputs		Output
S_n	R_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	?

If S_n = R_n = 0, and the clock pulse is applied, the output at the end of the clock pulse is same as the output before the clock pulse, i. e. $Q_{n+1} = Q_n$. This is indicated in the first row of the truth table.

If S_n = 1 R_n = 0, the output at the end of the clock pulse will be 1, where if S_n =0 and R_n = 1, then Q_{n+1} = 0. These are indicated in the second and third rows of the truth table respectively.

In the circuit of Fig. 4.4, it was mentioned that S = R = 1 is not allowed. Let us see what happens in the S-R FLIP-FLOP of Fig. 4.5 if S_n = R_n = 1. When the clock is present the outputs of gates G_3 and G_4 are both 0, making one of the inputs of G_1 and G_2 NAND gates 0.
Consequently, Q and $\bar{Q}$ both will attain logic 1 which is inconsistent with our assumption of complementary outputs. Now, when the clock pulse has passed away (CK = 0), the outputs of

G_3 and G_4 will rise from 0 to 1. Depending upon the propagation delays of the gates, either the stable state Q_{n+1} = 1($\bar{Q}_{n+1}$ =0) or Q_{n+1} = 0 ($\bar{Q}_{n+1}$ = 1) will result. That means the state of
the circuit is undefined, indeterminate or ambiguous and therefore is indicated by a question mark (fourth row of the truth table).
The condition $S_n = R_n = 1$ is forbidden and it must not be allowed to occur.

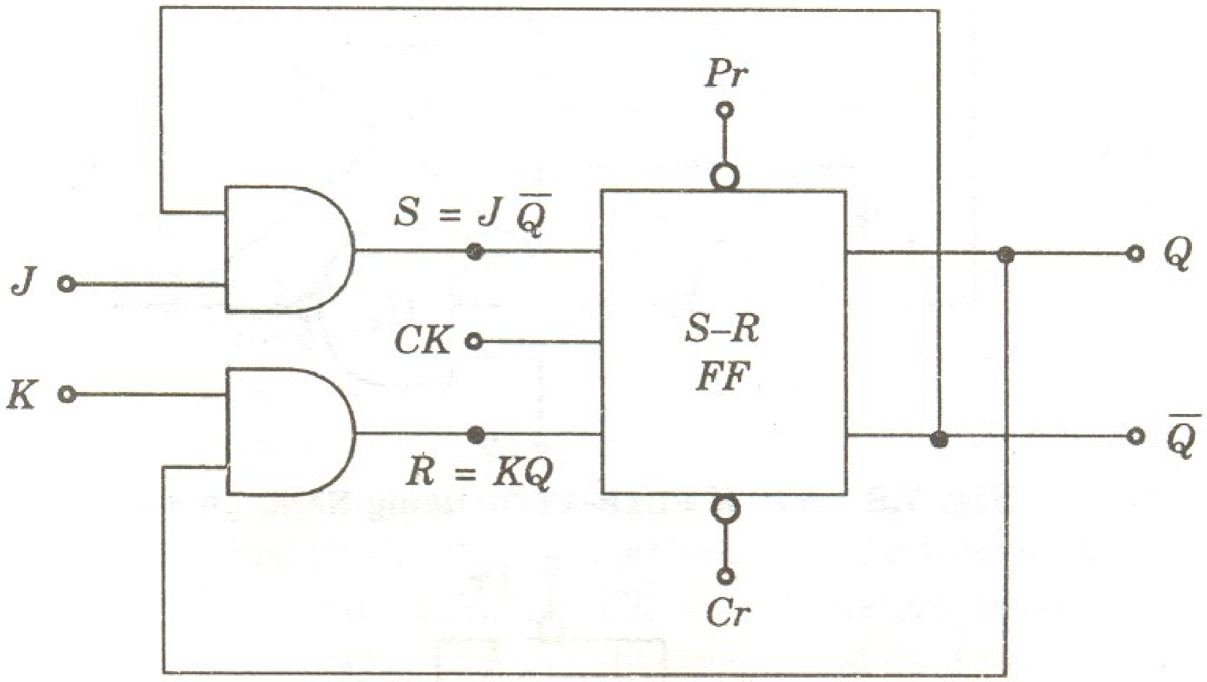
J-K FLIP-FLOP

The uncertainty in the state of an S-R FLIP-FLOP when $S_n = R_n = 1$ (fourth row of the truth table) can be eliminated by converting it into a J-K FLIP-FLOP. The data inputs are J and K which are ANDed with $\overline{Q}$ and Q, respectively, to obtain S and R inputs, i.e.

$$S = J.\ \overline{Q}$$

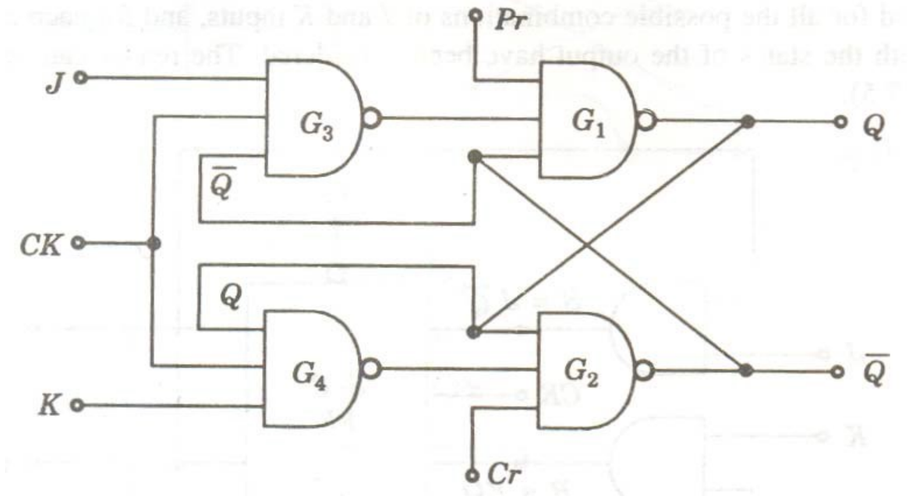
$$R = K.\ Q$$

A J-K FLIP-FLOP thus obtained is shown in Fig. Its truth table is given in Table which is the reduced to another for convenience. Table has been prepared for all the possible combinations of Land K inputs, and for each combination both the states of the output have been considered.



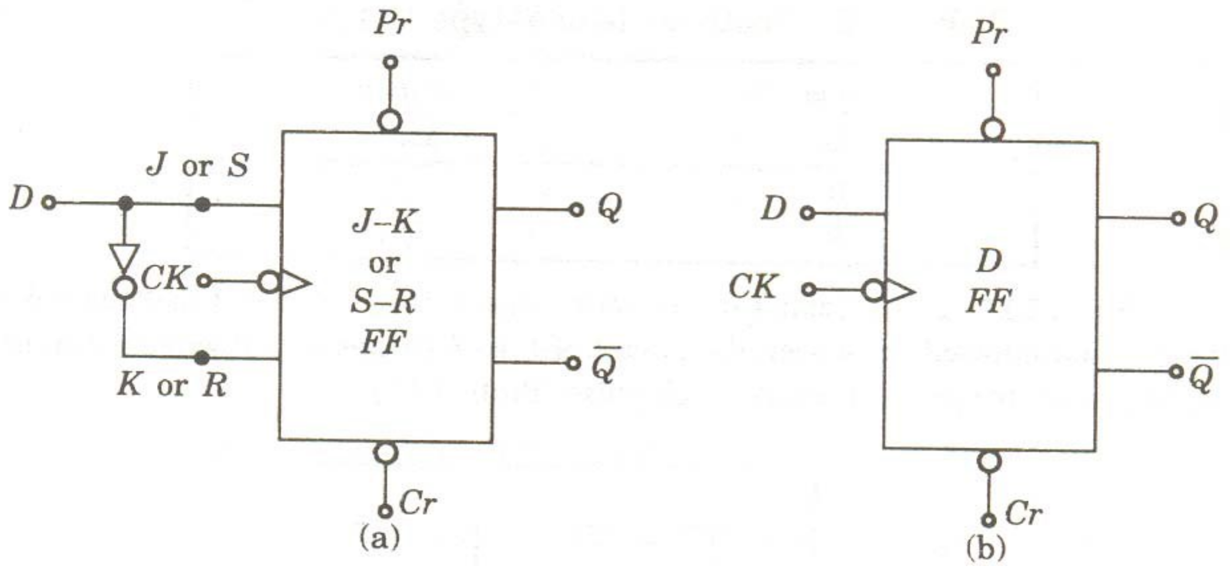
Data inputs		Outputs		Inputs to S-R FF		Output
J_n	K_n	Q_n	$\overline{Q}_n$	S_n	R_n	Q_{n+1}
0	0	0	1	0	0	0] = Q_n
0	0	1	0	0	0	1
1	0	0	1	1	0	1] = 1
1	0	1	0	0	0	1
0	1	0	1	0	0	0] = 0
0	1	1	0	0	1	0
1	1	0	1	1	0	0] = $\overline{Q}_n$
1	1	1	0	0	1	1

Inputs		Output
J_n	K_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	$\overline{Q}_n$



D-TYPE FLIP-FLOP

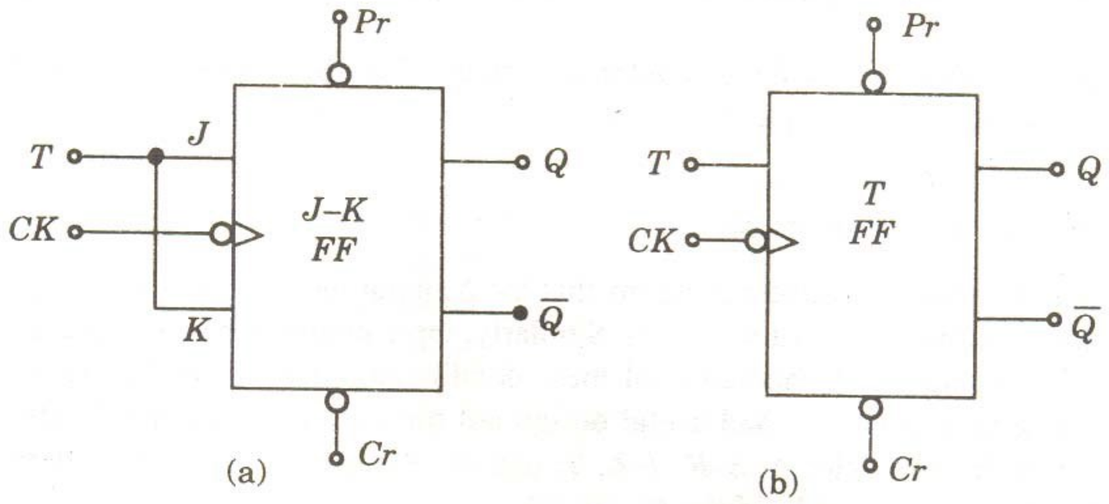
If we use only the middle two rows of the truth table of the S-R or J-K FLIP-FLOP, we obtain a D-type FLIP-FLOP as shown in Fig. below. It has only one input referred to as D-input or data input. Its truth table is given in Table from which it is clear that the output Qn+1 at the end of the clock pulse equals the input Dn before the clock pulse.



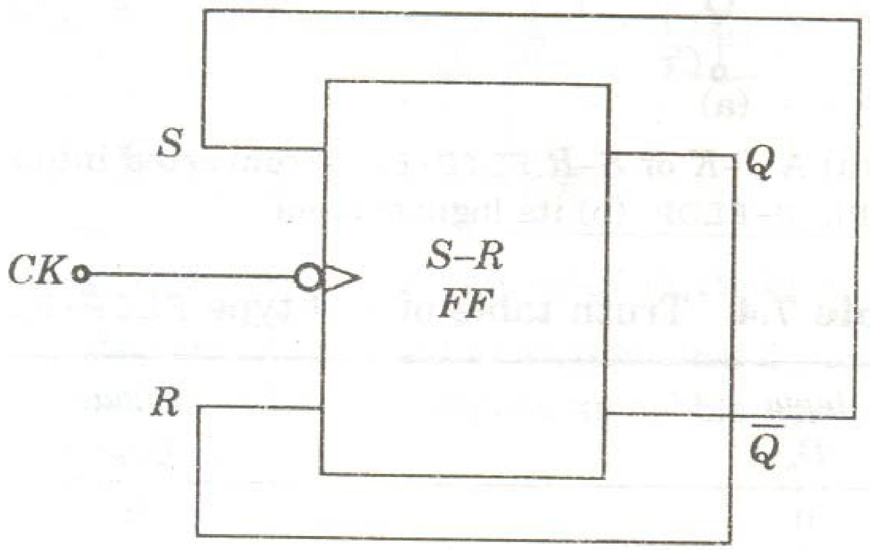
Input D_n	Output Q_{n+1}
0	0
1	1

T-TYPE FLIP-FLOP

In a J-K FLIP-FLOP, if J = K, the resulting FLIP-FLOP is referred to as a T-type FLIP-FLOP and is shown in Fig. It has only one input, referred to as T-input. Its truth table is given in Table from which it is clear that if T = 1 it acts as a toggle switch. For every clock pulse, the output Q changes.



Input T_n	Output Q_{n+1}
0	Q_n
1	$\overline{Q}_n$



EXCITATION TABLE OF FLIP-FLOP

The truth table of a FLIP-FLOP is also referred to as the *characteristic table* and specifies the operational characteristic of the FLIP-FLOP.

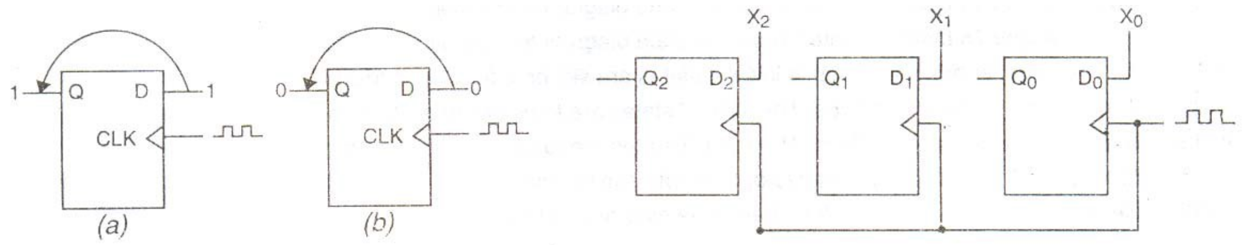
Present State	Next State	S-R S_n	FF R_n	J-K J_n	FF K_n	T-FF T_n	D-FF D_n
0	0	O	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	1	0
1	1	×	0	×	0	0	1

Registers

A register is device capable of storing a bit. The data can be serial or parallel. A register can convert a data from serial to parallel and vice versa. Shifting the digits to left and right is an important aspect of arithmetic operations. In this chapter we discuss these concepts.

BUFFER REGISTER

A register is used for storing and shifting data entered into it from an external source. We know that a flip flop is the basic storage element in digital system. Fig.below explains the concept of storing a 1 or 0 in a flip flop. A 1 is applied to the input of flip flop and a clock pulse is applied. 1 is stored by setting the flip flop. Even when 1 is removed from the input, the flip flop remains in the set state storing 1. If a 0 is applied at the input, 0 is stored in the flip flop on the application of clock pulse. For storing more bits we need more flip flops. Each state of a flip flop has one bit of storage capacity. Thus the number of stages is equal to the storage capacity. A buffer register is the simplest type of register. It can only store a digital word.



