Unit 1

01

NUMBER SYSTEM

Unit Structure

1.0 OBJECTIVES

This chapter would make you understand the following concepts

- What is difference between analog and digital system?
- Different numbering system.
- Conversion from one number system to another
- Floating point numbers
- Weighted codes binary coded decimal
- Non-weighted codes Excess 3 code
- Gray code and Alphanumeric codes
- Error detection and correction
- Universal Product Code and Code conversion

1.1 INTRODUCTION

The study of number systems is important from the viewpoint of understanding how data are represented before they can be processed by

any digital system including a digital computer. In this chapter we will discuss different number systems commonly used to represent data such as the binary, octal and hexadecimal number systems.

1.2 ANALOG SYSTEM, DIGITAL SYSTEM

Digital as well as Analog System, both are used to transmit signals from one place to another like audio/video. Digital system uses binary format as 0 and 1 whereas analog system uses electronic pulses with varying magnitude to send data.

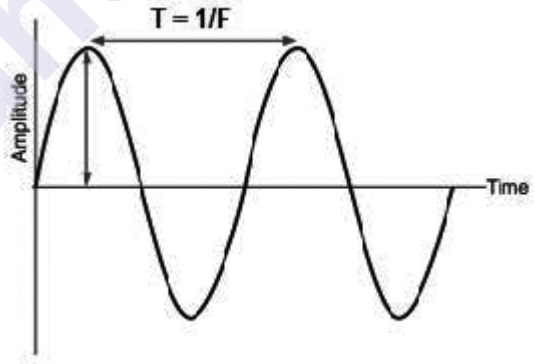
1.2.1 Analog system:

A signal is defined as any physical quantity that varies with time, space, or any other independent variable or variables. Mathematically signal can be function of one or more independent variables, for example,

S_1 (t) = 10 t
S_2 (t) = 10 t^2

Most of the signals find in science and engineering are analog in nature i.e. the signals are functions of a continuous variable, such as time or space, and usually take on values in continuous range. The most common example of analog signal is sinusoidal waveform as shown Fig. 1.2.1

The expression for the signal can be written as $S(t)=A \sin(\omega t+\theta)$

$S(t)=A \sin(\omega t+\theta)$ Where, **A** = Amplitude ω = Radian/sec θ = Phase **T** = Time duration of one cycle **F** = Frequency = **1/T**	

To measure an analog signal ***analog multimeter*** is used. The main problem with an analog signal is continuously varying with respect to time; the person has to be expert in *Time domain analysis* to find out perfect result.

1.2.2 Digital system:

To overcome the problems of an analog system; the digital system developed. Digital system requires digital information. Digital information

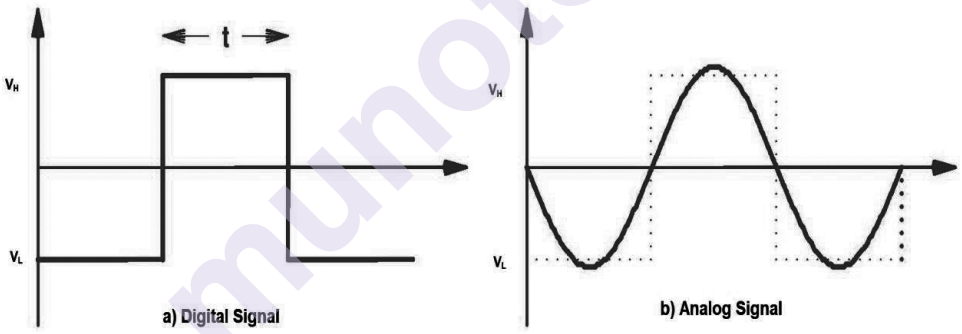
can be represented by fixed number of non – continuous or discrete symbols called as ***Digits.*** In digital system binary system is used which has only two digits ‘‘0’’ and ‘‘1’’.

Binary system: as we are using binary in digital system which restricts the digital signal to have only two distinct values.

Advantages of Binary system:

1. Most information processing systems are constructed by using switches (binary devices).
2. Binary signal are more reliable.
3. The basic decision making processes required of digital systems are binary.

Bits: As binary quantities are encountered in many different physical forms, it is convenient to have a common way of representing binary states by using digit symbols ‘‘0’’ and ‘‘1’’ to represent two possible values of binary quantity at any time. These symbols are called as ‘‘bits’’, abbreviation of term ‘‘**Bi**nary digits’’. Fig. 1.2.2 shows digital and analog signal. We use ‘‘1’’ to denote ‘‘IGH’’ and ‘‘0’’ to denote ‘‘LOW’’ level of the signal. Binary voltage values $\mathbf{V_H}$ and $\mathbf{V_L}$ are represented as ‘‘1’’ and ‘‘0’’ respectively.



Convention:

In binary system two states 1 and 0 are present. The voltage levels are predefined by the manufactures of chip and user cannot change it. Generally **HIGH LOGIC = 1 = V_{CC} = +5 volt** and **LOW LOGIC = 0 = GND = 0 volt** Currently we are in the digital world. The widest application of digital system is computers and to learn inside of the computers digital system is the base also it is easy to implement and in around 90% cases you will find that analog systems are replaced by digital systems.

1.3 NUMBERING SYSTEM

We will begin our discussion on various number systems by briefly describing the parameters that are common to all number systems.

Positional Number:

A number system is defined by digits or numerals. We can combine digits as per our requirement to represent full range. The number system with which we are normally familiar is Arabic Numerals, consists of 10 digits such as 0, 1, 2,…, 9.

Decimal Number System:

The 10 Arabic digits can be combined in various ways to represent any number. Fundamental way of constructing a number is to form a sequence or string of digits in which consecutive digits represent consecutive power of 10. For example take 3 digit number 876. The 876 represent, from left to right, hundreds (8), tens (7) and unit (6).

We can decompose the number as

$$876 = 8 \times 10^2 + 7 \times 10^1 + 6 \times 10^0 \quad \ldots (1)$$

This system is decimal number system which is a good example of **positional number** system. Here each digit of multi-digit number has fixed value (or weight) determined by its position. The number is also called as **weighted number system**. Presently in the above example we have considered only integer part. One may require to represent fractional part also. Here fractional part is denoted by sequences of digits whose weights are negative powers of 10. The integer part and fractional part represents the full number for example 1.414, both integer and fractional part are separated by special symbol **'.'** called a decimal point.

The number can be decomposed as

$$1.414 = 1 \times 10^0 + 4 \times 10^{-1} + 1 \times 10^{-2} + 4 \times 10^{-3} \quad \ldots (2)$$

Number Base:

The decimal number notation can be written in generalized form where quantity **10** is replaced by **'r'** called as **base** or **radix,** of the number system.

We will represent number $\mathbf{x_2\ x_1\ x_0\ .\ x^{-1}\ x^{-2}}$ as

$$x_2\ x_1\ x_0\ .\ x_{-1}\ x_{-2} = x_2 \text{ X } r^2 + x_1 \text{ X } r^1 + x_0 \text{ X } r^0 .\ x_{-1} \text{ X } r^{-1} + x_{-2} \text{ X } r^{-2} \ \ldots (3)$$

Following table shows various number systems of our interest.

System Name	Base 'r'	Digits / symbols used in the system
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.3.1 Binary number system:

We have already seen that binary number system has base / radix 2, which means it has only two digits, namely ‘‘0’’ and ‘‘1’’. The weights for binary number can be given by

Binary number	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	…
Equivalent decimal	16	8	4	2	1	0.5	0.25	0.125	…

By putting r = 2 in equation (3) we can get equation for binary as follows

$x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0 + x_{-1} \times 2^{-1} + x_{-2} \times 2^{-2}$ … (4)

Let’s have one example, If Binary number is $(101)_2$, then $x_2\ x_1\ x_0 = 101$

$x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$

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

$= 4 + 0 + 1 = (5)_{10}$

$\therefore\ (101)_2 = (5)_{10}$

1.3.2 Octal number system:

The octal number system has a base 8 and consists of 8 different digits or symbols such as 0 to 7. As there are 8 digits, 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary format. Following table shows 3 bit binary equivalent for each octal number

3-bit binary equivalent		Octal Number	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

The weights for actual number system can be given by

Octal number	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}	----
Equivalent decimal	512	64	8	1	1/8 = 0.125	1/64 = 0.01562	1/512 = 0.00195	----

By putting r = 8 in equation (4) we can get equation for octal as follows

$x_2 \times 8^2 + x_1 \times 8^1 + x_0 \times 8^0 + x_{-1} \times 8^{-1} + x_{-2} \times 8^{-2}$ … (5)

Let's have one example, If Octal number is $(357)_8$, then $x_2\ x_1\ x_0 = 357$

$x_2 \times 8^2 + x_1 \times 8^1 + x_0 \times 8^0$

$= 3 \times 8^2 + 5 \times 8^1 + 7 \times 8^0$

$= 192 + 40 + 7 = (239)_{10}$

$\therefore\ (357)_8 = (239)_{10}$

1.3.3 Hexadecimal number system:

The Hexadecimal number system has a base 16 and consists of 16 different digits or symbols. First ten digits or symbols are from decimal number system i.e. 0, 1, 2, …, 9 and next six are A, B, C, D, E and F representing 10, 11, 12, 13, 14 and 15 respectively. As there are 16 digits or symbols, 4 bits ($2^4 = 16$) are sufficient to represent any hexadecimal number in binary format. Following table shows 4 bit binary equivalent for each hexadecimal number.

4-bit binary equivalent		**Hexadecimal Number**		
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10 = A
1	0	1	1	11 = B
1	1	0	0	12 = C
1	1	0	1	13 = D
1	1	1	0	14 = E
1	1	1	1	15 = F

The weights for actual number system can be given by:

Hexadecimal number	16^2	16^1	16^0	16^{-1}	16^{-2}	…
Equivalent decimal	256	16	1	1/16= 0.0625	1/256 = 0.00390	…

By putting r = 16 in equation (5) we can get equation for hexadecimal as follows

$x_2 \times 16^2 + x_1 \times 16^1 + x_0 \times 16^0 + x_{-1} \times 16^{-1} + x_{-2} \times 16^{-2}$ …… (6)

Let's have one example, If Hexadecimal number is **$(1AF)_{16}$,** then

$x_2\ x_1\ x_0 = 1\ A\ F$

$x_2 \times 16^2 + x_1 \times 16^1 + x_0 \times 16^0$

$= 1 \times 16^2 + A \times 16^1 + F \times 16^0$

$= 256 + (10 \times 16) + (15 \times 1)$

$= 256 + 160 + 15 = (431)_{10}$

$\therefore\ (1AF)_{16} = (431)_{10}$

1.4 CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER

In this section we are going to study conversion from one type of number system to another i.e. decimal to binary / octal / hexadecimal or vice versa and hexadecimal to octal, hexadecimal to binary, binary to octal etc.

There are many methods or techniques which can be used to convert numbers from one base to another. In this chapter, we'll demonstrate the following

Decimal to Other Base System

- Other Base System to Decimal
- Other Base System to Non-Decimal
- Binary to Octal
- Octal to Binary
- Binary to Hexadecimal
- Hexadecimal to Binary

1.4.1 Decimal to Other Base System:

Step 1 : Divide the decimal number to be converted by the value of the new base.

Step 2 : Get the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.

Step 3 : Divide the quotient of the previous divide by the new base.

Step 4 : Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Example:
Decimal Number : 29_{10}

Calculating Binary Equivalent:

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

Decimal Number : 29_{10} = Binary Number : 11101_2

1.4.2 Other Base System to Decimal System:

Step 1 : Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 : Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3 : Sum the products calculated in Step 2. The total is the Equivalent value in decimal.

Example:
Binary Number: 11101_2

Calculating Decimal Equivalent:

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number : 11101_2 = Decimal Number : 291_0

1.4.3 Binary to Octal:

Step 1 : Divide the binary digits into groups of three (starting from the right).

Step 2 : Convert each group of three binary digits to one octal digit.

Example :

Binary Number : 10101_2

Calculating Octal Equivalent:

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	$2_8\ 5_8$
Step 3	10101_2	25_8

Binary Number : 10101_2 = Octal Number : 25_8

1.4.4 Octal to Binary:

Step 1 : Convert each octal digit to a 3-digit binary number (the octal digits may be treated as decimal for this conversion).

Step 2 : Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example:

Octal Number : 25_8

Calculating Binary Equivalent:

Step	Octal Number	Binary Number
Step 1	25_8	$2_{10}\ 5_{10}$
Step 2	25_8	$010_2\ 101_2$
Step 3	25_8	010101_2

Octal Number : 25_8 = Binary Number : 10101_2

1.4.5 Hexadecimal to Binary:

Step 1 : Convert each hexadecimal digit to a 4-digit binary number (the hexadecimal digits may be treated as decimal for this conversion).

Step 2 : Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example:

Hexadecimal Number : 15_{16}

Calculating Binary Equivalent

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	$1_{10}\ 5_{10}$
Step 2	15_{16}	$0001_2\ 0101_2$
Step 3	15_{16}	00010101_2

1.5 FLOATING POINT NUMBER

Floating point numbers are used to represent non integer fractional numbers and are used in technical calculations. E.g. 3.256, 2.1 and 0.0036.

The most commonly used floating point standard is the IEEE standard.