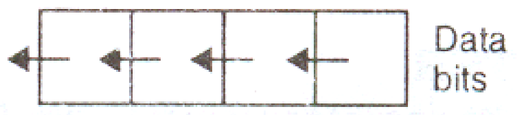
The X bits set the flip flops for loading. When the first positive clock edge arrives the output becomes

Q2 Q1 Q0 =X2 X1 X0

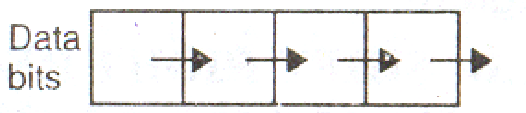
The buffer register is too primitive to be of any use. A control over the X bits is needed.

BASIC SHIFT OPERATIONS

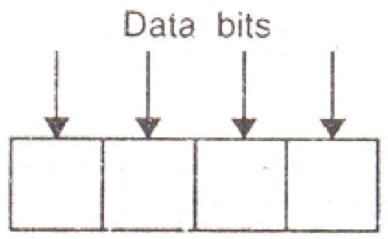
A simple example of shift operation is that in a calculator. Suppose we have to enter 356 in the calculator. First we press and release 3. The digit 3 appears in the display. Next we press and release 5. The digit 3 is shifted one place to the left and 5 appears on the extreme right. As we press and release 6, the digit 3 and 5 shift to the left and 6 appears in the extreme right position. This simple example illustrates two characteristics of a shift register. (1). It is a temporary memory and holds the number displayed. (2). When we press a new digit on the keyboard, the earlier number is shifted to the left. Thus, shift register has memory and shifting characteristics.
The basic shift operations are :
(a) serial shift left, then out
(b) serial shift right, then out
(c) parallel shift in
(d) parallel shift out
(e) rotate left
(f) rotate right.



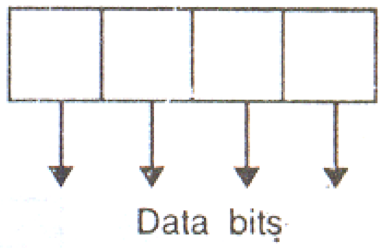
(a) Serial shift left, then out



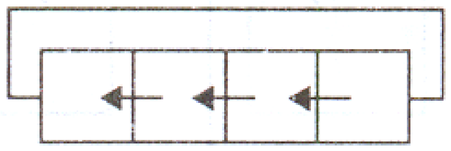
(b) Serial shift right, then out



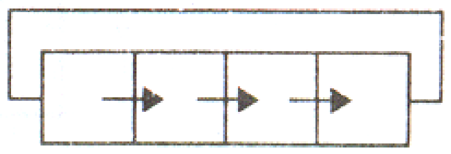
(c) Parallel shift in



(d) Parallel shift out



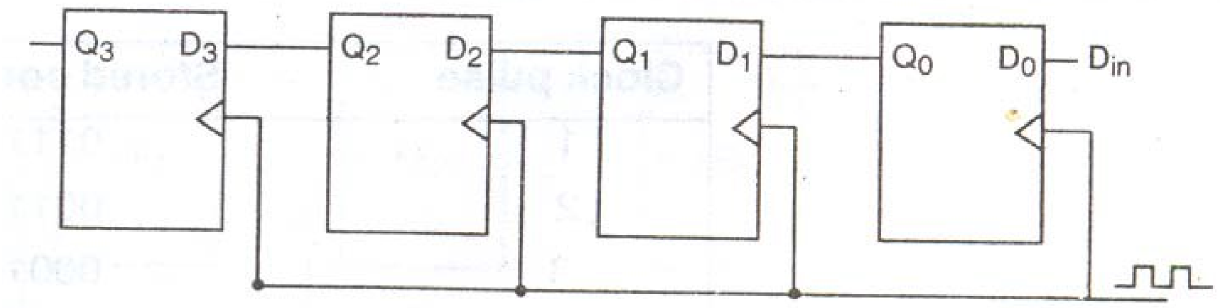
(e) Rotate left



(f) Rotate right

SHIFT LEFT REGISTER

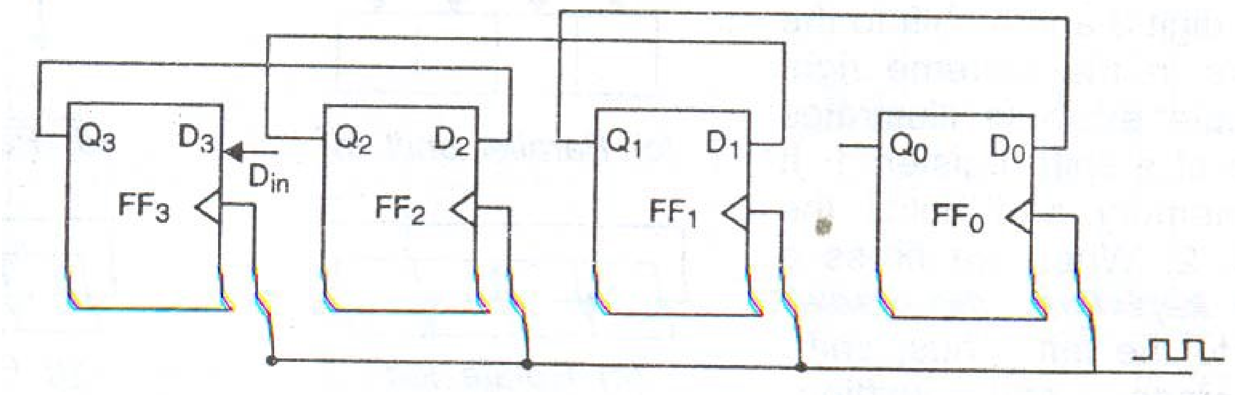
Figure shows a shift left register. It uses D flip flops. The circuit shown has positive edge triggering. It is a 4 bit register using 4 flip flops FF0, FF1, FF2 and FF3. Din is the input to FF0. Output of FF0 is Q0 and is fed to D1. Similarly Q1 is fed to D2 and Q2 to D3.
All the flip flops are clocked together by the clock pulse.



Initially Q3 Q2 Q1 = 0000
At the first positive edge of clock pulse FF0 is set and then Q3 Q2 Q1 Q0 = 0001
At the second positive edge of clock pulse FF1 is set and then Q3 Q2 Q1 Q0 = 0011
The positive edge of the third pulse sets FF2 and then Q3 Q2 Q1 Q0 = 0111
Finally, the positive edge of fourth clock pulse sets FF3 and then Q3 Q2 Q1 Q0 = 1111
As long as Din = 1, the stored word cannot change further. Now let Din be changed to 0. Then the successive clock pulses produce the following stored contents.

SHIFT RIGHT REGISTER

Figure shows a shift right register using D flip flops. As the name suggests the stored contents are shifted right on each clock pulse. The Q output is connected to the D input of preceding flip flop. At the arrival of each positive edge of clock shift right operation occurs. Let Din = 1 and Q3 Q2 Q1 Q0 = 0



The positive edge of first clock pulse sets up flip flop FF3 and
Q3 Q2 Q1 Q0 = 1000
The positive edge of second clock pulse makes the stored contents as

Q3 Q2 Q1 Q0 = 1100
The positive edge of third clock pulse gives
Q3 Q2 Q1 Q0 = 1110
and the positive edge of fourth clock pulse gives
Q3 Q2 Q1 Q0 = 1111
After this the stored contents remain the same till Din = 1.

SHIFT REGISTER OPERATIONS

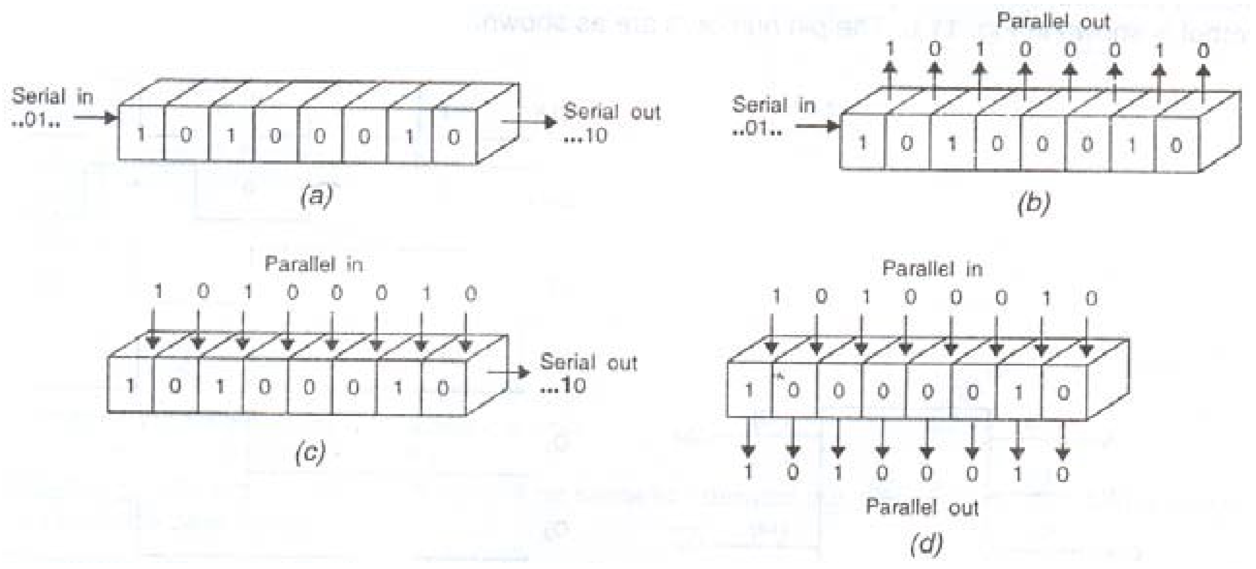
One method to describe the operation of shift register is the method of loading in and reading from the storage bits. There could be 4 such operations :

(a) **Serial in – Serial out :** The data is loaded into and read from the shift register serially. [Fig. (a)]

(b) Serial in – **Parallel out** : The data is loaded into the register serially but read in parallel (i.e., data is available from all bits simultaneously. [Fig.(b)].

(c) **Parallel in – Serial out :** The data is loaded in parallel, i.e., the bits are entered simultaneously in their respective stages and read serially. [Fig. (c)]

(d) **Parallel in – Parallel out** : The data is loaded and read from the register in paralle,
i.e., all bits are loaded simultaneously and read simultaneously. [Fig. (d)]

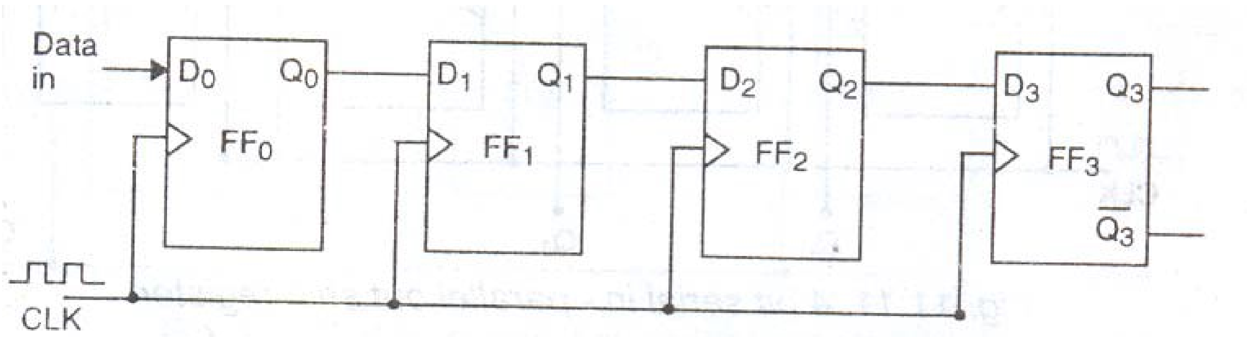


SERIAL IN – SERIAL OUT SHIFT REGISTER

Figure shows a 4 bit serial in – serial out shift register consisting of four D flip flops FF0, FF1, FF2 and FF3. As shown it is a positive edge triggered device. We study the working of this register for the data 1010 in the following steps :

1. Bit 0 is entered into data input line. D0 = 0, first clock pulse is applied, FF0 is
reset and stores 0.
2. Next bit 1 is entered. Q0 = 0, since Q0 is connected to D1, D1 becomes 0.

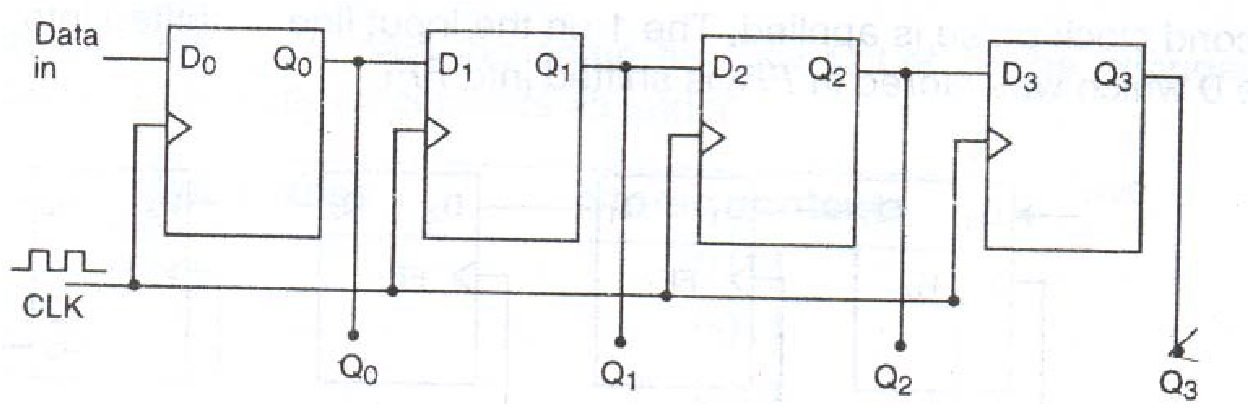
3. Second clock pulse is applied, The 1 on the input line is shifted into FF0 because FF0 sets. The 0 which was stored in FF0 is shifted into FF1.



4. Next bit 0 is entered and third clock pulse applied. 0 is entered into FF0 1 stored in FF0 is shifted to FF1 and 0 stored in FF1 is shifted to FF2.
5. Last bit 1 is entered and 4th clock pulse applied. 1 is entered into FF0, 0 stored in FF0 is shifted to FF1, 1 stored in FF1 is shifted to FF2 and 0 stored in FF2 is shifted to FF3. This completes the serial entry of 4 bit data into the register. Now the LSB 0 is on the output Q3.
6. Clock pulse 5 is applied. LSB 0 is shifted out. The next bit 1 appears on Q3 output.
7. Clock pulse 6 is applied. The 1 on Q3 is shifted out and 0 appears on Q3 output.
8. Clock pulse 7 is applied. 0 on Q3 is shifted out. Now 1 appears on Q3 output.
9. Clock pulse 8 is applied. 1 on Q3 is shifted out.

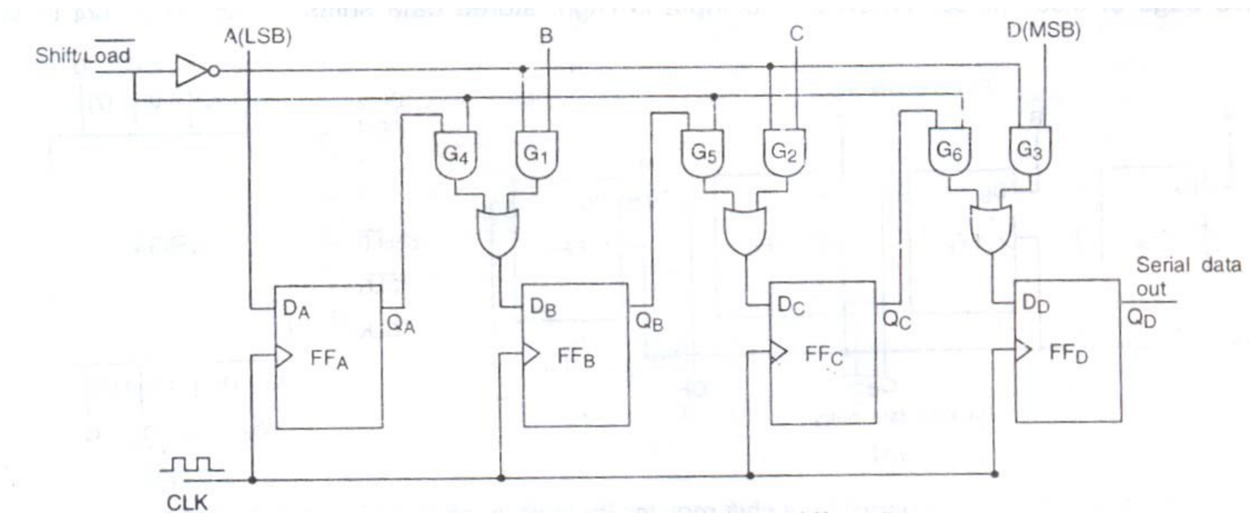
SERIAL IN – PARALLEL OUT SHIFT REGISTER

The data bits are entered serially but they are available at their respective positions simultaneously. Figure shows the circuit of 4 bit serial in – parallel out shift register using D flip flops. Q0, Q1, Q2 and Q3 are the output terminals and data is available at all of them together.



PARALLEL IN –SERIAL OUT SHIFT REGISTER

The data bits are entered simultaneously and taken out serially. Figure shows such a shift register with 4 bits. It uses D flip flops and 4 data input lines A, B, C, D. Moreover, it has a shift/ Load input which allows 4 bits of data to be entered into the register simultaneously. When shift/ Load is Low,



COUNTERS

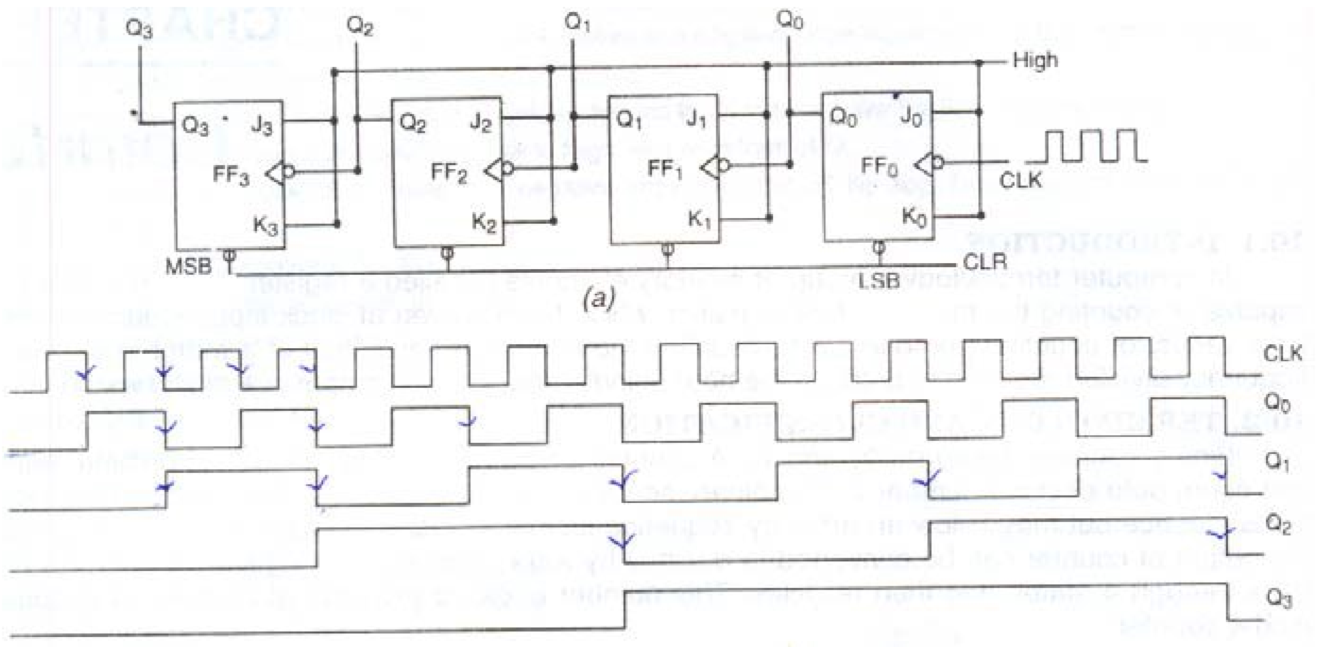
In computer terminology a group of memory elements is called a register. A counter is a register capable of counting the number of clock pulses which have arrived at clock input. Counters are used for a variety of counting purposes, e.g., counting the number of revolution of a motor in a given time, frequency division required to produce the hour, minute and seconds output in a digital watch etc.

TERMINOLOGY AND CLASSIFICATION

Binary counters generate 0s and 1s. A counter for counting 4 bits will start at decimal number 0 and count upto decimal number 3. The binary sequence is 00, 01, 10, 11. The counter may not have this sequence but may follow an arbitrary sequence also. If a decimal output of the count is required, the output of counter can be converted to decimal by a decoder. In counting from 0 to 3, the counter goes through 4 states and then recycles. This number is called modulus of counter. This counter is mod-4 counter. We know that each flip flop has 2 possible states. If a counter has N flip flops, it has a maximum of 2N states. Each of these 2N states has a unique combination of 0s and 1s stored in the different flip flops. A counter with N flip flops can have a maximum modulus of 2N. However, the actual number of states used may be less than maximum. If 4 flip flops are used, the maximum modulus can be 24 or 16. But a counter for counting upto 10 is known as decade or mod 10 counter though it has 4 flip flops and has a maximum modulus of 16. Counters are classified as a synchronous and asynchronous. Synchronous counter is clocked such that all the flip flops are clocked at the same time. For this purpose the clock terminal is connected to each stage of the counter. Asynchronous means that flip flops, in the counter, do not change state exactly at the same instant. In asynchronous counter the clock pulses are not connected directly to clock input of each flip flop in the counter. The counters mostly use JK flip flops connected properly. The word binary counter means a counter which counts and produces binary output 0000, 0001, 0010, 0011 etc. It goes through a binary sequence depending on its modulus.

CIRCUIT AND WORKING OF RIPPLE COUNTER:

Fig (a) shows the circuit of a bit ripple counter consisting of 4 edge triggered JK flip flop. As indicated by small circles at the CLK input of flip flops, the triggering occurs when CLK input gets a negative edge. Q0 is the least significant bit (LSB) and Q3 is the most significant bit (MSB). The flip flops are connected in series. The Q0 output is connected to CLK terminal of second flip flop. The Q1 output is connected to CLK terminal of third flip flop and so on. By adding more flip flop, a counter of any length can be built. It is known as a ripple counter because the carry moves through the flip flops like a ripple on water. Initially, CLR is made low and all flip flops reset giving an output Q = 0000. When CLR becomes high, the counter is ready to start. As LSB receives its clock pulse, its output changes from 0 to 1 and the total output Q = 0001. When second clock pulse arrives, Q0 resets and carries (i.e., Q0 goes from 1 to 0 and, second flip flop will receive CLK input). Now the output is Q = 0010. The third CLK pulse changes Q0 to 1 giving a total output Q = 0011. The fourth CLK pulse causes Q0 to reset and carry and Q1 also resets and carries giving a total output Q = 0100 and the process goes on. The action is shown in Table. The number of output states of a counter are known as modulus (or mod). A ripple counter with 4 flip flops can count from 0 to 15 and is, therefore, known as mod-16 counter while one with 6 flip flops can count from 0 to 63 and is a mod-64 counter and so on.

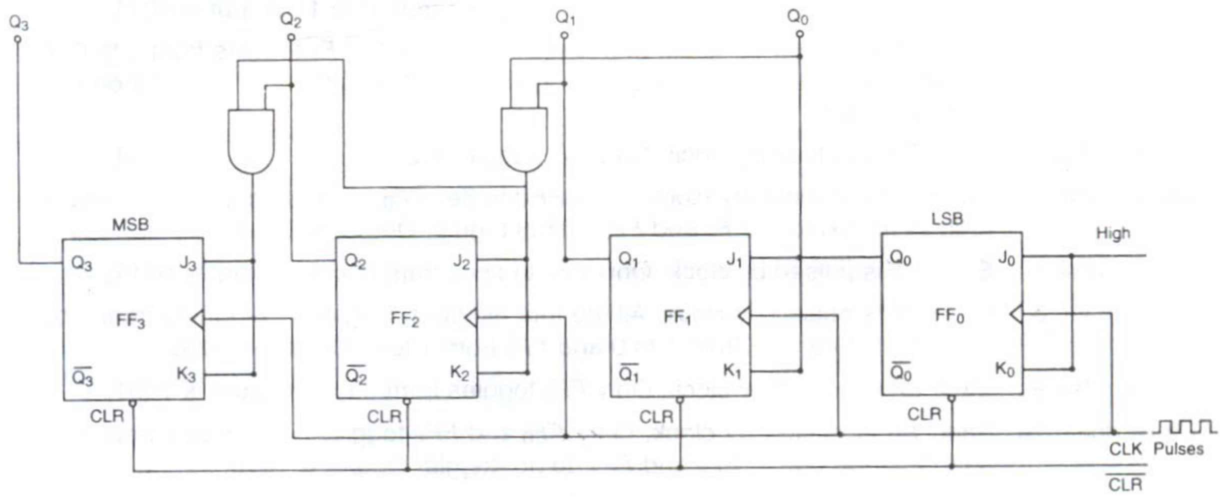


Count	Q_3	Q_2	Q_1	Q_0	Count	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

Ripple counters are simple to fabricate but have the problem that the carry has to propagate through a number of flip flops. The delay times of all the flip flops are added. Therefore, they are very slow for some applications. Another problem is that unwanted pulses occur at the output of gates. The timing diagram is shown in Fig (b). FF0 is LSB flip flop and FF3 is the MSB flip flop. Since FF0 receives each clock pulse, Q0 toggles once per negative clock edge as shown in Fig. (b). The remaining flip flops toggle less often because they receive negative clock edge from preceding flip flops. When Q0 goes from 1 to 0, FF1 receives a negative edge and toggles. Similarly, when Q1 changes from 1 to 0, FF2 receives a negative edge and toggles. Finally when Q2 changes from 1 to 0, FF3 receives a negative edge and toggles. Thus whenever a flip flop resets to 0, the next higher flip flop toggles. From the above discussion it is evident why this counter is known as ripple counter. As the 16th clock pulse is applied, the trailing edge of 16th pulse causes a transition in each flip flop. Q0 goes from High to Low, this causes Q1 go from High to Low which causes Q2 to go from High to Low which causes Q3 to go from High to Low. Thus the effect ripples through the counter. It is the delay caused by this ripple which results in a limitation on the maximum frequency of the input signal.

SYNCHRONOUS COUNTER

In a synchronous counter, all the flip flops are clocked together. Fig. shows a synchronous counter having positive edge triggered JK flip flops. Since all the flip flops are clocked together, the delay time is less. The flip flop corresponding to lest significant bit (LSB) has its input JK fed from voltage + VCC. Therefore, it responds to each positive clock edge. However, the other three flip flops can respond to the positive clock pulse under some certain conditions. The Q1 flip flop toggles on positive clock edge only if Q0 is 1. The Q2 flip flop toggles on positive clock edge only when Q0 and Q1 are (due to presence of AND circuit) and so on.

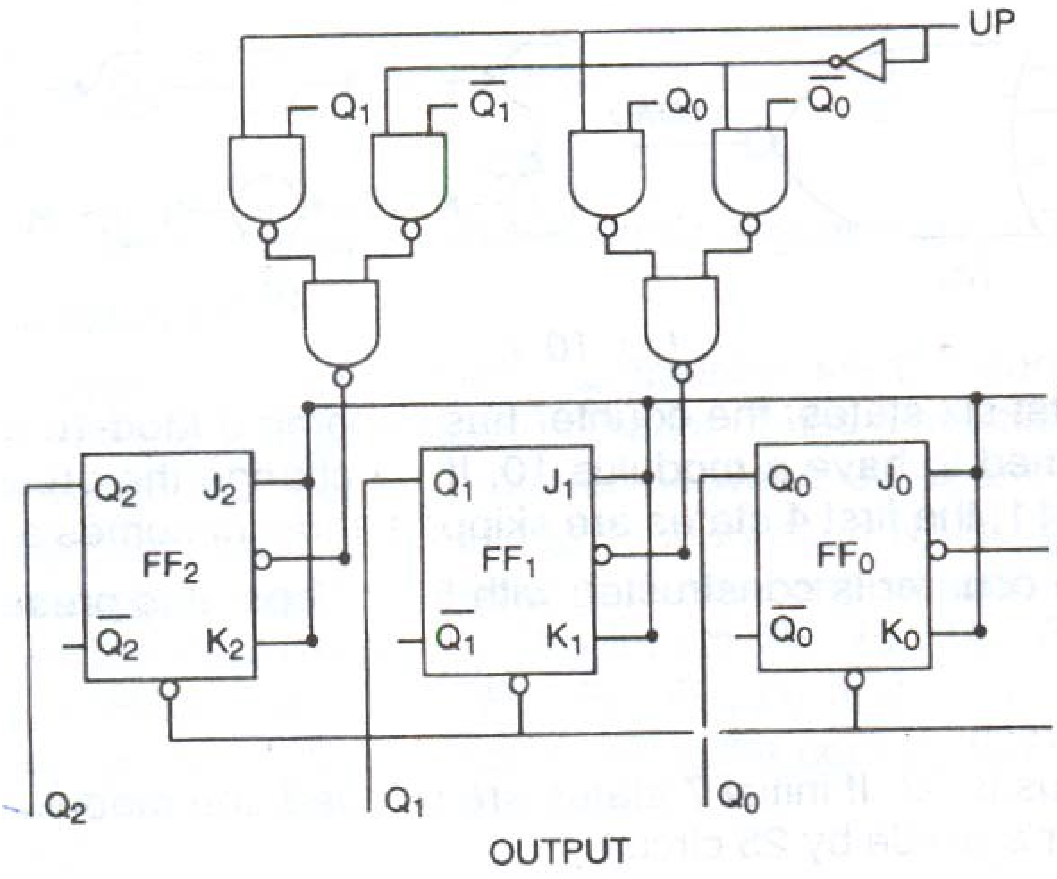


The circuit action and output for various clock pulses are as under:
Clock pulse 1 : Each FF is pulsed by clock. However, only FF0 can toggle because it is the only one with 1s applied to both J and K inputs. FF0 goes from 0 to 1.
Output is 0001.
Clock pulse 2 : Each FF is pulsed by clock. FF0 and FF1 have 1s applied to both J and K inputs. Therefore, FF0 and FF1 toggle. FF1 goes from 0 to 1 and FF0 goes from 1 to 0. Output is 0010.
Clock pulse 3 : Each FF is pulsed by clock. Only FF0 toggles from 0 to 1. Output is 0011.
Clock pulse 4 : Each FF is pulsed by clock. FF0 toggles from 1 to 0, FF1 toggles from 1 to 0. FF2 toggles from 0 to 1. FF3 does not toggle. (Its JK inputs do not have 1s on then). Output is 0100.
Clock pulse 5 : Each FF is pulsed by clock. Only FF0 toggles from 0 to 1. Output is 0101.
Clock pulse 6 : Each FF is pulsed by clock. Two FF toggle. FF0 toggle from 1 to 0 and FF1 toggles from 0 to 1. FF2 and FF3 do not toggle. Output is 0110.
Clock pulse 7 : Each FF is pulsed by clock. Only FF0 toggles from 0 to 1. Output is 0111.
Clock pulse 8 : Each FF is pulsed by clock. All the four flip flops toggle. FF3 toggles from 0 to 1, FF2 from 1 to 0, FF1 from 1 to 0 and FF0 from 1 to 0. Output is 1000.
Clock pulse 9 : Each FF is pulsed by clock. Only FF0 toggles from 0 to 1. Output is 1001.
Clock pulse 10 : Each FF is pulsed by clock. Only FF0 and FF1 toggle. FF0 toggles from 1 to 0 and FF2 from 0 to 1. FF2 and FF3 do not toggle. Output is 1010.
Clock pulse 11 : Each FF is pulsed by clock. Only FF0 toggles from 0 to 1. FF3, FF2 and FF1 do not toggle. Output is 1011.
Clock pulse 12 : Each FF is pulsed by clock. FF3 does not toggle. FF2 toggles from 0 to 1, FF1 from 1 to 0 and FF0 from 1 to 0. Output is 1100.
Clock pulse 13 : Each FF is pulsed by clock. FF3, FF2, FF1 do not toggle. Only FF0 toggles from 0 to 1. Output is 1101.
Clock pulse 14 : Each FF is pulsed by clock. FF3 and FF2 do not toggle. FF1 toggles from 0 to 1 and FF0 toggles from 1 to 0. Output is 1110.