According to this standard, floating point numbers are represented with 32 bits (single precision) or 64 bits (double precision).

It is an arithmetic operations consist of addition, subtraction, multiplication and division.

These operations are done with algorithms similar to those used on sign magnitude integers (because of the similarity of representation) example, only add numbers of the same sign. If the numbers are of opposite sign, must do subtraction.

Addition:

Example on decimal value given in scientific notation:

```
    3.25 x 10 ** 3
+   2.63 x 10 ** -1
    --------------------
```

First step: align decimal points

Second step: add

```
    3.25          x   10 ** 3
+   0.000263      x   10 ** 3
-----------------------------------
  3.250263        x   10 ** 3
```

(presumes use of infinite precision, without regard for accuracy)

Third step : normalize the result (already normalized!)

Example on fl pt. value given in binary:

		S	E	F
.25	=	0	01111101	00000000000000000000000
100	=	0	10000101	10010000000000000000000

to add these fl. pt. representations,

Step 1 : align radix points

shifting the mantissa LEFT by 1 bit DECREASES THE EXPONENT by 1
shifting the mantissa RIGHT by 1 bit INCREASES THE EXPONENT by 1

we want to shift the mantissa right, because the bits that fall off the end should come from the least significant end of the mantissa

-> choose to shift the .25, since we want to increase it's exponent.

-> shift by 10000101

01111101

00001000 (8) places.

with hidden bit and radix point shown, for clarity

0 01111101 00000000000000000000000 (original value)

0 01111110 10000000000000000000000 (shifted 1 place)

(note that hidden bit is shifted into msb of mantissa)

0 01111111 01000000000000000000000 (shifted 2 places)

0 10000000 00100000000000000000000 (shifted 3 places)

0 10000001 00010000000000000000000 (shifted 4 places)

0 10000010 00001000000000000000000 (shifted 5 places)

0 10000011 00000100000000000000000 (shifted 6 places)

0 10000100 00000010000000000000000 (shifted 7 places)

0 10000101 00000001000000000000000 (shifted 8 places)

Step 2 : add (don't forget the hidden bit for the 100)

```
      0 10000101 1.10010000000000000000000      (100)
+     0 10000101 0.00000001000000000000000      (.25)
      -----------------------------------------------
      0 10000101 1.10010001000000000000000
```

Step 3 : normalize the result (get the "hidden bit" to be a 1)

it already is for this example.

result is

0 10000101 10010001000000000000000

suppose that the result of an addition of aligned mantissas gives

10.11110000000000000000000

and the exponent to go with this is 10000000.

We must put the mantissa back in the normalized form. Shift the mantissa to the right by one place, and increase the exponent by 1.

The exponent and mantissa become

10000001 1.01111000000000000000000 0 (1 bit is lost off the least significant end)

Subtraction:

Like addition as far as alignment of radix points then the algorithm for subtraction of sign mag. numbers takes over.

Before subtracting,

- Compare magnitudes (don't forget the hidden bit!)
- Change sign bit if order of operands is changed.

Don't forget to normalize number afterward.

Example :

0 10000001 10010001000000000000000 (the representations)

\- 0 10000000 11100000000000000000000

Step 1 : align radix points

0 10000000 11100000000000000000000

becomes

0 10000001 11110000000000000000000 (notice hidden bit shifted in)

0 10000001 1.10010001000000000000000

\- 0 10000001 0.11110000000000000000000

Step 2 : subtract mantissa

1.10010001000000000000000

\- 0.11110000000000000000000

0.10100001000000000000000

Step 3 : put result in normalized form

Shift mantissa left by 1 place, implying a subtraction of 1 from the exponent.

0 10000000 01000010000000000000000

Multiplication:

example on decimal values given in scientific notation:

```
   3.0  x 10  ** 1
+  0.5 x 10   ** 2
----------------------
```

algorithm: multiply mantissas add exponents

```
   3.0 x 10   **  1
+  0.5 x 10   **  2
-------------------------
   1.50 x 10 **  3
```

example in binary: use a mantissa that is only 4 bits so that I don't spend all day just doing the multiplication part.

0 10000100 0100
x 1 00111100 1100
---

mantissa multiplication: 1.0100
(don't forget hidden bit) x 1.1100

00000
00000
10100
10100
10100

1000110000
becomes 10.00110000

add exponents: always add true exponents (otherwise the bias gets added in twice)

biased:

10000100
+ 00111100

10000100 01111111 (switch the order of the subtraction,
- 01111111 - 00111100 so that we can get a negative value)

00000101 01000011

true exp true exp
is 5. is -67

add true exponents 5 + (-67) is -62.

re-bias exponent: -62 + 127 is 65.

unsigned representation for 65 is 01000001.

put the result back together (and add sign bit).

1 01000001 10.00110000

normalize the result:

(moving the radix point one place to the left increases the exponent by 1.)

1 01000001 10.00110000

becomes

1 01000010 1.000110000

this is the value stored (not the hidden bit!):

1 01000010 000110000

Division:

It is similar to multiplication.

true division:

- do unsigned division on the mantissas (don't forget the hidden bit)
- subtract TRUE exponents

The IEEE standard is very specific about how all this is done.
Unfortunately, the hardware to do all this is pretty slow.

Some comparisons of approximate times:

- 2's complement integer add 1 time unit
- fl. pt add 4 time units
- fl. pt multiply 6 time units
- fl. pt. Divide 13 time units

There is a faster way to do division. It is called division by reciprocal approximation. It takes about the same time as a fl. pt. multiply. Unfortunately, the results are not always the same as with true division.

Division by reciprocal approximation:

instead of doing a / b
they do a x 1/b.
figure out a reciprocal for b, and then use the fl. pt.
multiplication hardware.

example of a result that isn't the same as with true division.

true division: 3/3 = 1 (exactly)
reciprocal approx: 1/3 = .33333333
3 x .33333333 = .99999999, not 1
It is not always possible to get a perfectly accurate reciprocal

1.6 WEIGHTED CODES BINARY CODED DECIMAL, NON-WEIGHTED CODES EXCESS 3 CODE

Binary code: The digital data is represented, stored and transmitted as group of binary bits. This group is also called as binary code.

- The binary code is represented by the number as well as alphanumeric letter.
- The group of symbols is called as a code.

Binary Codes for Decimal digits:

The following table shows the various binary codes for decimal digits 0 to 9.

Decimal Digit	8421 Code	2421 Code	84-2-1 Code	Excess 3 Code
0	0000	0000	0000	0011
1	0001	0001	0111	0100
2	0010	0010	0110	0101
3	0011	0011	0101	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1100	1010	1001
7	0111	1101	1001	1010
8	1000	1110	1000	1011
9	1001	1111	1111	1100

We have 10 digits in decimal number system. To represent these 10 digits in binary, we require minimum of 4 bits. But, with 4 bits there will be 16 unique combinations of zeros and ones. Since, we have only 10 decimal digits, the other 6 combinations of zeros and ones are not required.

8 4 2 1 code:

- The weights of this code are 8, 4, 2 and 1.
- This code has all positive weights. So, it is a positively weighted code.
- This code is also called as natural BCD Binary Coded decimal code

Example:

Let us find the BCD equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the BCD 84218421 codes of 7, 8 and 6 are 0111, 1000 and 0110 respectively.

$$\therefore 786_{10} = 011110000110_{BCD}$$

There are 12 bits in BCD representation, since each BCD code of decimal digit has 4 bits.

2 4 2 1 code:

- The weights of this code are 2, 4, 2 and 1.
- This code has all positive weights. So, it is a positively weighted code.
- It is an unnatural BCD code. Sum of weights of unnatural BCD codes is equal to 9.

- It is a self-complementing code. Self-complementing codes provide the 9's complement of a decimal number, just by interchanging 1's and 0's in its equivalent 2421 representation.

Example:

Let us find the 2421 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 2421 codes of 7, 8 and 6 are 1101, 1110 and 1100 respectively.

Therefore, the 2421 equivalent of the decimal number 786 is 110111101100.

8 4 -2 -1 code:

- The weights of this code are 8, 4, -2 and -1.
- This code has negative weights along with positive weights. So, it is a negatively weighted code.
- It is an unnatural BCD code.
- It is a self-complementing code.

Example:

Let us find the 8 4-2-1 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 8 4 -2 -1 codes of 7, 8 and 6 are 1001, 1000 and 1010 respectively.

Therefore, the 8 4 -2 -1 equivalent of the decimal number 786 is 100110001010.

Excess 3 code:

- This code doesn't have any weights. So, it is an un-weighted code.
- We will get the Excess 3 code of a decimal number by adding three 00110011 to the binary equivalent of that decimal number. Hence, it is called as Excess 3 code.
- It is a self-complementing code.

Example:

Let us find the Excess 3 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the Excess 3 codes of 7, 8 and 6 are 1010, 1011 and 1001 respectively.

Therefore, the Excess 3 equivalent of the decimal number 786 is 101010111001

1.7 GRAY CODE

A binary code used to represent digits generated from a mechanical sensor that may be prone to error. Used in telegraphy in the late 1800s, and also known as "reflected binary code," Gray code was patented by Bell Labs researcher Frank Gray in 1947.

Only Change One Bit:

In Gray code, there is only one bit location different between numeric increments, which make mechanical transitions from one digit to the next less error prone.

The following table shows the 4-bit Gray codes corresponding to each 4-bit binary code.

Decimal Number	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

- This code doesn't have any weights. So, it is an un-weighted code.
- In the above table, the successive Gray codes are differed in one bit position only. Hence, this code is called as unit distance code.

Binary code to Gray Code Conversion:

Follow these steps for converting a binary code into its equivalent Gray code.

- Consider the given binary code and place a zero to the left of MSB.
- Compare the successive two bits starting from zero. If the 2 bits are same, then the output is zero. Otherwise, output is one.
- Repeat the above step till the LSB of Gray code is obtained.

Example:

From the table, we know that the Gray code corresponding to binary code 1000 is 1100. Now, let us verify it by using the above procedure.

Given, binary code is 1000.

Step 1 : By placing zero to the left of MSB, the binary code will 01000.

Step 2 : By comparing successive two bits of new binary code, we will get the gray code as 1100.

1.8 ALPHANUMERIC CODES

Alphanumeric codes are sometimes called character codes due to their certain properties. Now these codes are basically binary codes. We can write alphanumeric data, including data, letters of the alphabet, numbers, mathematical symbols and punctuation marks by this code which can be easily understandable and can be processed by the computers. Input output devices such as keyboards, monitors, mouse can be interfaced using these codes. 12-bit Hollerith code is the better known and perhaps the first effective code in the days of evolving computers in early days. During this period punch cards were used as the inputting and outputting data. But nowadays these codes are termed obsolete as many other modern codes have evolved. The most common alphanumeric codes used these days are ASCII code, EBCDIC code and Unicode. Now we will discuss about them briefly.

1.8.1 ASCII code:

The full form of ASCII code is American Standard Code for Information Interchange. It is a seven bit code based on the English alphabet.

In 1967 this code was first published and since then it is being modified and updated. ASCII code has 128 characters some of which are enlisted below to get familiar with the code.

DEC	OCT	HEX	BIN	Symbol	Description
0	000	00	00000000	NUL	Null char
1	001	01	00000001	SOH	Start of Heading
2	002	02	00000010	STX	Start of Text
3	003	03	00000011	ETX	End of Text
4	004	04	00000100	EOT	End of Transmission
5	005	05	00000101	ENQ	Enquiry
6	006	06	00000110	ACK	Acknowledgment
7	007	07	00000111	BEL	Bell
8	010	08	00001000	BS	Back Space

9	011	09	00001001	HT	Horizontal Tab
10	012	0A	00001010	LF	Line Feed
11	013	0B	00001011	VT	Vertical Tab
12	014	0C	00001100	FF	Form Feed
13	015	0D	00001101	CR	Carriage Return
14	016	0E	00001110	SO	Shift Out / X-On
15	017	0F	00001111	SI	Shift In / X-O

There are many more codes which are not included here.

1.8.2 EBCDIC:

The EBCDIC stands for Extended Binary Coded Decimal Interchange Code. IBM invented this code to extend the Binary Coded Decimal which existed at that time. All the IBM computers and peripherals use this code. It is an 8 bit code and therefore can accommodate 256 characters. Below is given some characters of EBCDIC code to get familiar with it.

Char	**EBCDIC**	**HEX**	**Char**	**EBCDIC**	**HEX**	**Char**	**EBCDIC**	**HEX**
A	1100 0001	C1	P	1101 0111	D7	4	1111 0100	F4
B	1100 0010	C2	Q	1101 1000	D8	5	1111 0101	F5
C	1100 0011	C3	R	1101 1001	D9	6	1111 0110	F6
D	1100 0100	C4	S	1110 0010	E2	7	1111 0111	F7
E	1100 0101	C5	T	1110 0011	E3	8	1111 1000	F8
F	1100 0110	C6	U	1110 0100	E4	9	1111 1001	F9
G	1100 0111	C7	V	1110 0101	E5	blank	…	…
H	1100 1000	C8	W	1110 0110	E6	.	…	…
I	1100 1001	C9	X	1110 0111	E7	(	…	…
J	1101 0001	D1	Y	1110 1000	E8	+	…	…
K	1101 0010	D2	Z	1110 1001	E9	$	…	…
L	1101	D3	0	1111 0000	F0	*	…	…

	0011							
M	1101 0100	D4	1	1111 0001	F1	)	…	…
N	1101 0101	D5	2	1111 0010	F2	–	…	…
O	1101 0110	D6	3	1111 0011	F3			

1.8.3 ISCII Code:

ISCII stands for Indian Script Code for Information Interchange. IISCII was developed to support Indian languages on computer. Language supported by IISCI include Devanagari, Tamil, Bangla, Gujarati, Gurmukhi, Tamil, Telugu, etc. IISCI is mostly used by government departments and before it could catch on, a new universal encoding standard called Unicode was introduced.

1.8.4 Hollerith code:

In 1896, Herman Hollerith formed a company called the Tabulating Machine Company. This company developed a line of machines that used punched cards for tabulation. After a number of mergers, this company was formed into the IBM, Inc. We often refer to the punched-cards used in computer systems as Hollerith cards and the 12-bit code used on a punched-card is called the Hollerith code.

A Hollerith string is a sequence of l2-bit characters; they are encoded as two ASCII characters, containing 6 bits each. The first character contains punches 12,0,2,4,6,8 and the second character contains punches 11, 1, 3, 5, 7, 9. Interleaving the two characters gives the original 12 bits. To make the characters printable on ASCII terminals, bit 7 is always set to 0 and bit 6 is said to the complement of bit 5. These two bits are ignored when reading Hollerith cards.

Today, as punched cards are mostly obsolete and replaced with other storage medias so the Hollerith code is rendered obsolete.

1.8.5 Morse code:

The Morse code, invented in 1837 by Samuel F.B. Morse, was the first alphanumeric code used in telecommunication. It uses a standardized sequence of short and long elements to represent letters, numerals and special characters of a given message. The short and long elements can be formed by sounds, marks, pulses, on off keying and are commonly known as dots and dashes. For example : The letter ‘‘A’’ is formed by a dot followed by a dash. The digit 5 is formed by 5 dots in succession. The

International Morse code treats a dash equal to three dots. To see the details of Morse code table you can refer the Internet search engines.

Due to variable length of Morse code characters, the morse code could not adapt to automated circuits. In most electronic communication, the Baudot code and ASCII code are used.

Morse code has limited applications. It is used in communication using telegraph lines, radio circuits. Pilots and air traffic controllers also use them to transmit their identity and other information

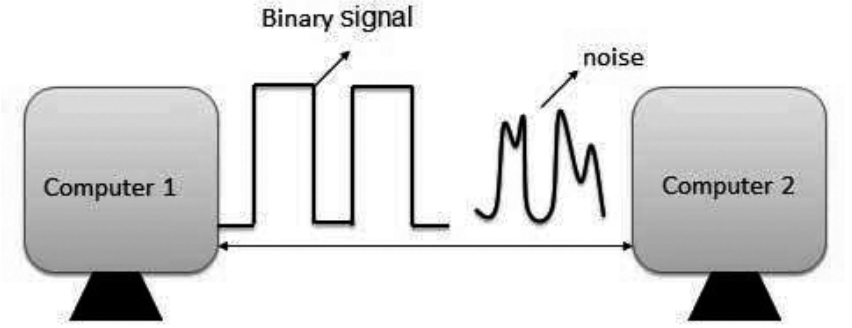
1.8.6 Teletypewriter (TTY):

A teletypewriter (TTY) is an input device that allows alphanumeric character to be typed in and sent, usually one at a time as they are typed, to a computer or a printer. The Teletype Corporation developed the teletypewriter, which was an early interface to computers. Teletype mode is the capability of a keyboard, computer, application, printer, display, or modem to handle teletypewriter input and output. Basically, this is a one-character-at-a-time mode of sending, receiving, or handling data, although it is often modified to handle a line of characters at a time. Since this mode requires little programming logic, it is often used where memory is limited. The Basic Input/Output Operating System (BIOS) sends messages to a PC display using teletype mode. Most printers offer a teletype mode. The simplest video display output format is text in teletype mode. Many modems today continue to include support for a TTY interface.

1.9 ERROR DETECTION AND CORRECTION

What is Error?:

Error is a condition when the output information does not match with the input information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.



Error-Detecting codes:

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message. A simple example of error-detecting code is parity check.

Error-Correcting codes:

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message

1.10 UNIVERSAL PRODUCT CODE

A UPC, short for universal product code, is a type of code printed on retail product packaging to aid in identifying a particular item. It consists of two parts – the machine-readable barcode, which is a series of unique black bars, and the unique 12-digit number beneath it.

The purpose of UPCs is to make it easy to identify product features, such as the brand name, item, size, and color, when an item is scanned at checkout. In fact, that's why they were created in the first place – to speed up the checkout process at grocery stores. UPCs are also helpful in tracking inventory within a store or warehouse.

To obtain a UPC for use on a product a company has to first apply to become part of the system. GS1 US, the Global Standards Organization, formerly known as the Uniform Code Council, manages the assigning of UPCs within the US.

Parts of a UPC:

After paying a fee to join, GS1 assigns a 6-digit manufacturer identification number, which becomes the first six digits in the UPC on all the company's products. That number identifies the particular manufacturer of the item.

The next five digits of the UPC is called an item number. It refers to the actual product itself. Within each company is a person responsible for issuing item numbers, to ensure that the same number isn‟t used more than once and that old numbers referring to discontinued products are phased out.

Many consumer products have several variations, based on, for example, size, flavor, or color. Each variety requires its own item number. So a box of 24 one-inch nails has a different item number than a box of 24 two-inch nails, or a box of 50 one-inch nails.

The last digit in the 12-digit UPC is called the check digit. It is the product of several calculations – adding and multiplying several digits in the code – to confirm to the checkout scanner that the UPC is valid. If the check digit code is incorrect, the UPC won't scan properly.

Advantages of UPCs:

- UPCs have a number of advantages to businesses and consumers. Because they make it possible for barcode scanners to immediately identify a product and its associated price, UPCs improve speed.
- They improve efficiency and productivity, by eliminating the need to manually enter product information.
- They also make it possible to track inventory much more accurately than hand counting, to know when more product is needed on retail shelves or in warehouses. Or when there is an issue with a particular product and consumers who purchased it need to be alerted or a recall issued, UPCs allow products to be tracked through production to distribution to retail stores and even into consumer homes.

1.11 CODE CONVERSION

In this section we are going to lean the following code conversions:

1 Binary to BCD 2 BCD to Binary

3 BCD to Excess-3 4 Excess-3 to BCD

1.11.1 Binary to BCD:

For the binary to BCD conversation the steps to be followed are as given below:

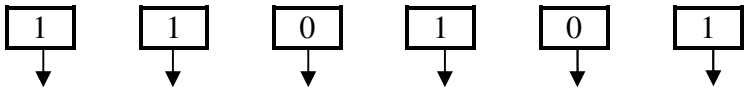
Step 1 : Convert the binary number to decimal

Step 2 : Convert decimal number into BCD

Example: Convert the binary number (110101)2 into BCD

Solution:

Step 1 : Convert the binary number to decimal



$2^5 + 2^4 + 0^3 + 2^2 + 0^1 + 2^0$

$32 + 16 + 0 + 4 + 0 + 1 = (53)_{10}$

Step 2 : Convert decimal number into BCD

(0 1 0 1) 0 0 1 1)

$\therefore (110101)_2 = (01010011)$ BCD

1.11.2 BCD to Binary:

Steps to be followed are as given below:

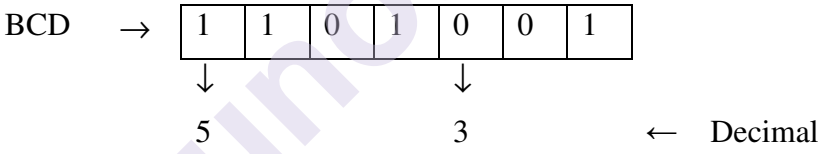
Step 1 : Convert the BCD number to decimal

Step 2 : Convert decimal number into binary

Example : Convert the BCD number (0101 0011)BCD into binary

Solution:

Step 1 : Convert the BCD number to decimal



Step 2 : Convert decimal number into binary

Use the long division method for decimal to binary conversation

$(53)_{10} = (110101)_2$

$(0101\ 0011)_{BCD} = (110\ 101)_2$

1.11.3 BCD to Excess-3:

Steps to be followed are as given below:

Step 1 : Convert BCD to decimal

Step 2 : Add $(3)^{10}$ to this decimal number

Step 2 : Convert the decimal number of step 2 into binary, to get the excess -3 code

Example : Convert (1001)BCD to excess -3

Solution:

BCD	1	0	0	1	
	↓				
Decimal		9			
Add 3	+	3			
		(12)10	Covert to Binary		$(1100)_2$

Therefore $\therefore (1001)_{BCD} = (1100)e_{x-3}$

1.11.4 Excess-3 to BCD Conversion:

Subtract $(0011)_2$ from each 4 bit excess-3 digit to obtain the corresponding BCD code.

Given XS- 3 number	1001	1010
Subtract $(0011)_2$	-0011	0011
	11	111
BCD	0110	0111

$(10011010)_{XS-3} = (0110\ 0111)_{BCD}$

UNIT END QUESTIONS

1. State the difference between analog and digital signals
2. Explain numbering system in brief
3. Explain floating point numbers with suitable example
4. Explain Universal Product Code
5. What is an error correction code?
6. Explain binary to decimal with suitable example
7. Explain decimal to binary with suitable example
8. Explain code conversion with suitable example
9. Convert the following fractional decimal numbers to equivalent binary number (show the step by step)

   1. 0.5682 **2.** 0.6954 **3.** 0.1235 **4.** 0.4754

*****

2

BINARY ARITHMETIC

Unit Structure

2.0 OBJECTIVE

This chapter would make you understand the following concepts

- What is binary arithmetic?
- Binary addition, subtraction, multiplication and division.
- Negative number representation
- Arithmetic in octal number system
- Arithmetic in hexadecimal number system
- BCD arithmetic
- Excess 3 arithmetic
- Examples on conversion

2.1 INTRODUCTION

Binary arithmetic is used in digital systems mainly because the numbers (decimal and floating-point numbers) are stored in binary format in most computer systems. All arithmetic operations such as addition, subtraction, multiplication, and division are done in binary representation

of numbers. It is necessary to understand the binary number representation to figure out binary arithmetic in digital computers.

Binary arithmetic is essential part of all the digital computers and many other digital systems.

2.2 BINARY ADDITION

- Binary addition is the key for binary multiplication, subtraction and division. The four most basic cases of binary addition are shown in Table 2.1.

Table 2.1 : Four cases of binary addition:

	A		B	Addition	Comment
Case 1	0	+	0	0	Same as decimal
Case 2	0	+	1	1	Addition
Case 3	1	+	0	1	
Case 4	1	+	1	10	• + • = • • = $(10)_2$

For cases 1, 2 and 3 of Table 2.1, the binary addition takes place by following the rules of decimal addition.

- But concentrate on case 4. Addition of binary 1 + 1 represent the combining of one pebble and one pebble to obtain a total of two pebble.

1 + 1 = • • two pebbles

- Since binary 10 stands for • • two pebbles, the result of binary addition 1 + 1 is 10.

$\therefore 1 + 1 = (10)_2$

2.1.1 Sum and Carry :

- Thus, the fourth case yields a binary two (10). When the binary numbers are added, the fourth case in Table 2.1 creates a sum of 0 in the given column and a carry of 1 over to the next column.
- The four basic rules of binary addition in terms of sum and carry are as follows :

Table 2.2 Rules for binary addition

Rule	A		B		Sum	Carry
1	**0**	**+**	**0**	**=**	**0**	**0**
2	**0**	**+**	**1**	**=**	**1**	**0**
3	**1**	**+**	**0**	**=**	**1**	**0**
4	**1**	**+**	**1**	**=**	**0**	**1**

2.3 BINARY SUBTRACTION

Rules for subtraction:

In order to understand the binary subtraction, we should remember some of the important rules of decimal subtraction. They are as follows:

1. To carry out the subtraction (A – B) where A and B are the two single digit decimal numbers. We have to consider two cases,

2. Case I : Digit A > Digit B :

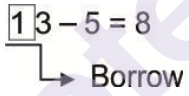
Let A = $(5)_{10}$ and B = $(3)_{10}$

Then A – B = (•••••) – (•••) = ••

$\therefore (5)_{10} - (3)_{10} = (2)_{10}$

3. Case II : Digit A < Digit B :

If A = (3)10 and B = (5)10 then we cannot perform (3 - 5) because we cannot take out 5 pebbles from 3. **Therefore, we have to borrow 1.** After borrowing, the subtraction is charged to,



2.3.1 Subtraction and Borrow:

- These two words will be used very frequently for the binary subtraction. For binary subtraction we have to remember the following four cases given in Table 2. 3

Table 2.3 : Four basic rules for binary subtraction:

Case	A		B	Subtraction	Borrow	Comment
1	0	-	0	0	0	Same as decimal
2	1	-	0	1	0	Same as decimal
3	1	-	1	0	0	Same as decimal
4	**0**	**-**	**1**	**1**	**1**	**Borrow needs to be taken**

- Consider case 4 in Table 2. 3. it is [0 - 1]. Hence a logic 1 borrowed. This will change the subtraction from [0 - 1] to [10 - 1] that means [•• - •] = • = 1.