Clock pulse 15 : Each FF is pulsed by clock. FF3, FF2, FF1 do not toggle. Only FF0 toggles from 0 to 1. Output is 1111.
Clock pulse 16 : Each FF is pulsed by clock. All FFs toggle from 1 to 0. Output is 0000.
Thus 1 cycle of operation is completed. The 4 bit synchronous counter of Fig. 10.13 is Mod-16 counter.

UP-DOWN COUNTER

As the name suggests an UP-DOWN counter can count either upwards from 0000 onwards or from the highest number (equal to modulus of counter) to 0000. Thus a 4 bit updown counter can count from 0000 to 1111 and also from 1111 to 0000.

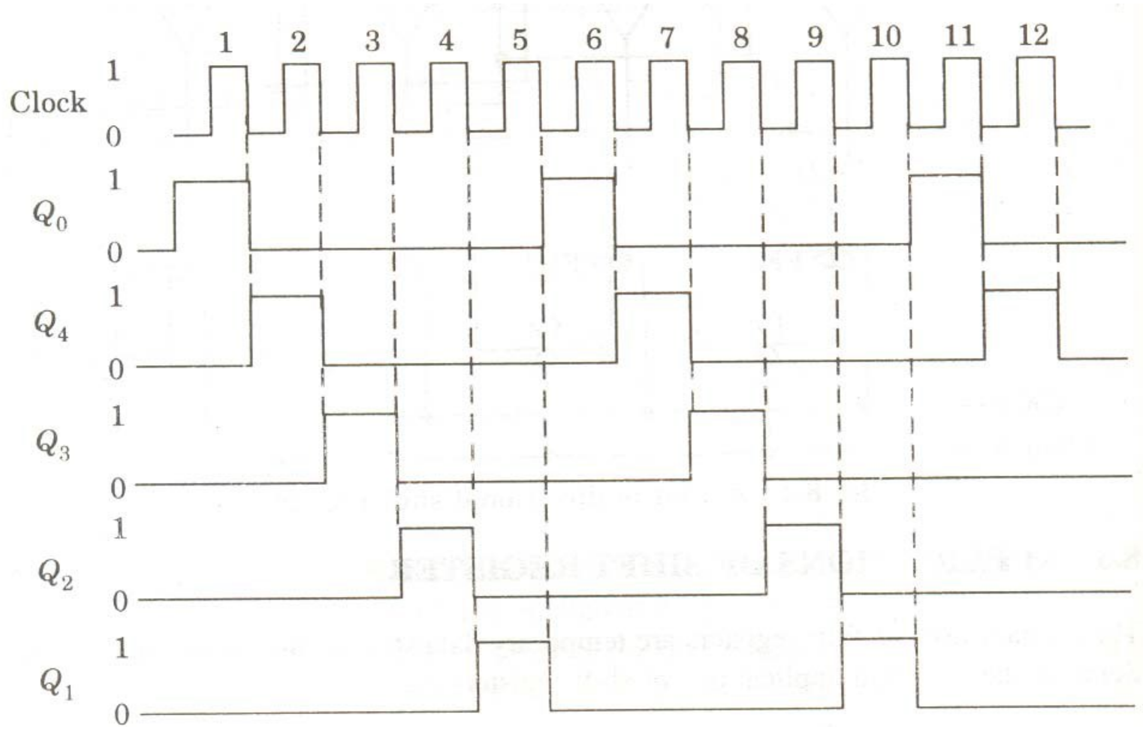


DIFFERENCE BETWEEN ASYNCHRONOUS AND SYNCHRONOUS COUNTER

The term asynchronous means that the counter is clocked in such a way that all the flip flops of the counter do not receive clock pulses at the same time. Ripple counter is an asynchronous counter. The clock pulses drive the clock input of the first flip flop. But the clock input of the second flip flop is driven by Q output of the first flip flop. The clock input of third flip flop is driven by the Q output of the second flip flop and so on. This results in propagation delay in each flip flop. In a 4 bit ripple counter having flip flops with propagation delay of 10 ns each, there will be a total propagation delay of 40 ns. This may be undesirable in many cases. The term synchronous means that all flip flops are clocked simultaneously. The clock pulses drive the clock input of all the flip flops together so that there is no propagation delay.

RING COUNTER

If the serial output Q0 of the shift register of Fig. is connected back to the serial input, then an injected pulse will keep circulating. This circuit is referred to as a *ring counter.* The pulse is injected by entering 00001 in the parallel form after clearing the FLIP-FLOPs. When the clock pulses are applied, this 1 circulates around the circuit. The waveforms at the Q outputs are shown in Fig. The outputs are sequential non-overlapping pulses which are useful for control-state counters, for stepper motor (which rotates in steps) which require sequential pulses to rotate it from one position to the next, etc.



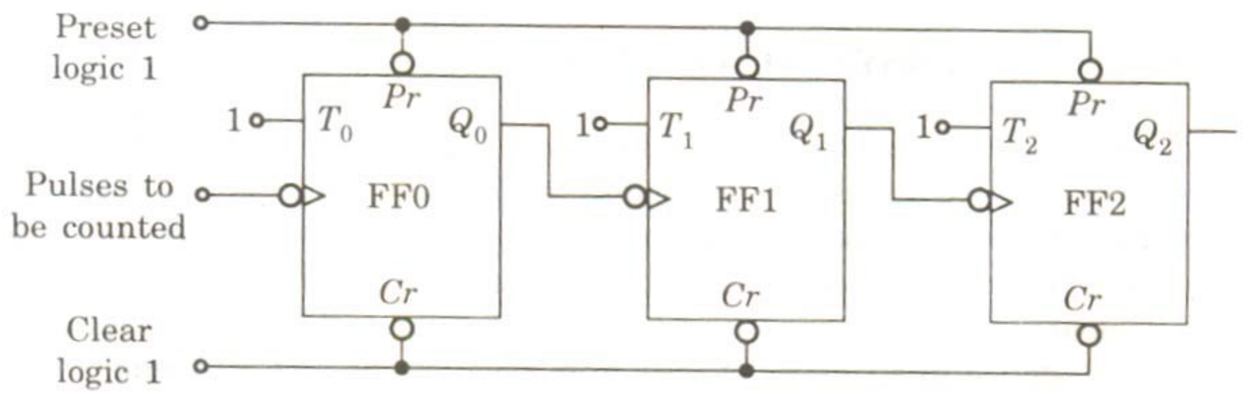
This circuit can also be used for counting the number of pulses. The number of pulses counted is read by noting which FLIP-FLOP is in 1 state. No decoding circuitry is required. Since there is one pulse at the output for each of the N clock pulses, this circuit is referred to as a *divide-by-N counter* or an N : 1 *scalar.*

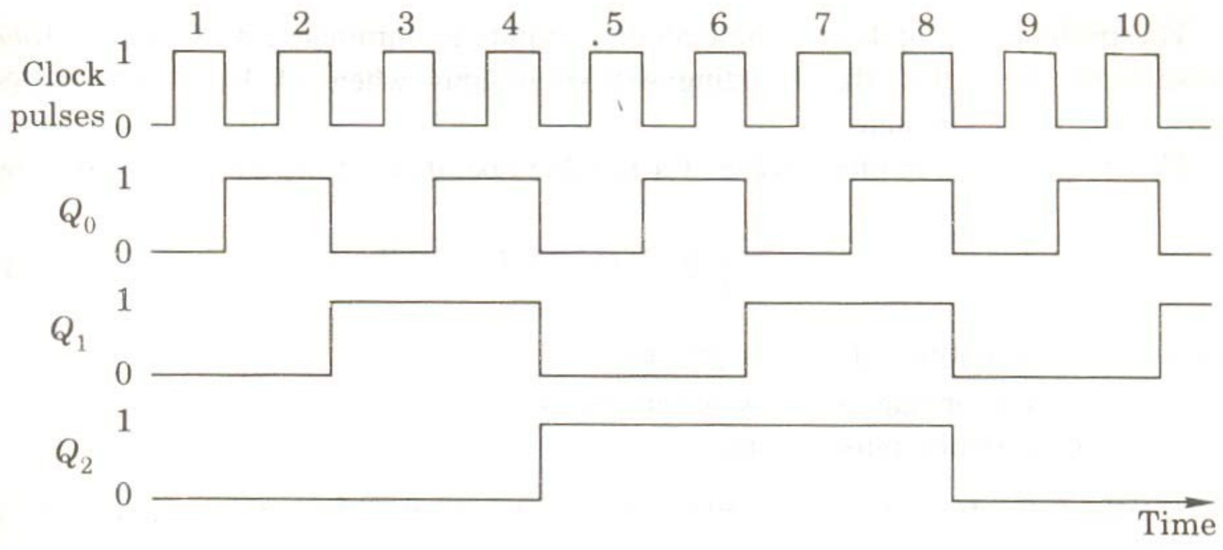
RIPPLE OR ASYNCHRONOUS COUNTER DESIGN

Consider the count sequence shown in Table 6.3. The number of states in this sequence is 8 which requires 3 FLIP-FLOPs (23 = 8) and Q2, Q1, and Q0 are the outputs of these FLIP-FLOPs. Assume master-slave FLIP-FLOPs.

Counter state	Count		
	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The output Q0 of the least-significant FLIP-FLOP, changes for every clock pulse. This can be achieved by using T-type FLIP-FLOP with T0 = 1. The output Q1 makes a transition (from 0 to 1 or 1 to 0) whenever Q0 changes from 1 to 0. Therefore, if Q0 is connected to the clock input of next T-type FLIP-FLOP, FF1 with T1 = 1, Q1 will change whenever Q0 goes from 1 to 0 (falling edge of clock pulse). Similarly, Q2 makes a transition whenever Q1 goes from 1 to 0 and this can be achieved by connecting Q1 to the clock input of the most-significant FLIP-FLOP, FF2 (and T2 = 1). The resulting circuit is shown in Fig. 6.8. The wave-forms of the outputs of the FLIP-FLOPs are shown in Fig. below.



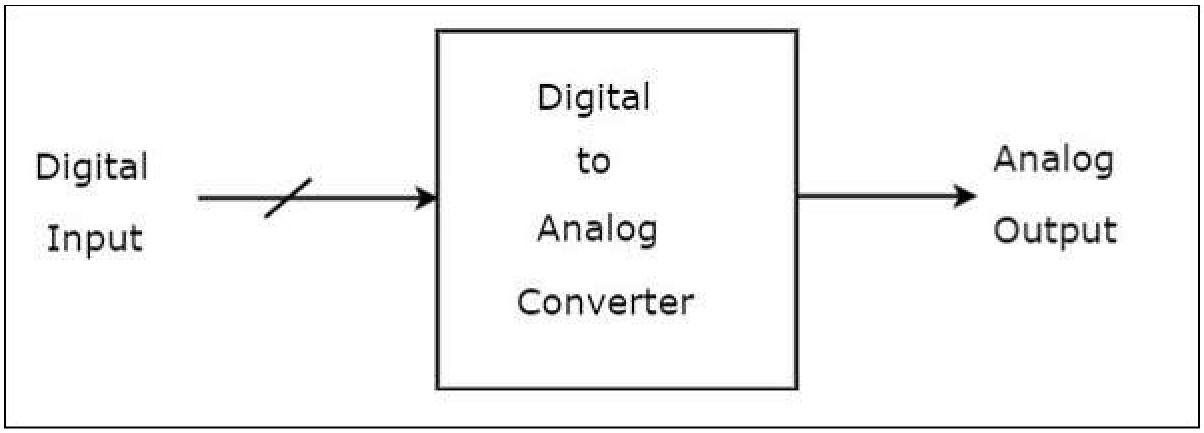

1
2
3
4
5
6
7
8
9
10
Clock pulses
1
0
Q_0
1
0
Q_1
1
0
Q_2
1
0
Time

Module-IV

A/D and D/AConverters

A Digital to Analog Converter (DAC) converts a digital input signal into an analog output signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1. This chapter deals with Digital to Analog Converters in detail.

The block diagram of DAC is shown in the following figure −



Types of DACs

There are two types of DACs

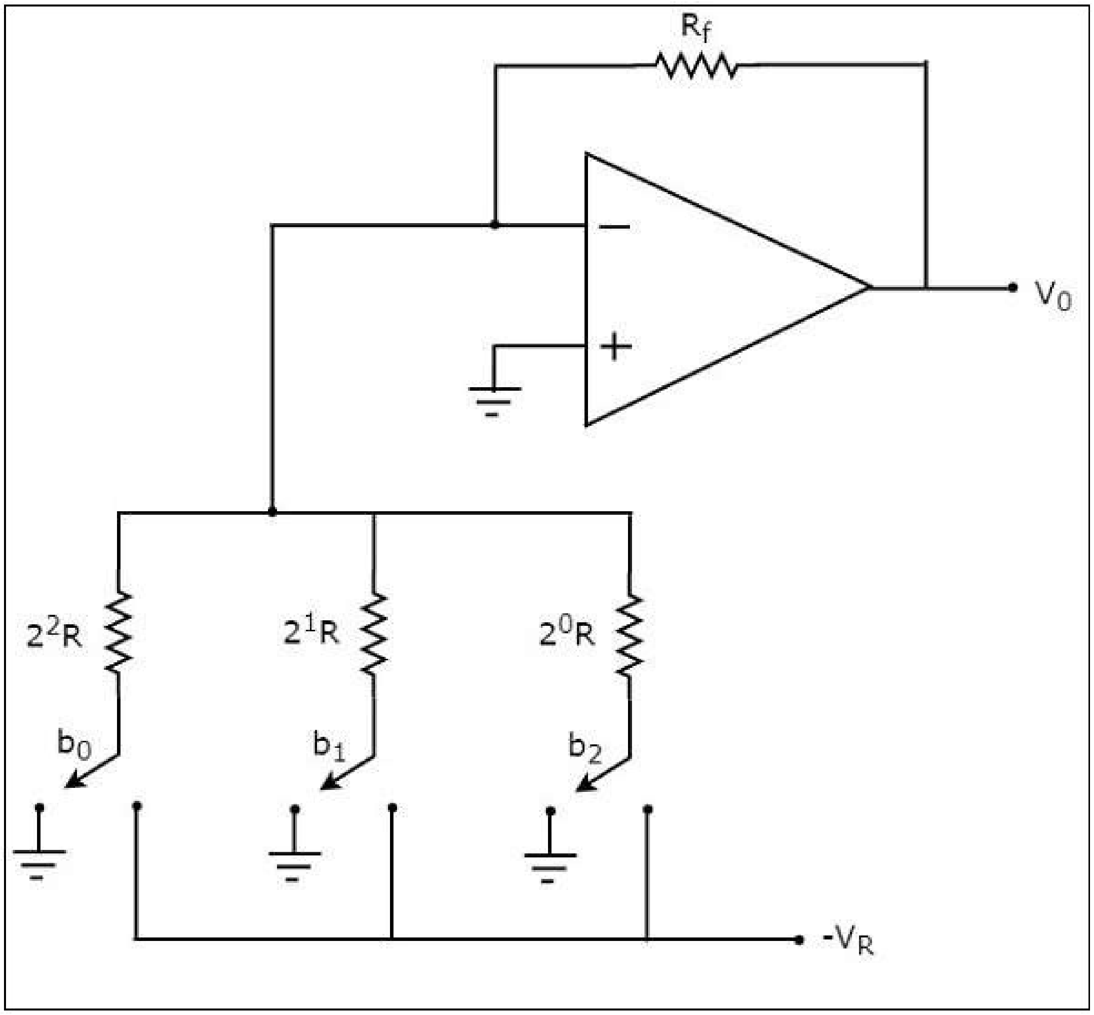
- Weighted Resistor DAC
- R-2R Ladder DAC

This section discusses about these two types of DACs in detail −

Weighted Resistor DAC

A weighted resistor DAC produces an analog output, which is almost equal to the digital (binary) input by using binary weighted resistors in the inverting adder circuit. In short, a binary weighted resistor DAC is called as weighted resistor DAC.

The circuit diagram of a 3-bit binary weighted resistor DAC is shown in the following figure −



Recall that the bits of a binary number can have only one of the two values. i.e., either 0 or 1. Let the 3-bit binary input is b2b1b0. Here, the bits b2 and b0 denote the Most Significant Bit (MSB) and Least Significant Bit (LSB) respectively. The digital switches shown in the above figure will be connected to ground, when the corresponding input bits are equal to '0'. Similarly, the digital switches shown in the above figure will be connected to the negative reference voltage, −VR−VR when the corresponding input bits are equal to '1'.

In the above circuit, the non-inverting input terminal of an op-amp is connected to ground. That means zero volts is applied at the non-inverting input terminal of op-amp.

According to the virtual short concept, the voltage at the inverting input terminal of opamp is same as that of the voltage present at its non-inverting input terminal. So, the voltage at the inverting input terminal's node will be zero volts.

The nodal equation at the inverting input terminal's node is:

$$\frac{0+V_R b_2}{2^0 R}+\frac{0+V_R b_1}{2^1 R}+\frac{0+V_R b_0}{2^2 R}+\frac{0-V_0}{R_f}=0$$

$$=> \frac{V_0}{R_f}=\frac{V_R b_2}{2^0 R}+\frac{V_R b_1}{2^1 R}+\frac{V_R b_0}{2^2 R}$$

$$=> V_0=\frac{V_R R_f}{R}\left\{\frac{b_2}{2^0}+\frac{b_1}{2^1}+\frac{b_0}{2^2}\right\}$$

Substituting, $R=2R_f$ *f* in above equation.

$$=> V_0=\frac{V_R R_f}{2R_f}\left\{\frac{b_2}{2^0}+\frac{b_1}{2^1}+\frac{b_0}{2^2}\right\}$$

$$=> V_0=\frac{V_R}{2}\left\{\frac{b_2}{2^0}+\frac{b_1}{2^1}+\frac{b_0}{2^2}\right\}$$

The above equation represents the output voltage equation of a 3-bit binary weighted resistor DAC. Since the number of bits is three in the binary (digital) input, we will get seven possible values of output voltage by varying the binary input from 000 to 111 for a fixed reference voltage, VR.

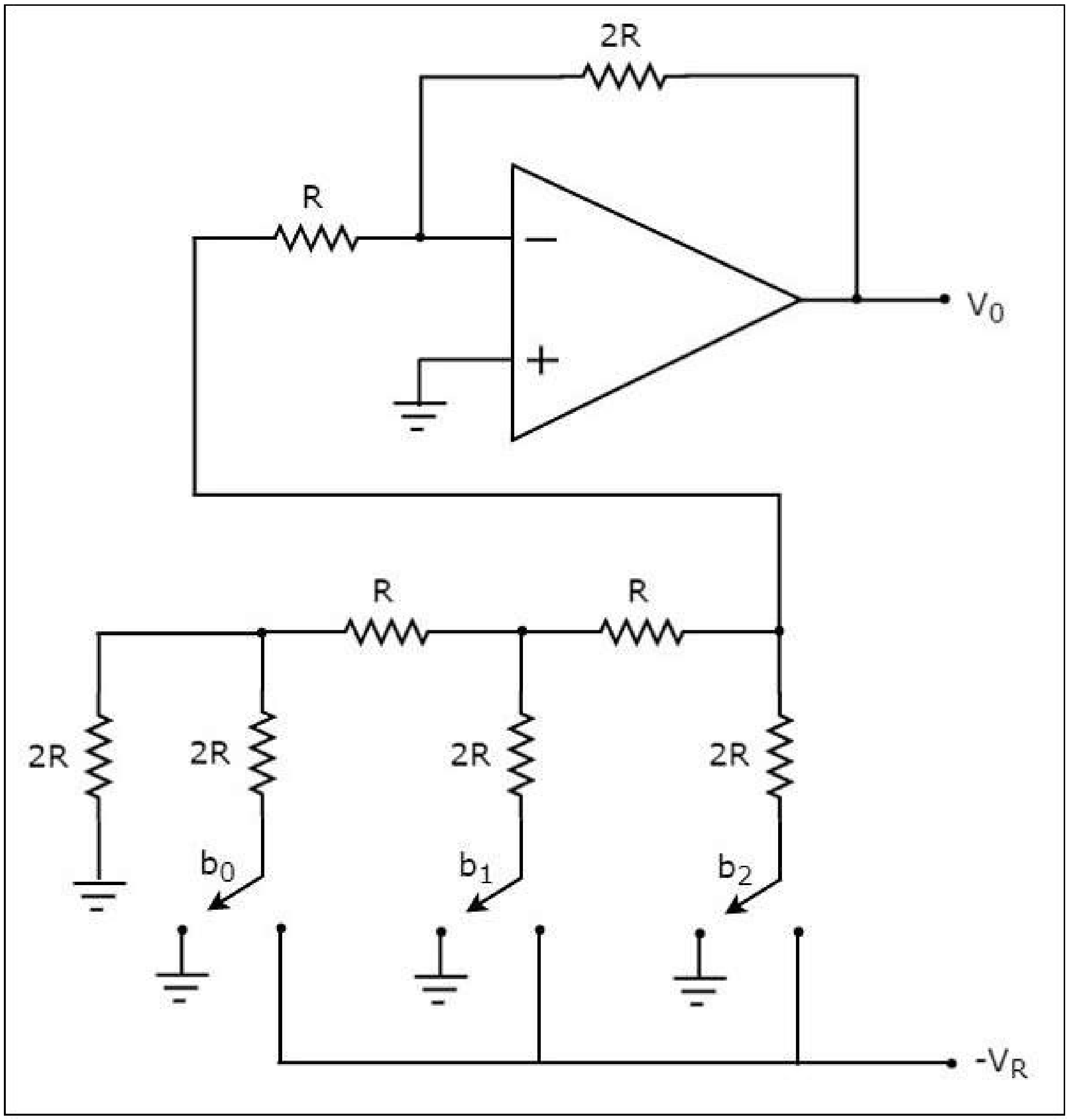
We can write the generalized output voltage equation of an N-bit binary weighted resistor DAC as shown below based on the output voltage equation of a 3-bit binary weighted resistor DAC.

$$=> V_0=\frac{V_R}{2}\left\{\frac{b_{N-1}}{2^0}+\frac{b_{N-2}}{2^1}+\ldots.+\frac{b_0}{2^{N-1}}\right\}$$

R-2R Ladder DAC

The R-2R Ladder DAC overcomes the disadvantages of a binary weighted resistor DAC. As the name suggests, R-2R Ladder DAC produces an analog output, which is almost equal to the digital (binary) input by using a R-2R ladder network in the inverting adder circuit.

The circuit diagram of a 3-bit R-2R Ladder DAC is shown in the following figure −

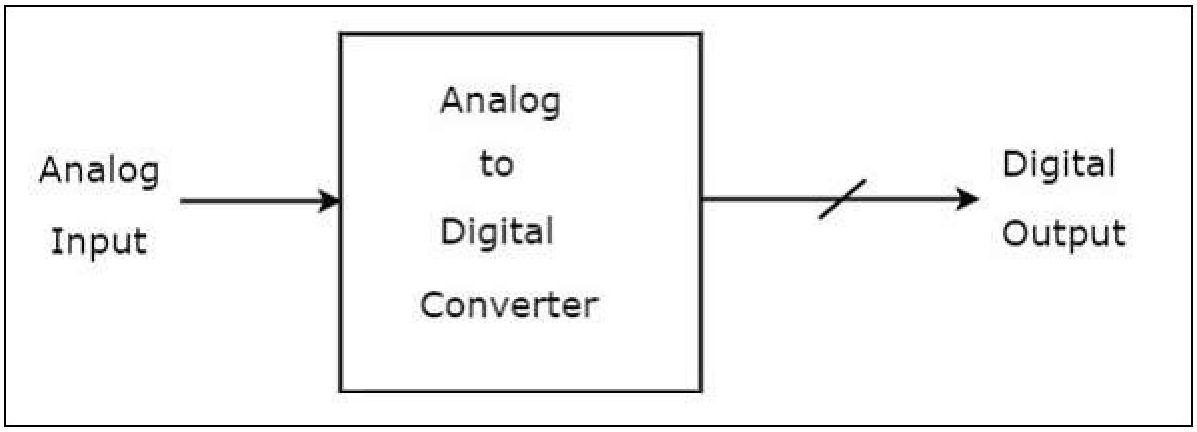


Recall that the bits of a binary number can have only one of the two values. i.e., either 0 or 1. Let the 3-bit binary input is b2b1b0. Here, the bits b2 and b0 denote the Most Significant Bit (MSB) and Least Significant Bit (LSB) respectively.
The digital switches shown in the above figure will be connected to ground, when the corresponding input bits are equal to '0'. Similarly, the digital switches shown in above figure will be connected to the negative reference voltage, VR when the corresponding input bits are equal to '1'.

It is difficult to get the generalized output voltage equation of a R-2R Ladder DAC. But, we can find the analog output voltage values of R-2R Ladder DAC for individual binary input combinations easily.

An Analog to Digital Converter (ADC) converts an analog signal into a digital signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1.

The block diagram of an ADC is shown in the following figure −



Observe that in the figure shown above, an Analog to Digital Converter (ADC) consists of a single analog input and many binary outputs. In general, the number of binary outputs of ADC will be a power of two.

There are two types of ADCs: Direct type ADCs and Indirect type ADC. This chapter discusses about the Direct type ADCs in detail.

If the ADC performs the analog to digital conversion directly by utilizing the internally generated equivalent digital (binary) code for comparing with the analog input, then it is called as Direct type ADC.

The following are the examples of Direct type ADCs −

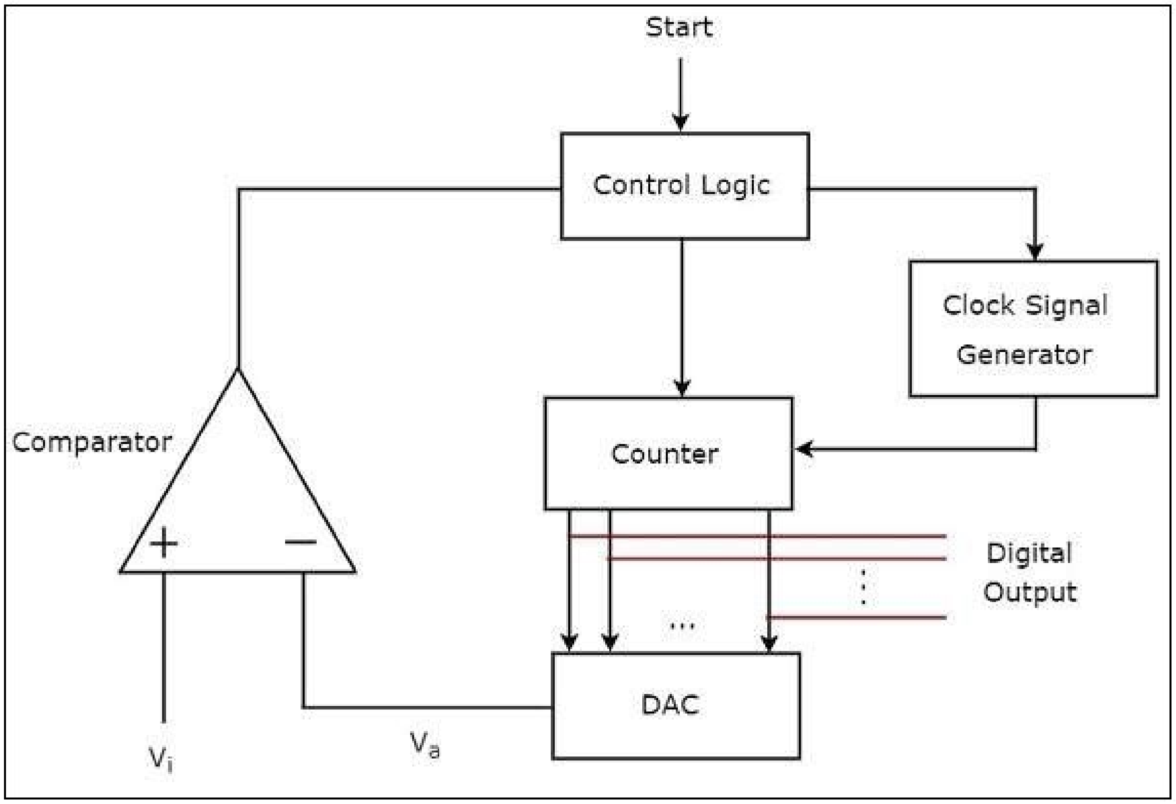
- Counter type ADC
- Successive Approximation ADC
- Flash type ADC

This section discusses about these Direct type ADCs in detail.