2.4 NEGATIVE NUMBER REPRESENTATION

Binary Subtraction using 1's and 2's Complements:

- The direct binary subtraction becomes complicated as the number size increases.
- Therefore we can represent the subtraction of A - B in the form of addition as: A+(- B).
- We can represent number B (which is to be subtracted) in its 1's complement or 2's complement form and use addition instead of subtraction to get the result of A - B.

2.4.1 Subtraction using 1's Complement :

The steps to be followed for subtraction $(A)_2$ - $(B)_2$ using 1's complement are as follows :

Step 1 : Convert number to be subtracted $(B)_2$ to its 1's complement.

Step 2 : Add $(A)_2$ and 1's complement of $(B)_2$ using the rules of binary addition.

Step 3 : If final carry is 1, then add it to the result of addition obtained in step 2 to get the final result of $(A)_2$ - $(B)_2$. Note that if the final carry is 1 then the subtraction is positive and i in its true form.

Step 4 : If the final carry produced in step 2 is 0, then the result obtained in step 2 is negative and in the 1's complement form. So, convert it into the true form by complementing all the bits.

The following examples will make the concept of subtraction using 1‟s complement crystal clear.

There are four possible cases depending on the magnitude and sign of the numbers involved.

Case 1: Number A and B, both positive and A > B.

Case 2: A and B both positive and A < B.

Case 3: Both numbers are negative.

Case 4: A = B.

Let us discuss them one by one.

Case 1 : A and B, both positive and A > B

Ex. 2.1 : Perform $(9)_{10}$ - $(4)_{10}$ using 1's complement method

Soln.:

Step1 : Convert $(4)_{10}$ into 1's complement:

$(4)_{10} = (0\ 1\ 0\ 0)_2$

and 1's complement of $(0100)_2 = (1\ 0\ 1\ 1)_2$

Step2 : Add $(9)_{10}$ and 1's complement of $(4)_{10}$:

		1	0	0	1
	+	1	0	1	1
1		0	1	0	0 Result

Step 3 : Add the final carry to the result obtained in step 2:

$(9)_{10}$		1	0	0	1	
1's complement of $(4)_{10}$:	+	1	0	1	1	
Final carry is generated →	[1]	0	1	0	0	Result
					1	
		0	1	0	1	Answer is positive and in true form

Thus the answer is $(0101)_2$.

Note: When the final carry is produced the answer is positive and in its true form.

$(9)_{10} - (4)_{10} = (5)_{10} = (0101)_2$ which we have obtained.

Case 2 : A and B both positive with A < B

Ex.2.2 : Subtract $(9)_{10}$ from $(4)_{10}$ using 1' s complement method

Soln. :

Given : $A = (4)_{10} = (0100)_2$ $B = (9)_{10} = (1001)_2$

Step 1 : Obtain 1's complement of $(9)_{10}$:

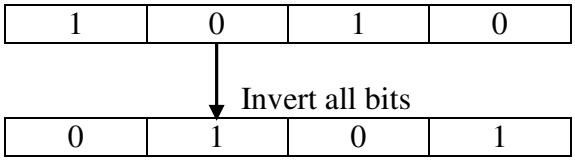
1's complement of $(9)_{10}$ or $(1001)_2$ is $(0110)_2$.

Step 2 : Add $(4)_{10}$ and 1's complement of $(9)_{10}$. :

$(4)_{10}$		0	1	0	0
1's complement of $(9)_{10}$	+	0	1	1	0
Final carry →	[0]	1	0	1	0

Note: As the final carry is 0, the answer is negative and its 1's complement form.

So convert the answer into its true form, as follows:



Answer

But $(0\ 1\ 0\ 1)_2 = (5)_{10}$

$\therefore (4)_{10} - (9)_{10} = (-5)_{10}$

Case 3 : Both number negative:

Ex. 2.3 : Perform $(-4)_{10} - (-8)_{10}$ using 1's complement method.

Soln. :

$$(-4)_{10} - (-8)_{10} = (-4)_{10} + (8)_{10}$$

Here number A i.e. $(-4)_{10}$ is negative and B is positive.

So we have to take the 1's complement of A.

Step 1 : Convert number A to 1's complements:

$$(4)_{10} = (0100)_2$$

1's complement of $(0100)_2 = (1011)_2$

Step2 : Add 1's complement of A to number B:

1's complement of	$(4)_{10}$	=		1	0	1	1	
	$(8)_{10}$	=	+	1	0	0	0	
Final carry	→		1	0	0	0	1	Result
								→**1**

Step2 : Add the final carry 1

Answer in the true form = $0\ 1\ 0\ 0 = (4)_{10}$

Thus, the answer is positive and in the true form

$\therefore (-4)_{10} - (-8)_{10} = (4)_{10}$

Case 4: **Equal and opposite numbers:**

The last possibility in subtraction of numbers is to subtract a number from itself. Refer previous example to understand the result of this subtraction**.**

Ex. 2.4 Subtract $(5)_{10}$ from $(5)_{10}$ using 1's complement.

Soln. : We are supposed to perform $(5)_{10} - (5)_{10}$.

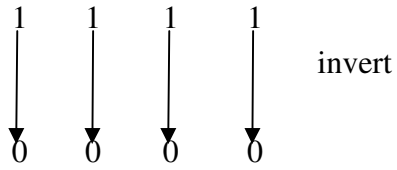
Step 1 : Obtain1's complement of $(5)_{10}$:

1's complement of $(5)_{10}$ or $(0\ 1\ 01)_2$ is $(1010)_2$.

Step 2 : Add $(5)_{10}$ and 1's complement of $(5)_{10}$. :

$(5)_{10}$		0	1	0	1	
1's complement of $(5)_{10}$	+	1	0	1	0	
	0	1	1	1	1	The answer is in 1's complement form

Step 3 : Convert the answer to its true form. :



Answer in its true form

$\therefore (5)_{10} - (5)_{10} = (0)_{10}$

2.4.2 Binary Subtraction using 2's Complement Method:

If the subtraction of two binary numbers A and B is to be performed using the 2's complement, then the following steps are to be followed.

Steps to be followed:

Step 1 : Add $(A)_2$ to the 2's complement of $(B)_2$

Step 2 : If the carry is generated then the result is positive and in its true form.

Step3 : If the carry is not produced, then the result is negative and in its 2's complements form.

Note: Carry is always to be discarded in the subtraction using 2's complement.

Depending on the magnitude and polarity of numbers A and B we will deal with different possibilities as follows:

Case 1 : Both A and B are positive with A > B.

Case 2 : Both A and B are positive with A <B.

Case 3 : Both the numbers are negative.

Case 4 : Both numbers are equal.

Case 1: Both numbers positive with A >B

The subtraction (A - B) for A > B is illustrated in Ex. 2.5

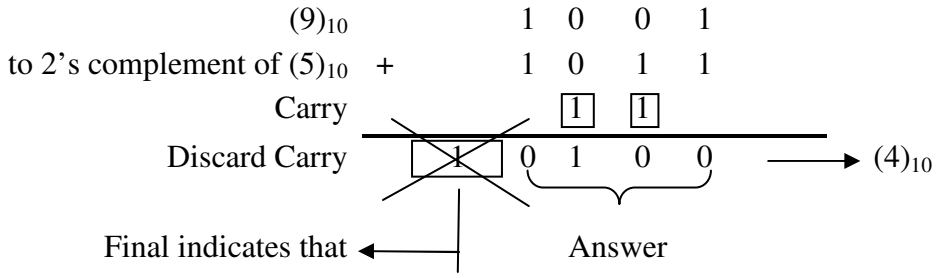
Ex.2.5: Perform $(9)_{10} - (5)_{10}$ using 2's complement method.

Soln. :

Step1 : Obtain 2's complement of $(5)_{10}$:

Decimal	**Binary**	**2's complement**
$(5)_{10}$	$(0101)_2$	1011

Step 2 : Add $(9)_{10}$ to 2's complement of $(5)_{10}$:



The answer is positive and in its true form.

$$\therefore (9)_{10} - (5)_{10} = (4)_{10}$$

Note : The final carry bit acts as assign bit for the answer. It is 1 then the answer is positive, and it is 0 then the answer is negative.

Case 2 : A and B both positive with A < B

Ex. 2.6 : Perform $(4)_{10} - (9)_{10}$ using 2's complement method.

Soln. : Convert both the numbers to binary

$(4)_{10} = (1000)_2$ and $(9)_{10} = (1001)_2$

Step 1 : Obtain 2's complement of $(9)_{10}$:

Decimal	**Binary**	**2's complement**
$(9)_{10}$	$(1001)_2$	(0111)

Step 2 : Add $(4)_{10}$ to 2's complement of $(9)_{10}$:

$(4)_{10}$ 0 1 0 0

2's complement of $(9)_{10}$ + 0 1 1 1

carry 1

Final carry 0 1 0 1 1

Answer is negative and in 2's complement form '0' indicates that the result is negative and in its 2's complement form

Step 3 : Convert the answer in its true form:

Answer 1 0 1 1 In 2's complement
Subtract 1 : - 1

1 0 1 0

Inverts all bits

0 1 0 1

Thus the answer is – $(0101)_2$ i.e. $(- 5)_{10}$.

Case 3 : A and B both negative

Ex. 2.8 : Perform $(- 4)_{10} - (- 6)_{10}$ using 2's complement method.

Soln. : So we have to perform $(- 4)_{10} + (6)_{10}$

Step 1 : Convert number A to 2's complement of :

Decimal	**Binary**	**2's complement**
$(4)_{10}$	$(0100)_2$	(1100)

Step 2 : Add to 2's complement of $(4)_{10}$ and $(6)_{10}$:

2's complement of $(4)_{10}$ 1 1 0 0
$(6)_{10}$ + 0 1 1 0
Discard carry 1 0 0 1 0 Result in true form

Final carry 1 indicates that the result is positive and in the true form

∴ Answer : 0010 = $(2)_{10}$

∴ $(- 4)_{10} + (6)_{10} = (2)_{10}$

Ex. 2.9 : Perform $(- 6)_{10} - (- 2)_{10}$ using 2's complement method

Soln. : So we have to perform $(- 6)_{10} + (2)_{10}$

Step 1 : Obtain 2's complement of (6)10 :

Decimal	**Binary**	**2's complement**
$(6)_{10}$	$(0110)_2$	(1010)

Step 2 : Add $(2)_{10}$ to 2's complement of $(6)_{10}$:

2's complement of $(6)_{10}$	1	0	1	0

$(2)_{10}$ + 0 0 1 0

carry 1

Final carry 0 1 1 0 0

Answer in the 2's complement form

Final carry indicates that the answer is negative

Step 3 : Bring the answer into its true form :

Answer 1 1 0 0 In 2's complement

Subtract 1 - 1

1 0 1 1 Answer in 1's complement

Invert all bits

0 1 0 0 Answer in true from

∴ Answer = $(0100)_2 = (4)_{10}$ and it is negative

∴ $(6)_{10} - (2)_{10} = -(4)_{10}$

Case 4 : A = B

Ex. 2.10 : Perform $(6)_{10} - (6)_{10}$ using 2's complement method.

Soln. :

Step 1 : Obtain 2's complement of $(6)_{10}$

Decimal	**Binary**	**2's complement**
$(6)_{10}$	$(0110)_2$	(1010)

Step 2 : Obtain the binary equivalent of $(6)_{10}$ Add 2's complement of $(6)_{10}$ to it:

$(6)_{10}$ 0 1 1 0

2's complement of $(6)_{10}$ + 1 0 1 0

carry 1 1

Final carry 1 0 0 0 0 Answer in the true form

Final carry 1 indicates that the answer is positive and in true form

∴ $(6)_{10} - (6)_{10} = 0$

Ex. 2.11 : perform the subtraction using

1. 1's compliment method **2. 2's compliment**

(1 1 0 1 0) – (1 0 0 0 0)

Soln. : (1 1 0 1 0) – (1 0 0 0 0)

Using 1's compliment method:

Step 1 : Obtain1's complement of $(1\ 0\ 0\ 0\ 0)_2$

Binary	1's compliment
1 0 0 0 0	0 1 1 1 1

Step 2 : Add (1 1 0 1 0) and 1's complement of $(1\ 0\ 0\ 0\ 0)_2$:

		1	1	0	1	0
		0	1	1	1	1
Final carry	1	0	1	0	0	1
Final carry=1, Add the carry +						1
		0	1	0	1	0

$$\therefore 11010 - 10000 = 01010$$

2's Compliment method:

Step 1 : Obtain 2's compliment of 1 0 0 0 0:

Binary	**2's compliment**
1 0 0 0 0	1 0 0 0 0

Step 2 : Add(1 1 0 1 0) and 1's complement of (1 0 0 0 0)$_2$:

	1	**1**	**0**	**1**	**0**	
			1	**1**	**0**	**1**	**0**
	+	**1**	**0**	**0**	**0**	**0**
Final carry	**1**	**0**	**1**	**0**	**1**	**0**

Final carry = 1 Discard the carry. The carry indicates that result is positive and its true form

$$\therefore (1\ 1\ 0\ 1\ 0) - (1\ 0\ 0\ 0\ 0) = (1\ 0\ 1\ 0)$$

Ex. 2.12 : Perform the subtraction using 1's compliment method

$(0\ 0\ 1\ 1\ .\ 1\ 0\ 0\ 1)_2 - (0\ 0\ 0\ 1\ .\ 1\ 1\ 1\ 0)_2$

Step 1 : Obtain 1's complement of $(0001.1110)_2$:

Binary	1's complement.
0001.1110	1110.0001

Step 2 : Add$(0011.1001)_2$ and 1's complement of $(0001.1110)_2$

Ex. 2.12 : Perform the subtraction using 1's compliment method

$(0\ 0\ 1\ 1\ .\ 1\ 0\ 0\ 1)_2 - (0\ 0\ 0\ 1\ .\ 1\ 1\ 1\ 0)_2$

Step 1 : Obtain 1's complement of $(0001.1110)_2$:

Binary	1's complement.
0001.1110	1110.0001

Step 2 : Add$(0011.1001)_2$ and 1's complement of $(0001.1110)_2$

		1	1	1	0	.	0	0	0	1
+		0	0	1	1	.	1	0	0	1
Final carry	(1)	0	0	0	1	.	1	0	1	0
+										1
		0	0	0	1	.	1	0	1	0

$(0011.1001)_2 - (0001.1110)_2 = (0001.1011)_2$

Ex. 2.13 : Perform the subtraction using 1's and 2's compliment method

$(1010)_2 - (1011)_2$

Solution : 1. Subtraction using 1's complement method

$(1010)_2 - (1011)_2$

Step 1 : Obtain 1's complement of $(1011)_2$

1's complement

1011 ----------→ 0100

Step 2 : Add $(1010)_2$ and 1's complement of $(1011)_2$:

First Number	:	1	0	1	0	
1st complement of second number	: +	0	1	0	0	
Final carry	0	1	1	1	0	←Answer

As carry is not generated answer is negative and in 1's complements form.