Counter type ADC

A counter type ADC produces a digital output, which is approximately equal to the analog input by using counter operation internally.

The block diagram of a counter type ADC is shown in the following figure



The counter type ADC mainly consists of 5 blocks: Clock signal generator, Counter, DAC, Comparator and Control logic.

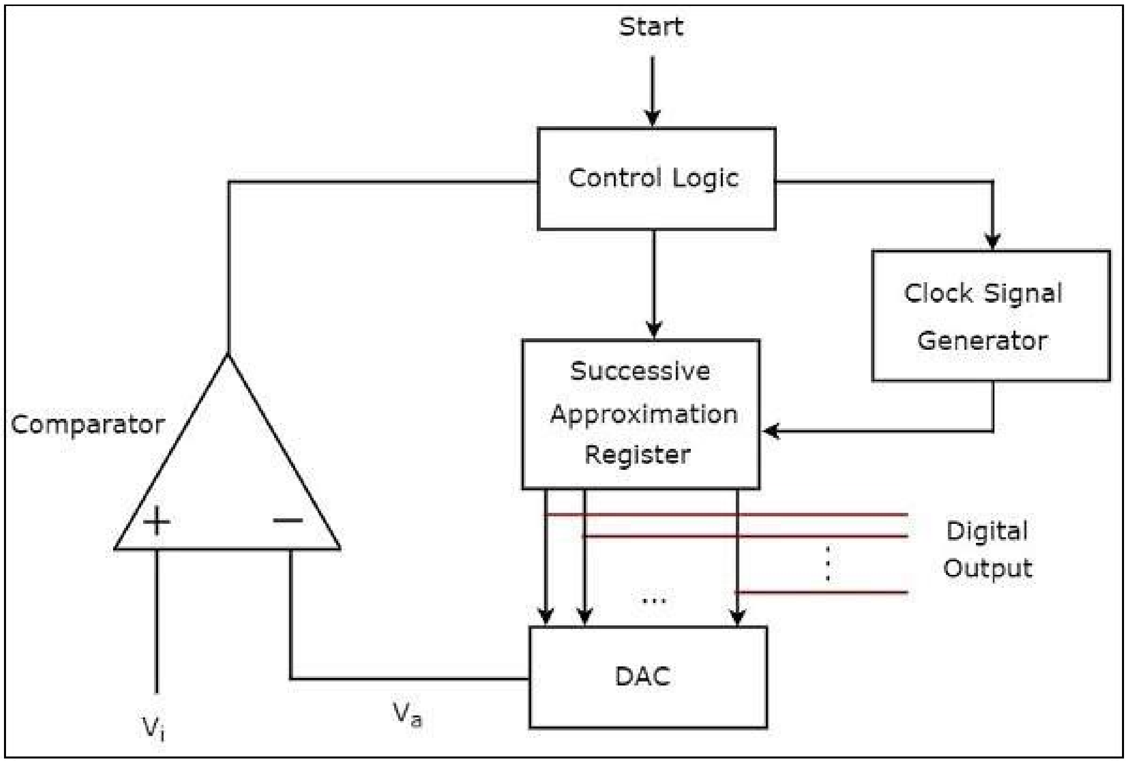
The working of a counter type ADC is as follows −

- The control logic resets the counter and enables the clock signal generator in order to send the clock pulses to the counter, when it received the start commanding signal.
- The counter gets incremented by one for every clock pulse and its value will be in binary (digital) format. This output of the counter is applied as an input of DAC.
- DAC converts the received binary (digital) input, which is the output of counter, into an analog output. Comparator compares this analog value, Va with the external analog input value Vi.
- The output of comparator will be '1' as long as Vi is greater than. The operations mentioned in above two steps will be continued as long as the control logic receives '1' from the output of comparator.
- The output of comparator will be '0' when Vi is less than or equal to Va. So, the control logic receives '0' from the output of comparator. Then, the control logic disables the clock signal generator so that it doesn't send any clock pulse to the counter.
- At this instant, the output of the counter will be displayed as the digital output. It is almost equivalent to the corresponding external analog input value Vi

Successive Approximation ADC

A successive approximation type ADC produces a digital output, which is approximately equal to the analog input by using successive approximation technique internally.

The block diagram of a successive approximation ADC is shown in the following figure



The successive approximation ADC mainly consists of 5 blocks− Clock signal generator, Successive Approximation Register (SAR), DAC, comparator and Control logic.

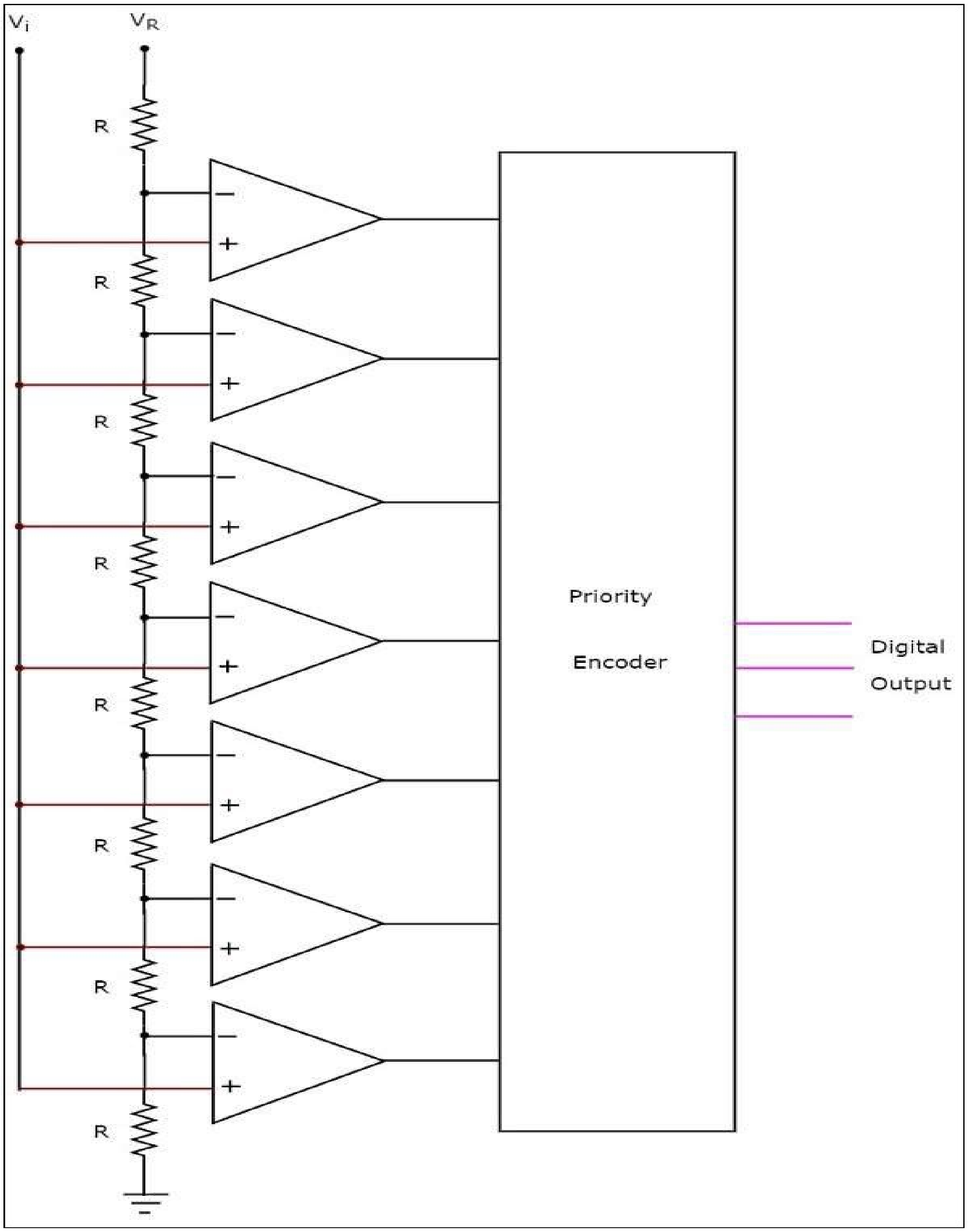
The working of a successive approximation ADC is as follows −

- The control logic resets all the bits of SAR and enables the clock signal generator in order to send the clock pulses to SAR, when it received the start commanding signal.
- The binary (digital) data present in SAR will be updated for every clock pulse based on the output of comparator. The output of SAR is applied as an input of DAC.
- DAC converts the received digital input, which is the output of SAR, into an analog output. The comparator compares this analog value Va with the external analog input value Vi
- The output of a comparator will be '1' as long as Vi is greater than Va Similarly, the output of comparator will be '0', when Vi is less than or equal to Va
- The operations mentioned in above steps will be continued until the digital output is a valid one.The digital output will be a valid one, when it is almost equivalent to the corresponding external analog input value Vi

Flash type ADC

A flash type ADC produces an equivalent digital output for a corresponding analog input in no time. Hence, flash type ADC is the fastest ADC.

The circuit diagram of a 3-bit flash type ADC is shown in the following figure −



The 3-bit flash type ADC consists of a voltage divider network, 7 comparators and a priority encoder.

The working of a 3-bit flash type ADC is as follows.

- The voltage divider network contains 8 equal resistors. A reference voltage VRVR is applied across that entire network with respect to the ground. The voltage drop across each resistor from bottom to top with respect to ground will be the integer multiples (from 1 to 8) of VR8VR8.
- The external input voltage Vi is applied to the non-inverting terminal of all comparators. The voltage drop across each resistor from bottom to top with respect to ground is applied to the inverting terminal of comparators from bottom to top.
- At a time, all the comparators compare the external input voltage with the voltage drops present at the respective other input terminal. That means, the comparison operations take place by each comparator parallelly.
- The output of the comparator will be '1' as long as Vi is greater than the voltage drop present at the respective other input terminal. Similarly, the output of comparator will be '0', when, Vi is less than or equal to the voltage drop present at the respective other input terminal.
- All the outputs of comparators are connected as the inputs of priority encoder. This priority encoder produces a binary code (digital output), which is corresponding to the high priority input that has '1'.
- Therefore, the output of priority encoder is nothing but the binary equivalent (digital output) of external analog input voltage, Vi

Module-V

Semiconductor memories and Programmable logic devices

MEMORY DEVICES

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

Classification of memories

RAM

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

1.) Static RAM.

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

2.) Dynamic RAM.

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

ROM. Read-only memory.

Information cannot be easily stored. The idea is that bits are initially defined and are never changed thereafter. As an example, it is generally prudent for the instructions used to initialize a computer upon initial power-up to be stored in ROM. The following terms refer to versions of ROM for which the stored bits can be over-written, but not easily.

PROM Programmable ROM.

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

EPROM.

ROM which can be erased using ultraviolet light.

EEPROM.

ROM which can be erased electronically.

ROM Organization

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

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

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

• An interconnection network between decoder outputs and OR gate inputs

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

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

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

A ROM consists of two parts:

• An n × 2n decoder

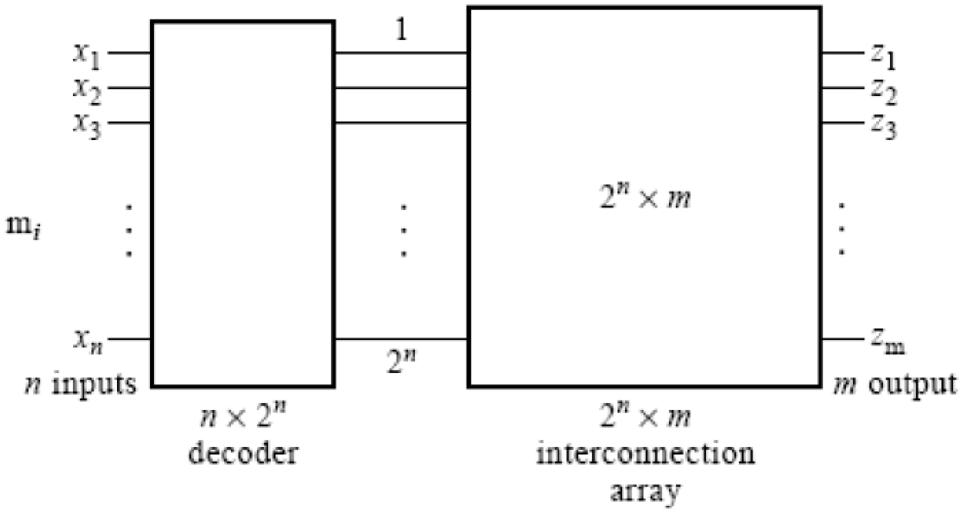
• A 2n × m array of switching devices that form interconnections between the 2n lines from
the decoder and the m output lines The 2n output lines from the decoder are called the word lines. Each of the 2n combinations that constitute the inputs to the interconnection array corresponds to a minterm and specifies an address. The memory consists of those connections that are actually made in the connection matrix between the word lines and the output lines.

Once made, the connections in the memory array are permanent.8 So this memory is not one
whose contents can be changed readily from time to time; we ―write‖ into this memory but once. However, it is possible to ―read‖ the information already stored (the connections
actually made) as often as desired, by applying input words and observing the output words. That‗s why the circuit is called read-only memory. Before you continue reading, think of two possible ways in which to fabricate a ROM so that one set of connections can be made and another set left unconnected.

Continue reading after you have thought about it.

A ROM can be almost completely fabricated except that none of the connections are made.
Such a ROM is said to be blank. Forming the connections for a particular application is called

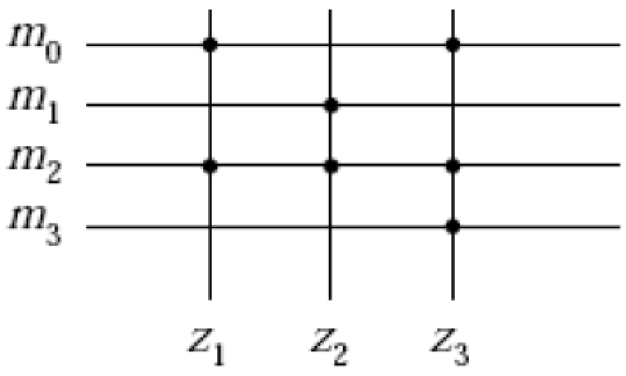
programming the ROM. In the process of programming the ROM, a mask is produced to cover those connections that are not to be made. For this reason, the blank form of the ROM is called mask programmable



Basic structure of a ROM.