Step 3 : Convert the answer into its true form:

Invert

1110 ----------> 0001

Thus the answer is $-(0001)_2$

2. Subtraction using 2's complement method

$(1010)_2 - (1011)_2$

Step1 : Obtain 2's complement of $(1011)_2$:

2's complement

(1011)----------> (0101)

Step2 : Add $(1010)_2$ and 2's complement of $(1011)_2$

1st number	: 1010
2's complement of 2nd number	: + 0101

Finally carry [0]	1111 ←Answer

As the final carry is not generated the answer is negative and in 2's complements form.

Step3 : Covert the answer into its true form:

2's complement

1111 ----------> 0001

Thus the answer is – $(0001)_2$

Ex. 2.14 : Perform following binary operation using 2' compliment method

1. $(1010)_2 - (101)_2$ **2.** $(1001)_2 - (1101)_2$

1. $(1010)_2 - (101)_2$

Let A = $(1010)_2$ and B= $(0101)_2$

Step 1 : Obtain 2's complement of $(0101)_2$

Invert Add-1

0101 ---------> 1010 ---------> 1011

∴ 2's complement of $(0101)_2$ is (1011)

Step 2 : Add A to 2's complement of B:

A:	1010
2's complement of B:	+ 1011

Discard carry: (1)	0101

As final carry is generated answer is positive and in true form.

As final carry is generated answer is positive and in true form.

Thus $(1010)_2 - (101)_2 = (101)_2$

2. $(1001)_2 - (1101)_2$

Let A = $(1001)_2$ and B= $(1101)_2$

Step 1 : Obtain 2's complement of B

2's complement

(1101) ----------> (0011)

Step 2 : Add A to 2's complement of B:

A:	1001
2's complement of B:	0011

Discard carry: (D)	1100

As final carry is generated answer is positive and in true form.

Step 3 : Convert answer in true form :

Answer True form.

Subtract **Invert**

1100 ⟶ **1011** ⟶ **0100**

Thus, answer is – $(0100)_2$

As final carry is not generated answer is negative and 2's complement form.

2.5 BINARY MULTIPLICATION

- The procedure used for binary multiplication is exactly same as that for the decimal multiplication.
- In fact binary multiplication is simpler than decimal ultiplication because only 0s and 1s are involved.

Rules of binary multiplication are as follows:

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

Generalized multiplication of two binary numbers:

Let the two binary numbers to be multiplied by 4-bit numbers $(A)_2 = A_3\ A_2\ A_1\ A_0$ and $(B)_2 = B_3\ B_2\ B_1\ B_0$. With A_0 and B_0 being LSBs. The generalized multiplication is shown in Fig.2.1.

				MSB			**LSB**	
				A_3	A_2	A_1	A_0	
			X	B_3	B_2	B_1	B_0	
Multiply($A_3A_2A_1A_0$)XB_0				$A_3\ B_0$	$A_2\ B_0$	$A_1\ B_0$	A_0B_0	
+ Multiply by B_1 :			$A_3\ B_1$	$A_2\ B_1$	$A_1\ B_1$	A_0B_1	-	Shift left by 1 position
+ Multiply by B_2 :		$A_3\ B_2$	$A_2\ B_2$	$A_1\ B_2$	$A_0\ B_2$	-	-	Shift left by 2 position
+ Multiply by B_3 :	$A_3\ B_3$	$A_2\ B_3$	A_1B_3	$A_0\ B_3$				Shift left by 3position

Addition with carry represents the results

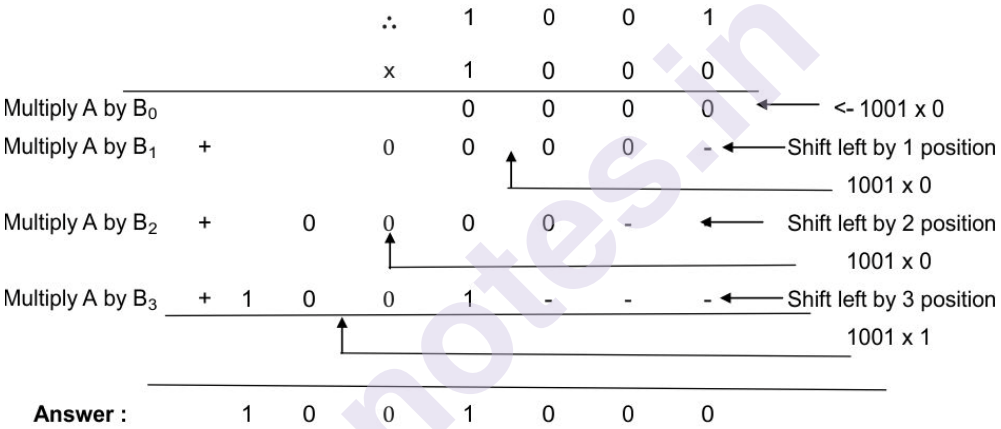
Fig. 2.1 : Generalized multiplication of two binary numbers

- The process of multiplication is illustrated in Fig. 2.1.
- Always start from LSB. Multiply = A_3, A_2, A_1 and A0 to obtain A_3 B_0, A_3 B_0, A_3 B_0, A_3 B_0 respectively.
- Shift left by 1 position by writing a "0" in the rightmost column and multiply (A_3 A_2 A_1 A_0) by B_1.
- Repeat the procedure for B_2 and B_3 with shifting left by 2 position and 3 positions respectively as shown Fig. 2.1.
- Add the entire product terms columns by column to obtain the answer

Ex. 2.15 : Multiply $(9)_{10}$ by $(8)_{10}$:

Soln. : Let A = A_3A_2A A_0 = $(1001)_2$

And B = $B_3B_2B_1B_0$ = $(1000)_2A$



$(1001)_2$ x $(1000)_2$ = $(1001000)_2$

Cross check : The decimal equivalent of the answer is obtain as follow:

Answer:	1	0	0	1	0	0	0
Binary weights:	64	32	16	8	4	2	1
Decimal Equivalent:	64+8	=72	9X8				

Ex 2.16 : Perform the binary multiplication 101.11x 111.01

			A		1	0	1.	1	1			
			B	x	1	1	1.	0	1			
					1	1	1	1	1			
					0	0	0	0	-			
+				1	0	1	1	1	-	-	←	Shift 1 by position
+			1	0	1	1	1	-	-	-	←	Shift 2 by position
+		1	0	1	1	1	-	-	-	-	←	Shift 3 by position
	1	0	1	0	0	1	1	0	1	1	←	Shift 4 by position

The binary point is placed after 4 position from LSB.

∴ Ans : 1 0 1 0 0 1 . 1 0 1 1

Cross check :

$A = (101.11)_2 = (5.75)_{10}$ and $B = (111.01)_2 = (7.25)_{10}$

$\therefore A \times B = (41.6875)_{10}$

Ans. :

1	0	1	0	0	1	.	1	0	1	1
32	+		8	+			1	+		0.5 +

$0.125+0.0625=(41.6875)_{10}$

Thus we have the correct answer

2.6 BINARY DIVISION

The division of binary numbers takes place in a similar way as that of decimal number. It called as the long division procedure.

Ex. 2.17 : Perform the following binary division 110 + 10

Soln.:

			1	1	
10)	1		1	0	
	1		0	↓	$110 = (6)_{10}$
			1	0	$10 = (2)_{10}$
	-		1	0	∴6+2=8
			0	0	

$(11)_2 = (3)_{10}$

Ex. 2.18 : Divide 1100 by 10

Soln. :

		1	1		← Quotient
100)	1	1	0	0	
	1	0	0	↓	
	0	1	0	0	
		1	0	0	
		0	0	0	← Remainder

$\therefore$ **Ans. :** $(11)_2$

Ex. 2.19 : Perform the division of $(110\ 110)_2 + (101)_2$

Soln.:

	1	0	1	0			← Quotient
101)	1	1	0	1	1	0	
	1	0	1				
	0	0	1	1	1		
			1	0	1		
			0	1	0	0	← Remainder

$\therefore$ Quotient = $(1010)_2 = (10)_{10}$

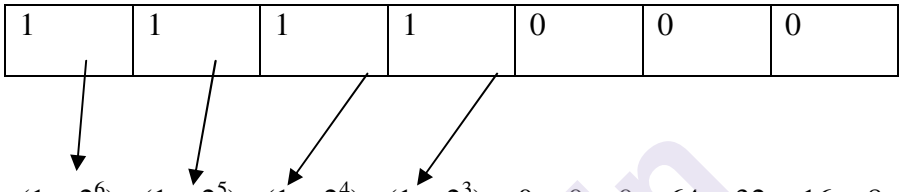
Remainder = $(0100)_2 = (4)_{10}$

Ex. 2.20 : Perform the following multiplication in binary number system: $(15)_{10}$ x $(8)_{10}$

Cross check the answer

Soln. : A = $(15)_{10} = (1111)_2$ and B = $(8)_{10} = (1000)_2$ B

					1	1	1	1
				x	1	0	0	0
Multiply by B_0					0	0	0	0
Multiply by B_1	+			0	0	0	0	-
Multiply by B_2	+		0	0	0	0	-	-
Multiply by B_3	+	1	1	1	1	-	-	-
Answer :		1	1	1	1	0	0	0



$(1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + 0 + 0 + 0 = 64 + 32 + 16 + 8 = (120)_{10}$

$\therefore (15)_{10} \times (8)_{10} = (120)_{10}$ …Ans.

2.7 ARITHMETIC IN OCTAL NUMBER SYSTEM

2.7.1 Octal Addition:

- The result of sum of two octal numbers is same as sum of their decimal equivalents as long as the decimal sum is less than 8.
- But if the decimal sum is equal to or greater than 8 then subtract 8 from it (in order to obtain the octal digit, and generate a carry 1. The octal addition is illustrated in Ex. 2.21

Ex.2.21 : Perform the addition of $(2)_8$ and $(4)_8$

Soln. : Note that the subscript 8 indicates that the numbers are octal numbers.

$$\therefore (2)_8 + (4)_8 = (6)_8$$

Note:

1. The addition takes place similar to that of decimal numbers.
2. As $(6)_8$ Is less than 8, no correction is necessary.