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

(a)

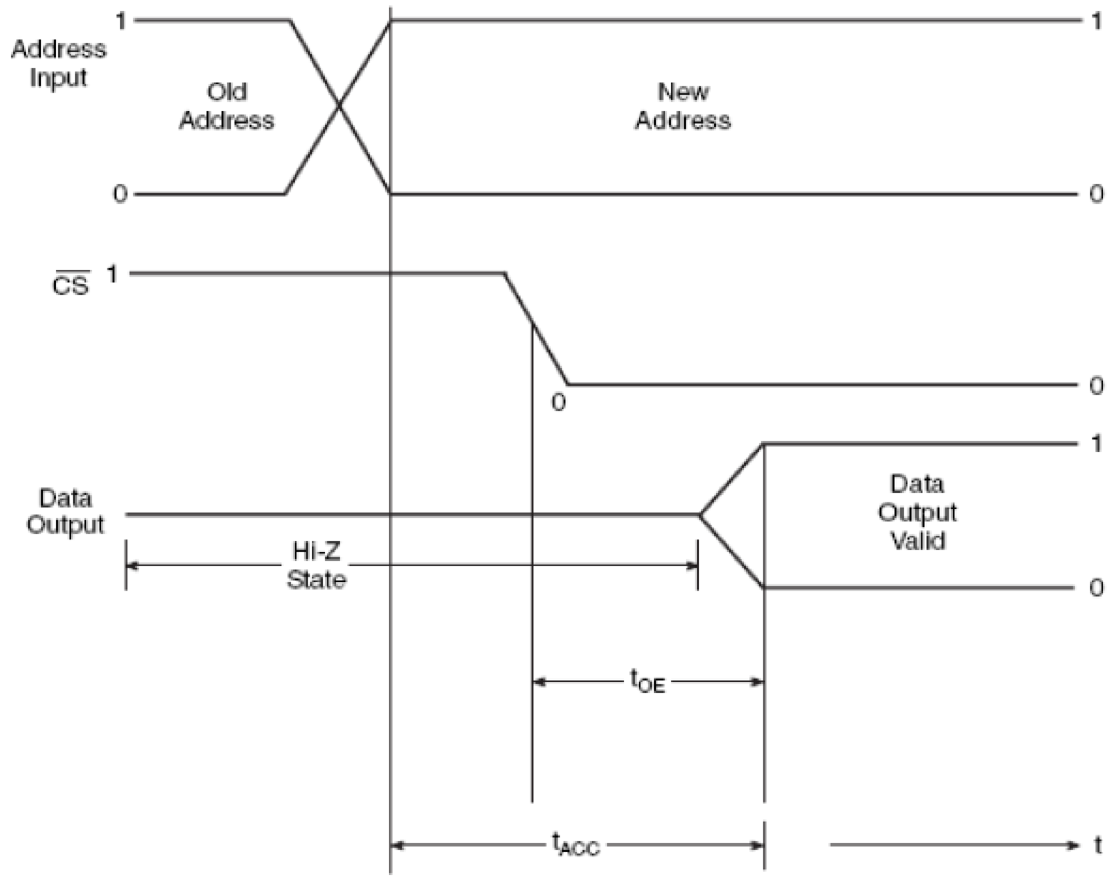


(b)

A ROM truth table and its program.

Mask-programmed ROM

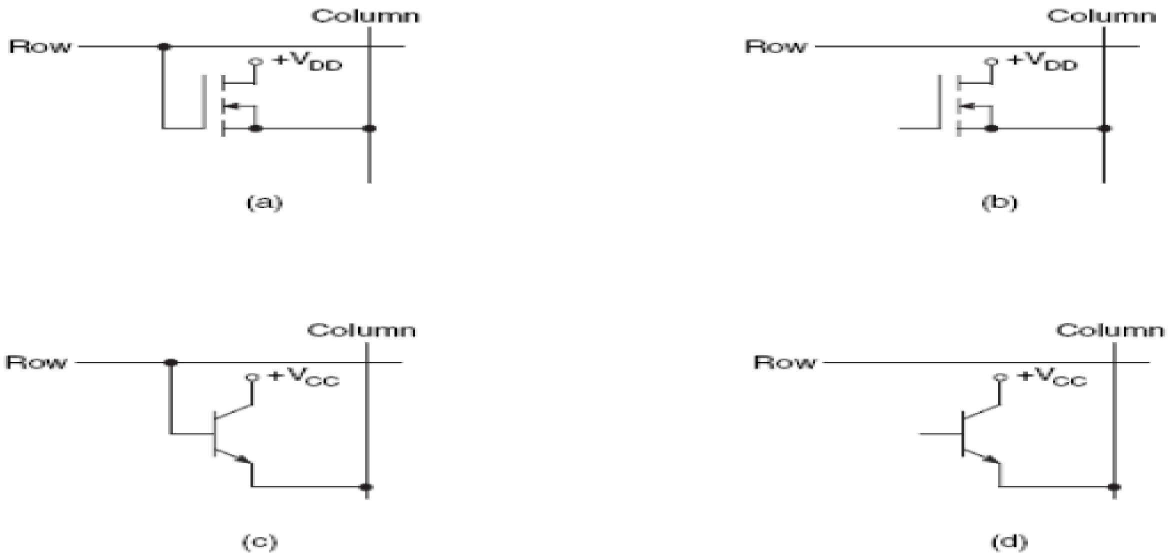
In the case of a mask-programmed ROM, the ROM is programmed at the manufacturer‘s site according to the specifications of the customer. A photographic negative, called a mask, is used to store the required data on the ROM chip. A different mask would be needed for storing each different set



Typical timing diagram a ROM read operation of information. As preparation of a mask is an expensive proposition, mask-programmed ROM is economical only when manufactured in large quantities. The limitation of such a ROM is that, once programmed, it cannot be reprogrammed.

Basic cell connection of a mask programmed ROM

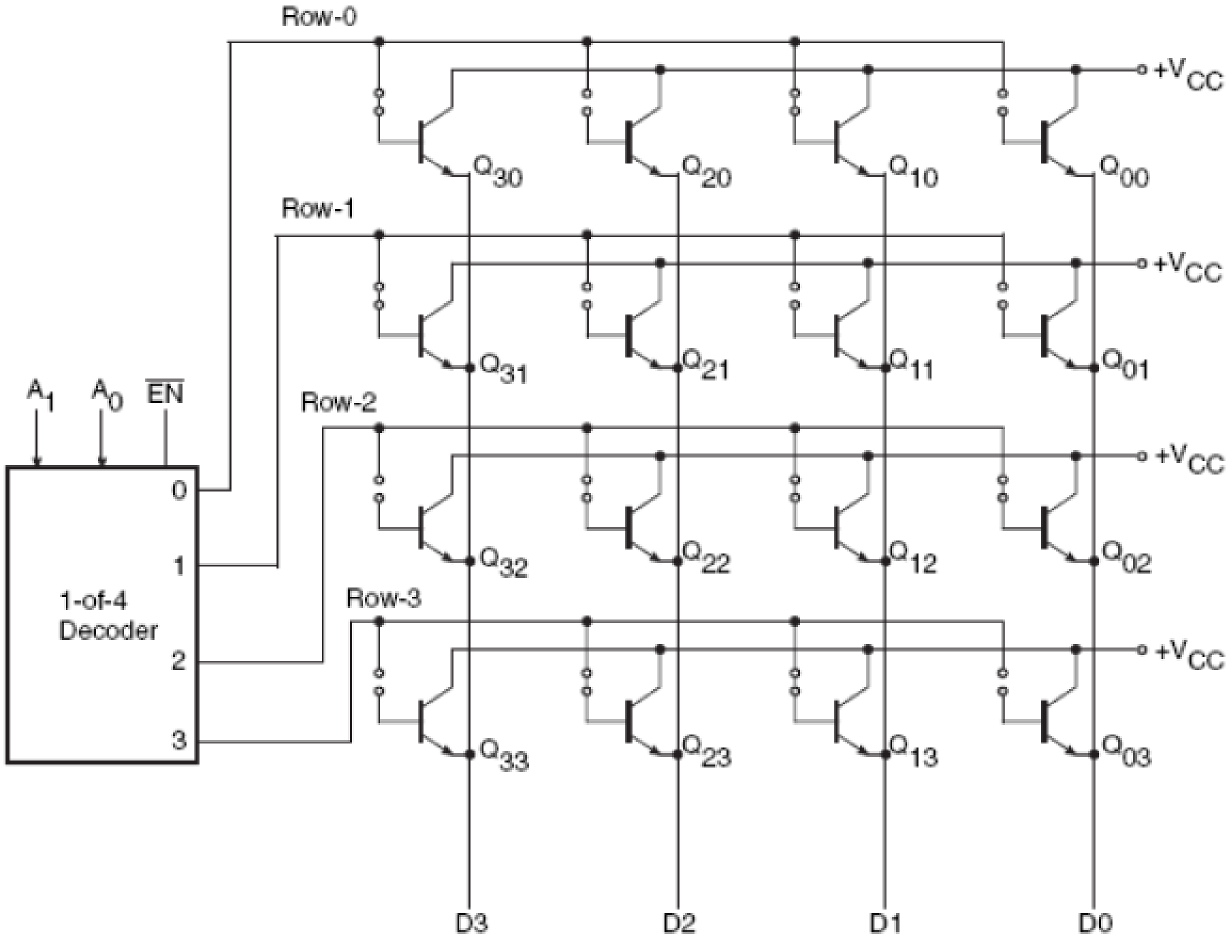
In the ROM architecture shown in Fig, the number of memory cells in a row represents the word size. The four memory cells in a row here constitute a four-bit register. There are four such



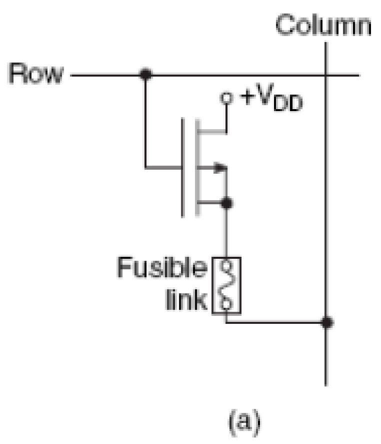
registers in this ROM. In a 16×8 ROM of this type there will be 16 rows of such transistor cells, with each row having eight memory cells. The decoder in that case would be a 1-of-16 decoder.

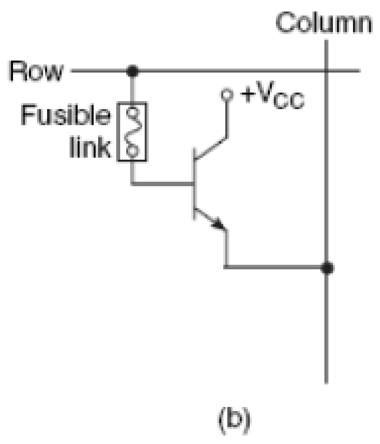
Programmable ROM

In the case of PROMs, instead of being done at the manufacturer's premises during the manufacturing process, the programming is done by the customer with the help of a special gadget called a PROM programmer. Since the data, once programmed, cannot be erased and reprogrammed, these devices are also referred to as one-time programmable ROMs. The basic memory cell of a PROM is similar to that of a mask-programmed ROM. Above show a MOSFET-based memory cell and bipolar memory cell respectively. In the case of a PROM, each of the connections that were left either intact or open in the case of a mask-programmed ROM are made with a thin fusible link, as shown in Fig.



Internal structure of a 4 x 4 bipolar mask programmed ROM





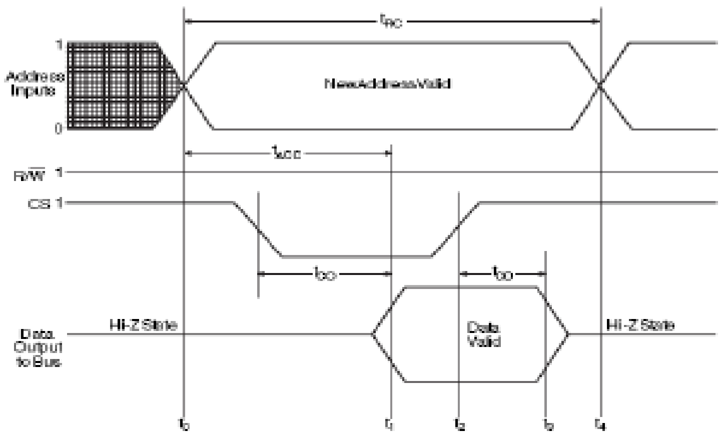
Basic Memory Cell of a PROM

Erasable PROM

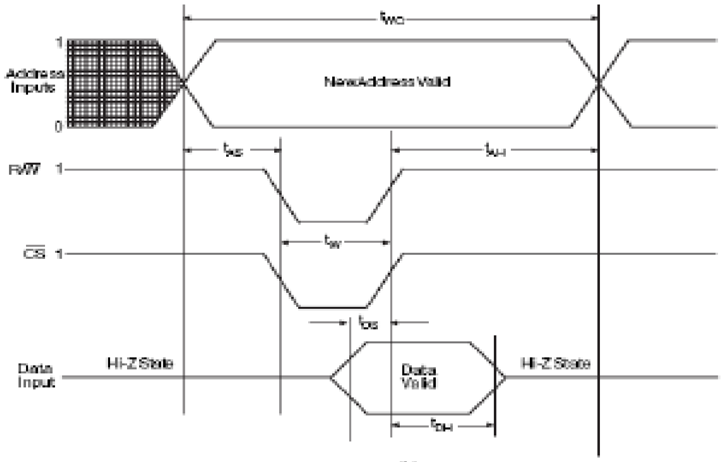
EPROM can be erased and reprogrammed as many times as desired. Once programmed, it is nonvolatile, i.e. it holds the stored data indefinitely. There are two types of EPROM, namely the ultraviolet-erasable PROM (UV EPROM) and electrically erasable PROM (EEPROM). The memory cell in a UV EPROM is a MOS transistor with a floating gate. In the normal condition, the MOS transistor is OFF. It can be turned ON by applying a programming pulse (in the range 10–25 V) that injects electrons into the floating-gate region. These electrons remain trapped in the gate region even after removal of the programming pulse. This keeps the transistor ON once it is programmed to be in that state even after the removal of power. The stored information can, however, be erased by exposing the chip to ultraviolet radiation through a transparent window on the top of the chip meant for the purpose. The photocurrent thus produced removes the stored charge in the floating-gate region and brings the transistor back to the OFF state. The erasing operation takes around 15– 20 min, and the process erases information on all cells of the chip. It is not possible to carry out any selective erasure of memory cells.

Random Access Memory

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



(a)



(b)

Memory Expansion

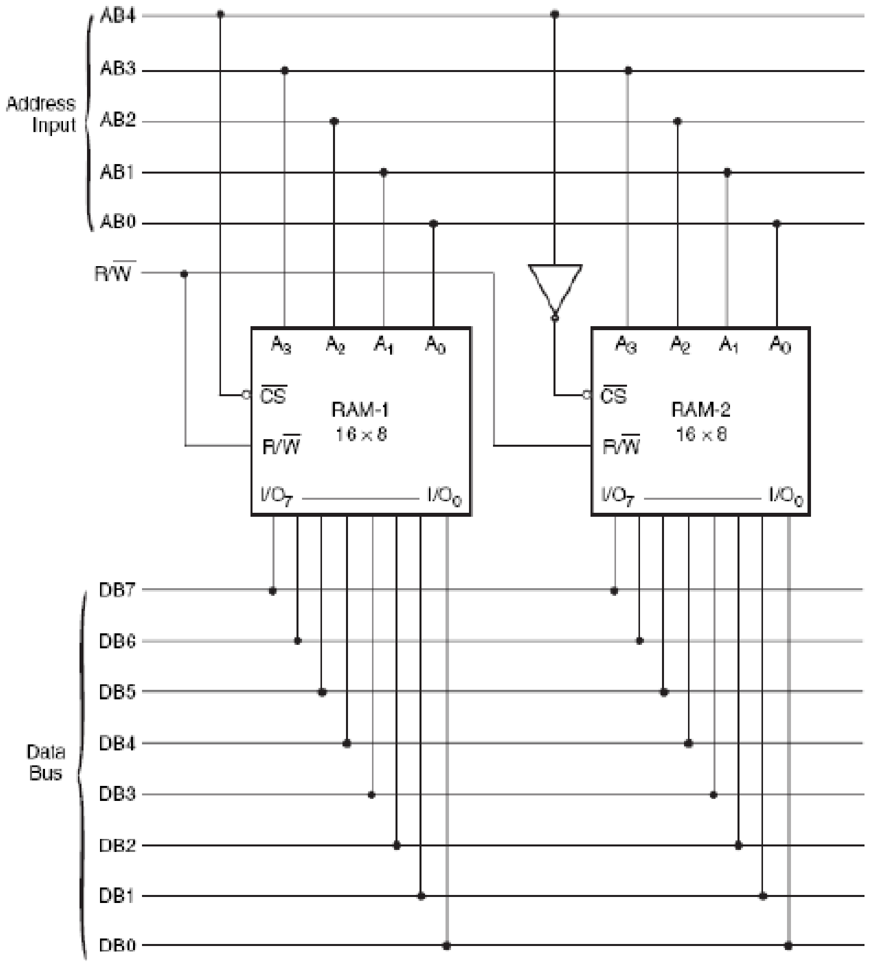
When a given application requires a RAM or ROM with a capacity that is larger than what is available on a single chip, more than one such chip can be used to achieve the objective. The required enhancement in capacity could be either in terms of increasing the word size or increasing the number of memory locations.

Word Size Expansion

Let us take up the task of expanding the word size of an available 16×4 RAM chip from four bits to eight bits. Below figure shows a diagram where two such RAM chips have been used to achieve the desired effect. The arrangement is straightforward. Both chips are selected or deselected together. Also, the input that determines whether it is a read or write operation is common to both chips. That is, both chips are selected for read or write operation together. The address inputs to the two chips are also common. The memory locations corresponding to various address inputs store four higher-order bits in the case of RAM-1 and four lower-order bits in the case of RAM-2. In essence, each of the RAM chips stores half of the word.

Memory Location Expansion

Below shows how more than one memory chip can be used to expand the number of memory locations. Let us consider the use of two 16×8 chips to get a 32×8 chip. A 32×8 chip would need five address input lines. Four of the five address inputs, other than the MSB address bit, are common to both 16×8 chips. The MSB bit feeds the input of one chip directly and the input of the other chip after inversion. The inputs to the two chips are common. Now, for first half of the memory locations corresponding to address inputs 00000 to 01111 (a total of 16 locations), the MSB bit of the address is ‗0', with the result that RAM-1 is selected and RAM-2 is deselected. For the remaining address inputs of 10000 to 11111 (again, a total of 16 locations), RAM-1 is deselected while RAM-2 is selected. Thus, the overall arrangement offers a total of 32 locations, 16 provided by RAM-1 and 16 provided by RAM-2. The overall capacity is thus 32×8.



Programmable Logic Devices

• Advantage of PLDs - can be programmed to incorporate a complex logic function within a single IC but at MSI or LSI level.

• But for larger & more complex functions – VLSI is appropriate; it can contain thousands to millions of gates within a single IC chip.

• Three ways of designing VLSI circuits:

1. Full Custom Design
2. Standard Cell Design
3. Gate Array Design

• Full Custom Design: Entire design of the chip, down to the smallest detail of the layout is performed

➢ Very expensive

➢ Suitable only for dense, fast ICs in bulk quantities

• Standard Cell Design: Large part of the design is performed ahead of time, used in previous designs. Pre-designed parts are connected to form IC design. Like hierarchical design procedure.

➢ Intermediate cost

➢ Lower density & lower performance than full custom

• Gate Array Design: Pattern of gates fabricated in Silicon that is repeated thousands of times, so that the entire chip contains identical gates. It requires that the design specify how the gates are interconnected. Many steps of fabrication process are common and independent of final logic function. These steps are economical as they can be used for a number of different designs. Additional fabrication steps are required to interconnect the gates in order to customize the gate array to the particular design.

• New approaches of VLSI yield high capacity PLDs called Complex Programmable Logic Devices (CPLDs) or Field Programmable Gate Arrays (FPGAs). These have the following properties:

1. Substantial amounts of uncommitted combinational logic
2. Pre-implemented flip-flops
3. Programmable interconnections between the combinational logic, flip-flops, and the chip input/outputs

• Aside from these properties, VLSI PLDs differ significantly from vendor to vendor. Some are following:

• Altera MAX 7000 CPLDs based on EEPROM. It has:

➢ 16 identical logic array blocks, all of whose outputs fed into the programmable interconnect array that also receives inputs from the I/O control blocks. These I/O blocks control the input and output of the circuit.

➢ Each logic block contains 16 cells, each with a flip-flop in addition to basic PLD-like combinational logic structure.

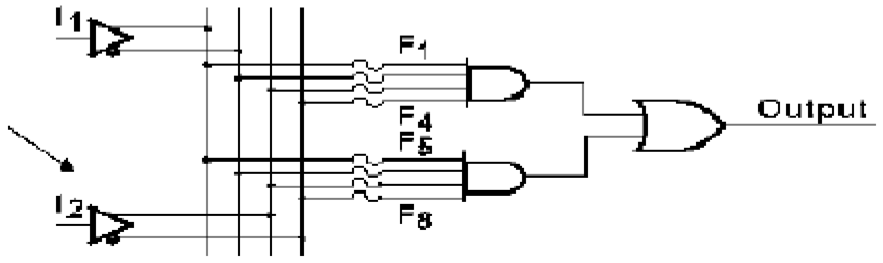
- Some of the AND gates in the cell are used for flip-flop control, such as Preset, Clear, Clock, etc.
- Flip-flop itself can be programmed to act as a D, T, JK, or SR flip-flop.

Xilinx XC4000 FPGA is implemented in an array of programmable blocks of logic called Configurable Logic Blocks (CLBs).

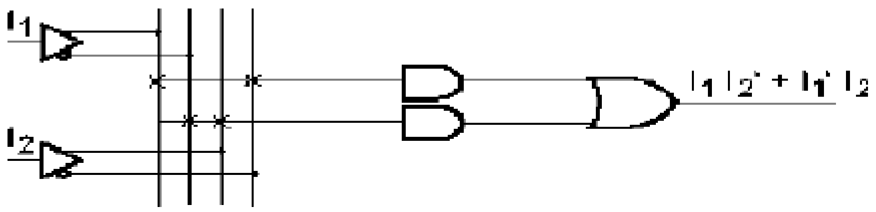
- Input to and output from the array is handled by Input/Output Blocks (IOBs) along the edges of the array.
- IOBs & CLBs are interconnected by a variety of programmable interconnection structures called switch matrices.

PROGRAMMABLE ARRAY LOGIC

The PAL device is a special case of PLA which has a programmable AND array and a fixed OR array. The basic structure of Rom is same as PLA. It is cheap compared to PLA as only the AND array is programmable. It is also easy to program a PAL compared to PLA as only AND must be programmed. The figure below shows a segment of an unprogrammed PAL. The input buffer with non-inverted and inverted outputs is used, since each PAL must drive many AND Gates inputs. When the PAL is programmed, the fusible links (F1, F2, F3…F8) are selectively blown to leave the desired connections to the AND Gate inputs. Connections to the AND Gate inputs in a PAL are represented by Xs, as shown here:



(a) Unprogrammed



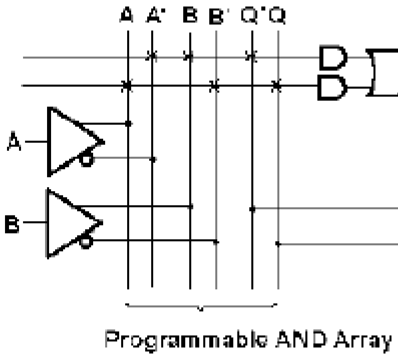
(b) Programmed

segment of an unprogrammed and programmed PAL.As an example, we will use the PAL segment of figure to realize the functionI1I2'+I1I2. theXs indicate that the I1 and I2' lines are connected to the first AND Gate, and the I1' and I2 lines are connected to the other Gate.

Typical combinational PAL have 10 to 20 inputs and from 2 to 10 outputs with 2to 8 AND gates driving each OR gate. PALs are also available which contain D flip-flops with inputs driven from the programming array logic. Such PAL provides a convenient way of realizing sequential networks. Figure below shows a segment of a sequential PAL. The D flip-flop is driven from the OR gate, which is fed by two AND gates. The flip-flop output is fed back to the programmable AND array through a buffer. Thus the AND gate inputs can be connected to A, A‘, B, B‘, Q, or Q‘. The Xs on the diagram show the realization of the next state equation.

$$Q+ = D = A'BQ' + AB'Q$$

The flip-flop output is connected to an inverting tristate buffer, which is enabled when EN = 1



Segment of a Sequential PAL

Figure below shows a logic diagram for a typical sequential PAL, the 16R4.This PAL has an AND gate array with 16 input variables, and it has 4 D flip-flops. Each flip-flop output goes through a tri state-inverting buffer (output pins 14-17). One input (pin 11) is used to enable these buffers. The rising edge of a common clock (pin 1) causes the flip-flops to change the state. Each D flip-flop input is driven from an OR gate, and each OR gate is fed from 8 AND gates. The AND gate inputs can come from the external AL inputs (pins2-9) or from the flip-flop outputs, which are fed back internally. In addition there are four input/output (i/o) terminals (pins 12,13,18 and 19), which can be used as either network outputs or as inputs to the AND gates. Thus each AND gate can have a maximum of 16 inputs (8 external inputs, 4 inputs fed back from the flip-flop outputs, and 4 inputs from the i/o terminals). When used as an output, each I/O terminal is driven from an inverting tri state buffer. Each of these buffers is fed from an OR gate and each OR gate is fed from 7 AND gates. An eighth AND gate is used to enable the

What is Programmable Logic?

In the world of digital electronic systems, there are three basic kinds of devices: memory, microprocessors, and logic. Memory devices store random information such as the contents of a spreadsheet or database. Microprocessors execute software instructions to perform a wide variety of tasks such as running a word processing program or video game. Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform. Fixed Logic Versus Programmable Logic Logic devices can be classified into two broad categories - fixed and programmable. As the name suggests, the circuits in a fixed logic device are permanent, they perform one function or set of functions - once manufactured, they cannot be changed. On the other hand, programmable logic devices (PLDs) are standard, off-the-shelf parts that offer customers a wide range of logic capacity, features, speed, and voltage characteristics - and these devices can be changed at any time to perform any number of functions. With fixed logic devices, the time required to go from design, to prototypes, to a final manufacturing run can take from several months to more than a year, depending on the complexity of the device. And, if the device does not work properly, or if the requirements change, a new design must be developed. The up-front work of designing and verifying fixed logic devices involves substantial "non-recurring engineering" costs, or NRE. NRE represents all the costs customers incur before the final fixed logic device emerges from a silicon foundry, including engineering resources, expensive software design tools, expensive photolithography mask sets for manufacturing the various metal layers of the chip, and the cost of initial prototype devices. These NRE costs can run from a few hundred thousand to several million dollars. With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. The PLD that is used for this prototyping is the exact same PLD that will be used in the final production of a piece of end equipment, such as a network router, a DSL modem, a DVD player, or an automotive navigation system. There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device. Another key benefit of using PLDs is that during the design phase customers can change the circuitry as often as they want until the design operates to their satisfaction. That's because PLDs are based on re-writable memory technology - to change the design, the device is simply reprogrammed. Once the design is final, customers can go into immediate production by simply programming as many PLDs as they need with the final software design file.

CPLDs and FPGAs

The two major types of programmable logic devices are field programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs). Of the two, FPGAs offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA now shipping, part of the Xilinx Virtex™ line of devices, provides eight million "system gates" (the relative density of logic). These advanced devices also offer features such as built-in hardwired processors (such as the IBM Power PC), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing.
CPLDs, by contrast, offer much smaller amounts of logic - up to about 10,000 gates. But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control

applications. CPLDs such as the Xilinx Cool Runner™ series also require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery operated, portable applications such as mobile phones and digital handheld assistants

The PLD Advantage

Fixed logic devices and PLDs both have their advantages. Fixed logic devices, for example, are often more appropriate for large volume applications because they can be mass-produced more economically. For certain applications where the very highest performance is required, fixed logic devices may also be the best choice.

However, programmable logic devices offer a number of important advantages over fixed logic devices, including:

• PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.

• PLDs do not require long lead times for prototypes or production parts - the PLDs are already on a distributor's shelf and ready for shipment.

• PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets - PLD suppliers incur those costs when they design their programmable devices and are able to amortize those costs over the multi-year lifespan of a given line of PLDs.

PLDs allow customers to order just the number of parts they need, when they need them, allowing them to control inventory. Customers who use fixed logic devices often end up with excess inventory which must be scrapped, or if demand for their product surges, they may be caught short of parts and face production delays.

• PLDs can be reprogrammed even after a piece of equipment is shipped to a customer.

In fact, thanks to programmable logic devices, a number of equipment manufacturers now tout the ability to add new features or upgrade products that already are in the field. To do this, they simply upload a new programming file to the PLD, via the Internet, creating new hardware logic in the system. Over the last few years programmable logic suppliers have made such phenomenal technical advances that PLDs are now seen as the logic solution of choice from many designers. One reasons for this is that PLD suppliers such as Xilinx are "fabless" companies; instead of owning chip manufacturing foundries, Xilinx out sources that job to partners like IBM Microelectronics and UMC, whose chief occupation is making chips. This strategy allows Xilinx to focus on designing new product architectures, software tools, and intellectual property cores while having access to the most advanced semiconductor process technologies. Advanced process technologies help PLDs in a number of key areas: faster performance, integration of more features, reduced power consumption, and lower cost. Today Xilinx is producing programmable logic devices on a state-of-the-art 0.13-micron low k copper process - one of the best in the industry.

Complex programmable logic devices:

A complex programmable logic device (CPLD) is a programmable logic device with complexity between that of PALs and FPGAs, and architectural features of both. The building block of a CPLD is the macro cell, which contains logic implementing disjunctive normal form expressions and more specialized logic operations.

Features in common with PALs:

▪ Non-volatile configuration memory. Unlike many FPGAs, an external configuration ROM isn't required, and the CPLD can function immediately on system start-up.

▪ For many legacy CPLD devices, routing constrains most logic blocks to have input and output signals connected to external pins, reducing opportunities for internal state storage and deeply layered logic. This is usually not a factor for larger CPLDs and newer CPLD product families.

Features in common with FPGAs:

▪ Large number of gates available. CPLDs typically have the equivalent of thousands to tens of thousands of logic gates, allowing implementation of moderately complicated data processing devices. PALs typically have a few hundred gate equivalents at most, while FPGAs typically range from tens of thousands to several million.

▪ Some provisions for logic more flexible than sum-of-product expressions, including complicated feedback paths between macro cells, and specialized logic for implementing various commonly-used functions, such as integer arithmetic.

The most noticeable difference between a large CPLD and a small FPGA is the presence of on-chip non-volatile memory in the CPLD. This distinction is rapidly becoming less relevant, as several of the latest FPGA products also offer models with embedded configuration memory.

The characteristic of non-volatility makes the CPLD the device of choice in modern digital designs to perform 'boot loader' functions before handing over control to other devices not having this capability. A good example is where a CPLD is used to load configuration data for an FPGA from non-volatile memory.

CPLDs were an evolutionary step from even smaller devices that preceded them, PLAs (first shipped by Signetics), and PALs. These in turn were preceded by standard logic products, that offered no programmability and were "programmed" by wiring several standard logic chips together.

The main distinction between FPGA and CPLD device architectures is that FPGAs are internally based on Look-up tables (LUTs) while CPLDs form the logic functions with sea of-gates (e.g. sum of products).