Ex.2.22 : Add $(7)_8$ and $(4)_8$

Soln. :

Step 1 : Add the numbers by assuming them to be decimal :

$$(7) + (4) = (11)_{10}$$

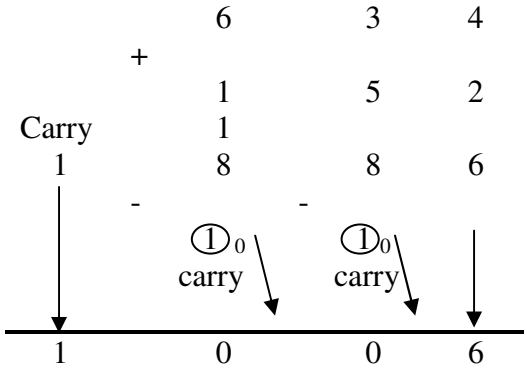
Step 2 : Correct the result, as it is greater than 8:

Two corrections will have to be carried out as follows:

1. Subtract 8 from the result: $(1\ 1\ )_{10} - (8)_{10} = 3$.
2. Generate a carry = 1.

$$\therefore (7)_8$$
$$+ (4)_8$$
$$\text{Carry } \boxed{1}\ \ 3$$

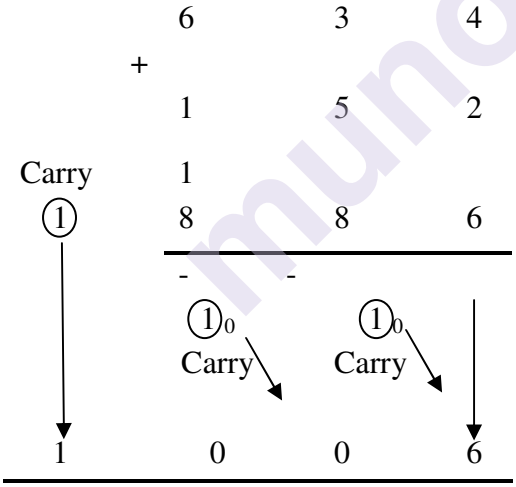
$$\therefore (7)_8 + (4)_8 = (13)_8$$



Note : Whenever column addition is greater than or equal to 8, subtract 8 and generate carry 1 and add this to next column

Ex. 2.23 : Add $(634)_8$ and $(152)_8$

Solution : The addition takes place as shown below



Note: Whenever column addition is greater than or equal to 8, subtract 8 and generate carry 1. Add this carry to next column.

Hence $(634)_8+(152)_8 =(1006)_8$

Ex. 2.24 : Add the octal number 354_8, 266_8, 123_8

Solution :

	3	**5**	**4_8**
+			
	2	**6**	**6_8**
+			
	1	**2**	**3_8**
Carry :	①	①	
	7	**14**	**13**
	-	**8**	**-** **8**
		①$_6$	**①$_5$**
		Carry	**Carry**
	7	**6**	**5** **Final Answer**

Note: Whenever column addition is greater than or equal to 8, subtract 8 and generate carry 1. Add this carry to next column.

$(354)_8 + (266)_8 + (123)_8 = (765)_8$

2.7.2 Subtraction of Octal Numbers:

The following methods can be used for octal subtraction:

1. Direct subtraction.
2. Convert the numbers to binary, perform the subtraction and convert the result back to octal.
3. Use the 7's complement method.
4. Use the 8's complement method

2.7.3 Method 1 : Direct Subtraction:

- In the direct subtraction of octal numbers, we use the same rules as used in decimal subtraction.
- That means, if the Minuend (number for which second number is to be subtracted) is less than the subtrahend (number to be subtracted) then we take borrow and return carry.
- The direct subtraction of octal numbers is illustrated in the following example.

Ex. 2.25 : Perform $(75)_8 - (68)_8$ without conversion.

Soln. : A= $(75)_8$ B= $(68)_8$

Borrow:		6	} $(13)_{10}$
A:	7	5	
-			
B:	6	8	
Carry:	①	} $(7)_{10}$	
Result:	0	5	

$\therefore (75)_8 - (68)_8 = (05)_8$

2.7.4 Octal Multiplication:

Now we are going to learn multiplication in octal system. If you directly multiply octal number it becomes bit complicated therefore normal procedure followed is as follows:

Step1 : Convert octal to binary. (both, multiplier and multiplicand.)
Step 2 : Perform simple binary multiplication.
Step 3 : After performing multiplication, whatever answer you get in binary, convert it to equivalent octal.

Ex. 2.26 : Perform $(12)_8 \times (7)_8$

Soln. :

Step 1 : Convert both octal number to binary:

$(12)_8 = (0\ 0\ 1\ 0\ 1\ 0) = (1\ 0\ 1\ 0)_2$

$(7)_8 = (1\ 1\ 1)_2 = (1\ 1\ 1)_2$

Step 2 : Perform binary multiplication :

```
                    1   0   1   0
                        1   1   1
                ---------------------
                    1   0   1   0
                1   0   1   0   -
            1   0   1   0   -   -
        ---------------------------
        1   0   0   0   1   1   0   Binary
        |       |       |
        1       0       6           (106)8
```

To cross check.

$(12)_8 \times (7)_8 = (1 \times 8^1 + 2 \times 8^0) \times [7 \times 8^0] = (10)_{10} \times (7)_{10} = (70)_{10}$

Consider result i.e. $(106)_8$

$(106)_8 = 1 \times 8^2 + 0 \times 8^1 + 6 \times 8^0 = (64 + 0 + 6)_{10} = (70)_{10}$

2.7.5 Octal Division:

For division in octal system, we follow the same steps i.e.

Step 1 : Convert octal to binary (both the given numbers).
Step 2 : Perform binary division.
Step 3 : Convert given binary quotient and remainder to octal.

Ex. 2.27 : perform $(24)_8 \div (4)_8$

Soln. :

Step 1 : Convert both numbers to binary

2	4	octal	4		101	Quotient
010	100	Binary	100	Divisor	100) 10100	
					100	
					0100	
					0100	
					0000	Remainder

2.8 HEXADECIMAL ARITHMETIC

In this section we are going to discuss the hexadecimal addition and subtraction.

2.8.1 Hex Addition:

1. The sum of two hex numbers is same as the sum of their decimal equivalents as long as a decimal sum is less than 16. (i. e. from 0 to 15)
2. But if the decimal sum is equal to or greater than 16 then subtract 16 from it (in order to obtain the hex digit) and generate a carry 1. The hex addition is illustrated in Ex. 2.28

Ex. 2.28 : Perform the addition of (8)H and (5)H.

Soln. : The subscript H indicates that both the numbers are hex numbers.

$$\therefore 8_H + 5_H = (13)_{10} = D_H$$

Note : 1. The addition takes place similar to that of decimal numbers.
2. As D_H is less than 16, no correction is necessary.

Ex. 2.29 : Add 9_H and 8_H.

Soln. :

Step 1 : Add the numbers by assuming them to be decimal numbers:

$$9 + 8 = (17)_{10}$$

Step 2 : Correct the result because it is greater than 16.

Two corrections will have to be carried out as follows:

1. Subtract 16 from the result : $(17)_{10} - (16)_{10} = 1$
2. Generate a carry = 1.

	$\therefore$	9_H
	+	8_H
Carry	1	7

Ex 2.30 : Add $C2_H$ and $3E_H$

Soln :

Step 1 : Add the digits by assuming them to be digital:

C 2 ---------> $(12)_{10}$ $(2)_{10}$

\+

\+ 3 E ---------> $(3)_{10}$ $(14)_{10}$

Step 2 : Correct the result:

$(15)_{10}$ $(16)_{10}$

\+ ① $-(16)_{10}$

$(16)_{10}$ ①

\- $(16)_{10}$

① 0

Final Answer (1 0 $0)_{16}$

Hence $(C2)_H + (3E)_H = (100)_H$

2.8.2 Hex Subtraction:

There are various methods used to perform the hex subtraction. They are as follows:

1. Direct subtraction.
2. Subtraction by converting the given numbers into binary.
3. Subtraction using 15's complement.
4. Subtraction using 16's complement.

2.8.3 Method 1: Direct Subtraction:

- This method is similar to the direct octal subtraction.
- If the Minuend is less than the subtrahend then we have to take borrow and return carry.
- The concept of direct hex subtraction is illustrated in the following examples.

Ex 2.31 (A): Perform the following hex subtraction without converting the numbers. (a) $(A)_{16} - (8)_{16}$ (b) $(73)_H - (1C)_H$

Soln. : A) $(A)_{16} - (8)_{16}$:

Borrow :

A : A_{10}}

B : 8 **$(A)_{16} - (8)_{16} = (2)_{16}$**

Carry :

Result : 2

(b) $(73)_{16} - (1C)_{16}$:

Borrow	**:**		**16**	$(19)_{10}$
A	**:**	**7**	**3**	
B	**:**	**-1**	**c**	$(2)_{10}$
Carry	**:**	**1**		
Result	**:**	**5**	**7**	$(12)_{10}$

2.8.4 Hexadecimal Multiplication:

If you directly multiply hex numbers, the thing becomes bit complicated. Therefore it will be better if one follows following steps

Step 1 : Convert hex to binary. (Multiplier and Multiplicand both).

Step 2 : Perform Simple binary multiplication.

Step 3 : After performing multiplication, whatever answer you get in binary, convert it to equivalent hex. This you can achieve by grouping 4 binary bits. Add extra zeros where required.

Ex 2.32 : Multiply $(72)_{16}$ and $(39)_{16}$

Soln. :

Step1 : Convert hex to binary :

$(72)_{16} = (0\ 1\ 1\ 1\ 0\ 0\ 1\ 0)_2 = (1\ 1\ 1\ 0\ 0\ 1\ 0)_2$

$(39)_{16} = (0\ 0\ 1\ 1\ 1\ 0\ 0\ 1)_2 = (1\ 1\ 1\ 0\ 0\ 1\ )_2$

Step 2 : perform binary multiplication :

						1	1	1	0	0	1	0	
					+		1	1	1	0	0	1	
						1	1	1	0	0	1	0	
					0	0	0	0	0	0	0	-	
				0	0	0	0	0	0	0	-	-	
			1	1	1	0	0	1	0	-	-	-	
		1	1	1	0	0	1	0	-	-	-	-	
	1	1	1	0	0	1	0	-	-	-	-	-	
	10	10	1	1	1	1	1						
1	1	0	0	1	0	1	1	0	0	0	1	0	binary

Step 3 : Convert binary to hex:

0 0 0 1	1 0 0 1	0 1 1 0	0 0 1 0
1	9	6	2

Ex 2.33 : Perform $(A2C)_{16}$ and $(B42)_{16}$

Soln. :

Step1 : Convert hex to binary :

$(A2C)_{16} = (1\ 0\ 1\ 0 \quad 0\ 0\ 1\ 0 \quad 1\ 1\ 0\ 0)_2$

$(B42)_{16} = (1\ 0\ 1\ 1 \quad 0\ 1\ 0\ 0 \quad 0\ 0\ 1\ 0)_2$

											1	0	1	0	0	0	1	0	1	1	0	0
										x	1	0	1	1	0	1	0	0	0	1	0	0
											0	0	0	0	0	0	0	0	0	0	0	0
										1	0	1	0	0	0	1	0	1	1	0	0	-
									0	0	0	0	0	0	0	0	0	0	0	0	-	-
								0	0	0	0	0	0	0	0	0	0	0	0	-	-	-
							0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-
						0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-
					1	0	1	0	0	0	1	0	1	1	0	0	-	-	-	-	-	-
				0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-
			1	0	1	0	0	0	1	0	1	1	0	0	-	-	-	-	-	-	-	-
		1	0	1	0	0	0	1	0	1	1	0	0	-	-	-	-	-	-	-	-	-
	0	0	0	0	0	0	0	0	0	0	0	0		-	-	-	-	-	-	-	-	-
1	0	1	0	0	0	1	0	1	1	0	0	-		-	-	-	-	-	-	-	-	-
1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0

Ex 2.34 : Perform $(FA.2)_{16}$ and $(11.D)_{16}$

Soln. :

Step1 : Convert hex to binary :

$(FA.2)_{16} = (1111\ 1010\ .\ 0010)_2$

$= (111\ 11010\ .\ 001)_2$

$(11.D)_{16} = (0001\ 0001\ .\ 1101)_2$

$= (10001\ .\ 1101)_2$

Perform binary multiplication:

										1	1	1	1			1	0	1	0	0	0	1	0
											x		1			0	0	0	1	1	1	0	1
											1	1	1	1		1	0	1	0	0	0	1	0
										0	0	0	0	0		0	0	0	0	0	0	0	-
								1	1	1	1	1	0		1	0	0	0	0	0	-	-	
							1	1	1	1	1	0	1		0	0	0	1	0	-	-	-	
						1	1	1	1	1	0	1	0		0	0	1	0	-	-	-	-	
					0	0	0	0	0	0	0	0	0		0	0	0	-	-	-	-	-	
				0	0	0	0	0	0	0	0	0	0		0	0	-	-	-	-	-	-	
			0	0	0	0	0	0	0	0	0	0	0		0	-	-	-	-	-	-	-	
		1	1	1	1	1	1	0	0	0	0	1	0		-	-	-	-	-	-	-	-	
↓																							
1	0	0	0	1	0	1	1	0	0	1	1	1			0	1	0	1	1	0	0	0	

Step 2 : Convert binary to hex:

0001	0001	0110	0111	0101	1010
1	1	6	7	5	A

∴ Ans. : $(1167.5A)_{16}$

2.8.5 Hex Division:

To perform division in hex number you have to follow steps given below :

Step 1 : Convert hex number to equivalent binary.

Step 2 : Perform binary division.

Step 3 : Convert final answer in binary i.e. Quotient and Remainder to equivalent hex.

Ex. 2.35 : Perform $(24)_H + (8)_H$.

Step 1 : Convert both numbers into binary :

2	4	Hex	8
0010	0100	Binary	1000

Step 2 : Perform division

```
                100 Quotient
            ------------------
1000)  100100
       1000
       ------
            100 Remainder
```

Step 3 : Convert the answer into hex

Quotient : $100 = (0100)_2 = (4)_H$

Remainder : $100 = (0100)_2 = (4)_H$

2.9 BCD ARITHMETIC

BCD is a binary code of the ten decimal digits. It is not a binary equivalent.

To perform BCD addition:

- Add the BCD digits as regular binary numbers.
- If the sum is 9 or less and no carry was generated, it is a valid BCD digit.
- If the sum produces a carry, the sum is invalid and the number 6 (0110) must be added to the digit.
- If the sum is greater than nine, the sum is invalid and the number 6 (0110) must be added to the digit.
- Repeat for each of the BCD digits.

Addition:

There are four basic rules to adding two binary digits.

0 + 0 = 0 carry 0

0 + 1 = 1 carry 0

1 + 0 = 1 carry 0

1 + 1 = 0 carry 1

- Binary digits are added two at a time and any carry must be carried over to the next higher column of digits.
- To get the sum of three digits, add the first two and then add the sum to the third digit.
- To add large binary numbers add one column of digits starting with the least significant position. Add any carry into the next significant column.

For Example:

Add the binary numbers 00111 and 10101 and show the equivalent decimal addition

0	1	1					
0	0	1	1	1			7
1	0	1	0	1		2	1
1	1	1	0	0	=	2	8

Subtraction:

There are four possible combinations when subtracting two binary digits.

0 – 0 = 0 borrow 0

0 – 1 = 1 borrow 1

1 – 0 = 1 borrow 0

1 – 1 = 0 borrow 0

- The borrow from a column of digits must be subtracted from the next most significant column.
- To subtract a large binary number from another large binary number, a borrow may need to be carried over several bit positions.

For Example:

Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

	1	1	1				
~~1~~	~~0~~	~~1~~	0	1		2	1
0	0	1	1	1			7
1	1	1	0	0	=	1	4

2.10 EXCESS-3 ARITHMETIC

2.10.1 Excess - 3 Addition:

To add the two excess - 3 numbers A and B follow the procedure given below : Steps to be followed :

Step1 : Add the two excess - 3 numbers using the rules of binary addition.

Step2 : If carry is 1, then add $(0011)_2$ or $(3)_{10}$ to the sum.

Step3 : If carry is 0, then subtract $(0011)_3$ or $(3)_{10}$ from the sum.

Ex. 2.36 : Add $(7)_{10}$ and $(6)_{10}$ in excess - 3.

Soln. : Convert $(7)_{10}$ and $(6)_{10}$ in excess - 3.

$(7)_{10} = (1010)_{XS-3}$ and $(6)_{10} = (1001)_{XS-3}$

Step 1 : Add two excess - 3 numbers :

$$\begin{array}{rl} & 1\ 0\ 1\ 0 \quad \rightarrow \text{Excess - 3 for } (7)_{10} \\ + & 1\ 0\ 0\ 1 \quad \rightarrow \text{Excess - 3 for } (6)_{10} \\ \hline \text{Final carry} \rightarrow & \mathbf{1}\ 0\ 0\ 1\ 1 \quad \rightarrow \text{Sum} \end{array}$$

Step 2 : Carry = 1 so add 0011 to the sum :

$$\begin{array}{cccc|cccc|l} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & \leftarrow \text{sum} \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \leftarrow \text{Add(3)} \\ \hline 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \end{array}$$

1_{10} $\qquad$ 3_{10}

$$\therefore (7)_{10} + (6)_{10} = (13)_{10}$$

Ex. 2.37 : Add $(2)_{10}$ and $(3)_{10}$ in excess - 3.

Soln. : Convert $(2)_{10}$ and $(3)_{10}$ in excess - 3.

$(2)_{10} = (0\ 1\ 0\ 1)_{XS-3}$, $(3)_{10} = (0\ 1\ 1\ 0)_{XS-3}$

Step 1 : Add two excess - 3 numbers :

$$\begin{array}{rccccc} & 1 & 0 & 1 & 0 & \leftarrow \text{Excess – 3 for } (7)_{10} \\ + & 1 & 0 & 0 & 1 & \leftarrow \text{Excess – 3 for } (6)_{10} \\ \hline \text{Final carry} \leftarrow & 1 & 0 & 0 & 1 & 1 \quad \text{Sum} \end{array}$$

Step 2 : Carry = 0 so subtract 0011 from the sum :

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \leftarrow \text{Sum} \\ 0 & 0 & 1 & 1 \leftarrow \text{Subtract (3)} \\ \hline 1 & 0 & 0 & 0 \end{array}$$

$(5)_{10}$

$$\therefore (2)_{10} + (3)_{10} = (5)_{10}$$

2.10.2 Excess - 3 Subtraction:

The steps to be followed for the Excess - 3 subtraction A - B are as follows : **Steps to be followed** :

Step 1 : Take the complement of B (subtrahend).

Step 2 : Add A (minuend) and complement of B using the rules of binary addition.

Step 3 : If carry = 1, then add $(3)_{10}$ to the result and end around the carry. The result is positive.

Step 4 : If carry = 0, then subtract $(3)_{10}$ from the sum obtained in step 2 and take the complement. The result is negative.

Ex. 2.38 : Perform the subtraction $(8)_{10} - (3)_{10}$ in excess - 3.

Soln. :

Step1 : Convert the numbers into excess 3 :

$(8)_{10} = (\ 1\ 0\ 1\ 1\ )_{XS\text{-}3}$ and $(3)_{10} = (\ 0\ 1\ 1\ 0\ )_{XS\text{-}3}$

Step2 : Take complement of (0 1 1 0) :

Number B = 0 1 1 0

Complement of B =1 0 0 1

Step 3 : Add A and complement of B :

```
                1   0   1   1   ← Excess -3 for (8)10
        +       1   0   0   1   ← Complement of B
        +           1   1
  ------------------------------
Final carry 1   0   1   0   0
                    Sum
```

Step 4 : If carry = 1, then add (3)10 to the sum and end around carry :

```
                1
                    0   1   0   0   Sum
            +       1   0   0   1   Add (3)10
  ------------------------------
Add end             1   0   1   1   1
around carry
                                →   1
                1   1       1
  ------------------------------
                1   0       0   0   Final answer in excess -3
                    (5)10
```

Ex. 2.39 : Perform the subtraction $(3)^{10} - (8)_{10}$ in excess - 3.

Soln.:

Step1 : Convert the numbers into excess 3 :

$(3)_{10} = (0\ 1\ 1\ 0)_{XS\text{-}3}$ and $(8)_{10} = (1\ 0\ 1\ 1)_{XS\text{-}3}$

Step2 : Take complement of (1 0 1 1) :

Number B = 1 0 1 1

Complement of B =0 1 0 0

Step 3 : Add A and complement of B :

```
    0 1 1 0    A
+   0 1 0 0    Complement of B
  ----------
```

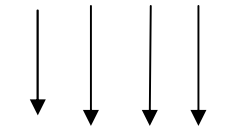
Final carry = 0 1 0 1 0

Step 4 : Final carry = 0, then subtract $(3)_{10}$ from sum :

```
    1  0   1  0   Sum
-   0  0   1  1   Add (3)10
  ---------------
    0  1   1  1   Final carry is negative and in
                  complemented excess -3form
```

Step 5 : Take complement of final answer:

0 1 1 1 Answer in compliment XS-3 for compliment



1 0 0 0 Answer in true XS-3 and negative

$(-5)_{10}$

Thus, $(3)_{10} - (8)_{10} = (-5)_{10}$

Ex. 2.40 : Perform the following addition in excess – 3 code.

$(9)_{10} + (6)_{10}$

Sol. : 9+6

Decimal		Excess -3	
9		1100	
+ 6		1001	
15	Final Carry	0101	**Carry = 1, so add 3 to sum and carry**
	(1)		
	+ 0011	0011	**Add 3**
	11	111	
	0100	1000	**Final answer in excess - 3**
	$(1)_{10}$	$(-5)_{10}$	

$\therefore (9)_{10} + (6)_{10} = (15)_{10}$

Ex. 2.41 : Subtract $(168)_{10}$ from in $(234)_{10}$ in Excess – 3.

Soln :

Step 1 : Obtain the excess - 3 equivalent of given numbers:

Number B : $(168)_{10}$ = 0100 1001 1011

Number A : $(234)_{10}$ = 0101 0110 0111

Step 2 : Take the complement of number B :

Number B = 0100 1001 1011

Compliment of Number B = 1011 0110 0100

Step 3 : Add A and compliment of B :

Decimal		Excess 3			
234		0101	0110	0111	
168	+	1011	0110	0100	Compliment of B
66		111	11	1	
Final carry =1		1 0000	1100	1011	
Thus the answer positive add the end around + carry				→ 1	
		0000	1100	1100	Sum

Step 4 : Add or subtract 011 for correction : Add 0011 to 0000 because carry = 1 was produced along with 0000

	0000	1100	1100	Sum in step 3
+	0011			Add 0011
	0011	1100	1100	

Now subtract 0011 from 1100 as the carry =0 along with them

0011	1100	1100	Sum in step 3
-	0011	0011	Subtract 0011
0011	1001	1001	Answer in XS -3 form
0	6	6	

REVIEW QUESTIONS

Q. 1 Explain binary arithmetic in brief

Q. 2 Explain binary arithmetic with suitable example

Q. 3 Solve following examples

1. $(10101)_2 + (10111)_2$
2. $(11101)_2 + (101010)_2$
3. $(11111)_2 - (10111)_2$
4. $(1110)_2 / (10)_2$
5. $(1011)_2 * (111)_2$

Q.4 Explain arithmetic's in octal number with suitable example

Q. 5 Explain arithmetic's in hexadecimal number with suitable example

Q. 6 What is BCD arithmetic?

*****

Unit 2

3

LOGIC GATES

Unit Structure

3.0 INTRODUCTION

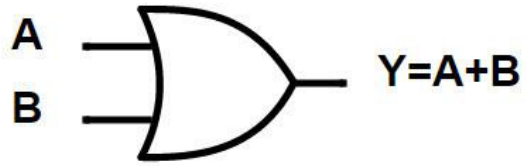
- The logic gate is the most basic building block of any digital system, including computers. Each one of the basic logic gates is a piece of hardware or an electronic circuit that can be used to implement some basic logic expression.
- While laws of Boolean algebra could be used to do manipulation with binary variables and simplify logic expressions, these are actually implemented in a digital system with the help of electronic circuits called logic gates. The three basic logic gates are the OR gate, the AND gate and the NOT gate.

3.1 OR GATE

- An OR gate performs an ORing operation on two or more than two logic variables.
- The OR operation on two independent logic variables A and B is written as "Y = A+B " and reads as Y equals A OR B.

- An OR gate is a logic circuit with two or more inputs and one output.
- The output of an OR gate is LOW only when all of its inputs are LOW. For all other possible input combinations, the output is HIGH.
- Figure below shows the circuit symbol and the truth table of a two-input OR gate. The operation of a two-input OR gate is explained by the logic expression:

Boolean Expression : $Y = A+B$



Symbol for OR gate

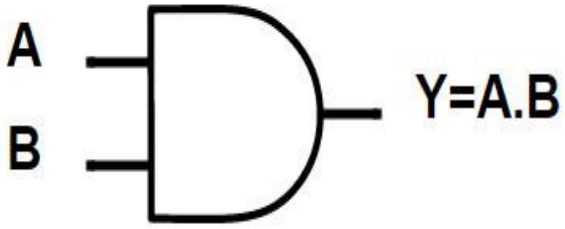
TRUTH TABLE:

Input		**Output Y= A+B**
A	B	
0	0	0
0	1	1
1	0	1
1	1	1

3.2 AND GATE

- An AND gate is a logic circuit having two or more inputs and one output.
- The AND operation on two independent logic variables A and B is written as "Y = A• B " and reads as Y equals "A AND B"
- The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW.
- When interpreted for a positive logic system, this means that the output of the AND gate is a logic ''1'' only when all of its inputs are in logic ''1'' state. In all other cases, the output is logic ''0''.
- The Boolean expression, logic symbol and truth table of a two-input AND gate are shown below;

Boolean Expression : Y = A• B



Symbol for AND Gate

TRUTH TABLE:

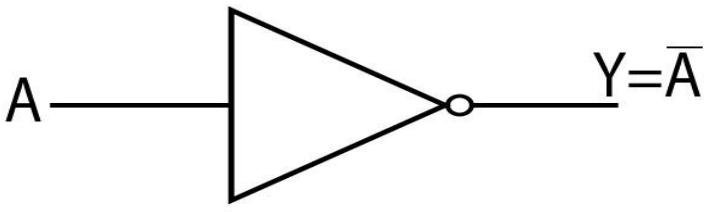
Input A	Input B	Output Y= A•B
0	0	0
0	1	0
1	0	0
1	1	1

3.3 NOT GATE

- A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input.
- That is, a LOW input produces a HIGH output, and vice versa.
- When interpreted for a positive logic system, a logic ‘‘0’’ at the input produces a logic ‘‘1’’ at the output, and vice versa.
- It is also known as a ‘‘complementing circuit’’ or an ‘‘inverting circuit’’ or hex inverter’’.

The Boolean expression, logic symbol and truth table of a NOT gate are shown below;

Boolean Expression : $Y = \overline{A}$



Symbol for NOT gate

TRUTH TABLE:

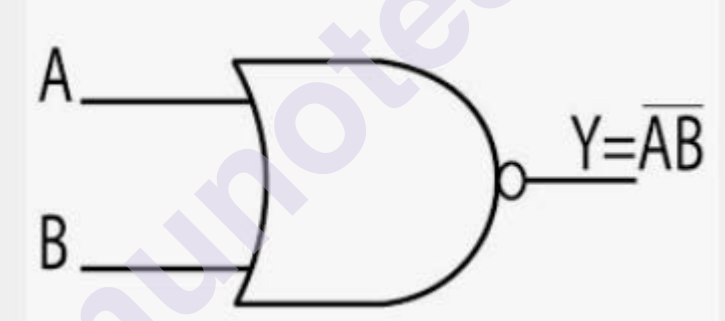
Input	Output **Y=** $\overline{A}$
0	1
1	0

Symbol for NOT gate

3.4 NAND GATE

- A NAND gate is a logic circuit having two or more inputs and one output.
- The NAND operation on two independent logic variables A and B is written as "$Y = \overline{AoB}$
- **"Y =A.B** " and reads as Y equals "A AND B the whole BAR"
- The output of an NAND gate is HIGH when any one of its input is low
- The Boolean expression, logic symbol and truth table of a two-input AND gate are shown below;

Boolean Expression : "$Y = \overline{A \bullet B}$



Symbol for NAND Gate

Truth Table:

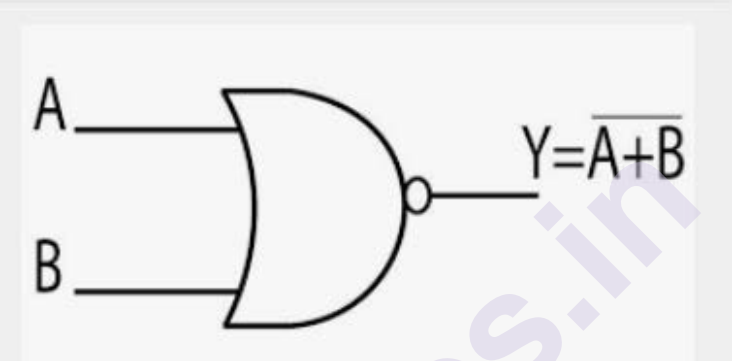
Input		**Output**
A	B	"$Y = \overline{A \bullet B}$
0	0	**1**
0	1	1
1	0	1
1	1	0

3.5 NOR GATE

- A NOR gate performs an complementary operation of OR gate on two or more than two logic variables.

- The NOR operation on two independent logic variables A and B is written as
- "$Y = \overline{A + B}$" " and reads as Y equals "A OR B the whole BAR".
- The output of an OR gate is HIGH only when all of its inputs are LOW. For all other possible input combinations, the output is LOW.
- Figure below shows the circuit symbol and the truth table of a two-input OR gate. The operation of a two-input OR gate is explained by the logic expression:

Boolean Expression "$Y = \overline{A + B}$"



Symbol for NOR gate

TRUTH TABLE:

Input		Output
A	B	"$Y = \overline{A + B}$"
0	0	**1**
0	1	0
1	0	0
1	1	0

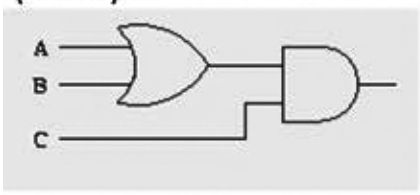
3.6 LAWS OF BOOLEAN ALGEBRA

Name	AND form	OR form
Identity Law	1 • A = A	0 + A = A
Null Law	0 • A = 0	1 + A = 1
Idempotent Law	A • A = A	A + A = A
Inverse Law	A • Ā = 0	A + Ā = 1
Commutative Law	A B = B A	A + B = B + A
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) = AB + AC
Absorption Law	A(A+B) = A	A + AB = A
De Morgan's Law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

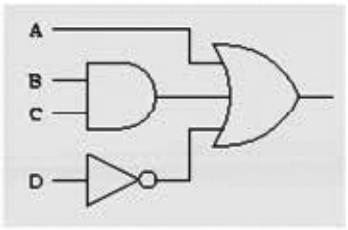
Question : Drawing logic diagram for Boolean expression given below

1) **(A + B) C**
2) **A + BC +** $\overline{D}$
3) **AB +** $\overline{AC}$
4) $\overline{(A+B)}(C+D)\overline{C}$

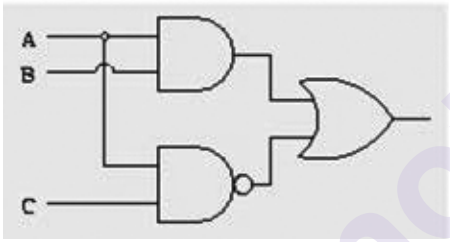
(A + B)C



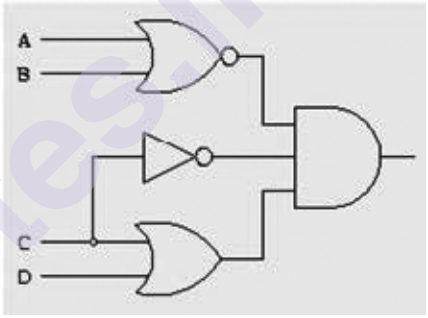
A + BC + $\overline{D}$



AB + $\overline{AC}$



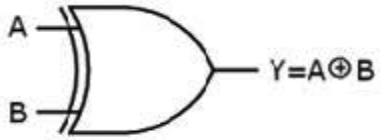
$\overline{(A + B)}(C + D)\overline{C}$



3.7 EXCLUSIVE-OR Gate(EX-OR)

- An EX-OR gate is a logic circuit having two or more inputs and one output.
- The EX-OR operation on two independent logic variables A and B is written as **"Y = A ⊕ B "** and reads as Y equals " EX-OR B".
- The output of an EX-OR gate is a logic '1' when the inputs are unlike and a logic '0' when the inputs are like for two input EX-OR gate, while if the input is more than two then output is logic "1" when odd number of inputs are "HIGH" and logic "0" when even number of inputs are "LOW".
- The Boolean expression, logic symbol and truth table of a two-input EX-OR gate are shown below;

Boolean Expression : Y = A ⊕ B = $\overline{A}B + A\overline{B}$



Two Input EX-OR Gate Symbol

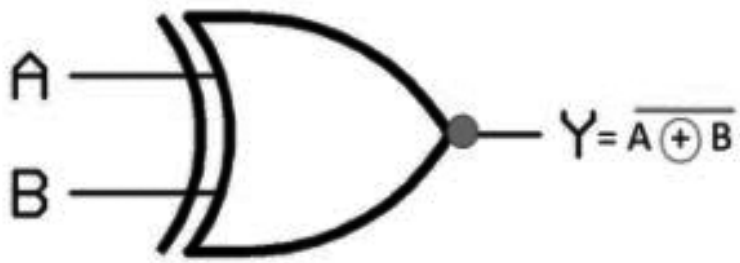
Two Input TRUTH TABLE:

Input		Output
A	**B**	**Y = A ⊕ B**
0	0	**1**
0	1	0
1	0	0
1	1	1

3.8 EXCLUSIVE-NOR Gate(EX-NOR)

- The EX-OR operation on two independent logic variables A and B is written as **"Y = A ⊕ B "**and reads as Y equals " EX-NOR B".
- The output of an EX-NOR gate is a logic '1' when the inputs are like and a logic '0' when the inputs are unlike for two input EX-OR gate, while if the input is more than two then output is logic "1" when even number of inputs are "HIGH" and logic "0" when odd number of inputs are "LOW".
- The Boolean expression, logic symbol and truth table of a two-input EX-NOR gate are shown below;

Boolean Expression : Y = $\overline{A \oplus B}$



Two Input EX-NOR Gate Symbol

Two Input TRUTH TABLE:

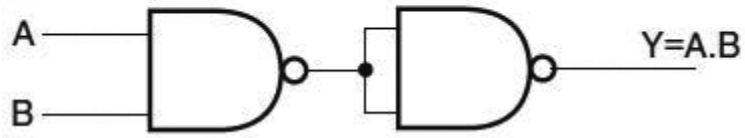
Input		Output $Y = \overline{A \oplus B}$
A	B	
0	0	1
0	1	0
1	0	0
1	1	1

3.9 UNIVERSAL GATE

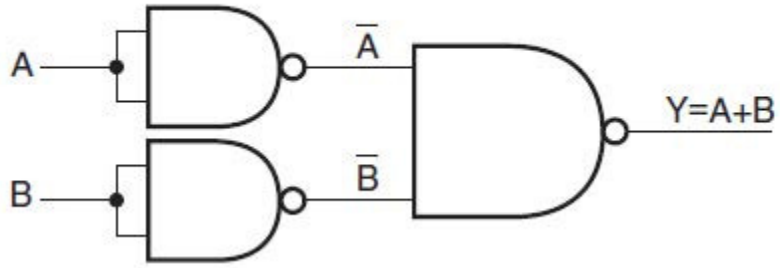
- OR, AND and NOT gates are the three basic logic gates as they together can be used to construct the logic circuit for any given Boolean expression.
- NOR and NAND gates have the property that they individually can be used to hardware-implement a logic circuit corresponding to any given Boolean expression. That is, it is possible to use either only NAND gates or only NOR gates to implement any Boolean expression.
- This is so because a combination of NAND gates or a combination of NOR gates can be used to perform functions of any of the basic logic gates.
- It is for this reason that NAND and NOR gates are universal gates

NAND as NOT:

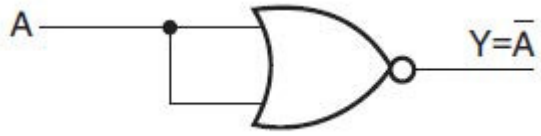
NAND as AND:



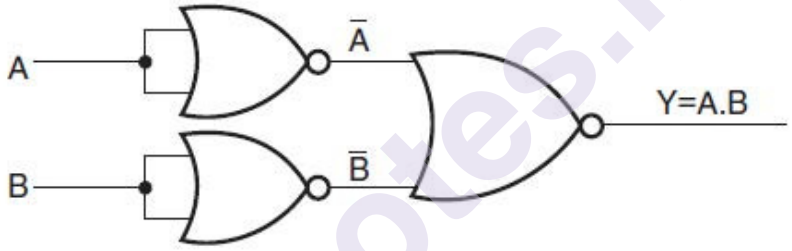
NAND as OR:



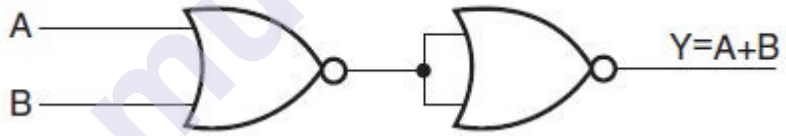
NOR as NOT:



NOR as AND:



NOR as OR:



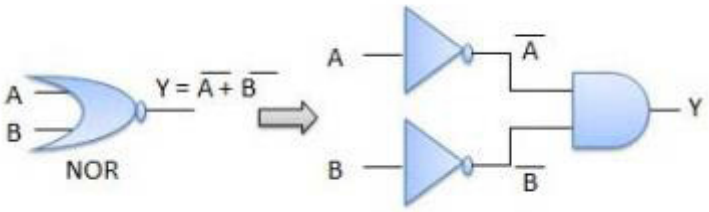
3.10 DE MORGAN'S THEORMS

First Theorem:

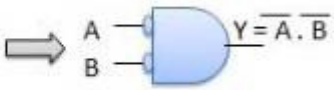
- It states that, the complement of a sum equals the product of complements.

OR

- It states that, the output of NOR gate is equal to the output of bubbled AND gate.



NOR ☰ Bubbled AND



Truth Table:

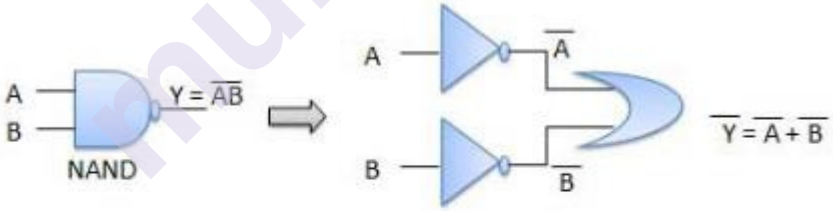
Input		L.H.S. A+B	R.H.S $\overline{A}.\overline{B}$
A	**B**	**1**	**1**
0	**0**	**0**	**0**
0	**1**	**0**	**0**
1	**0**	**0**	**0**

Second Theorem:

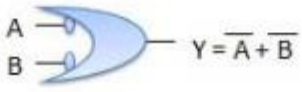
- It states that, the complement of a product equals the sum of complements.

OR

- It states that, the output of NAND gate is equal to the output of bubbled OR gate.



NAND ☰ Bubbled OR



Truth Table:

Input		L.H.S	R.H.S
A	B	$\overline{A+B}$	$\overline{A}\cdot\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

3.11 UNIT END QUESTION

1. For the logic expression Y=AB'+A'B. Obtain the truth table, name the gate and operation performed and symbol for it also realize this using AND,OR,NOT gates.

2. Prove the given Boolean expression using Boolean laws and draw the circuit for it using NAND gates only.
   A.B+A'B+A'B'=A'+B

3. State and prove De-Morgan's theorem and realize it using basic gates.

4. What is meant by universal logic gate? Draw logic circuits showing construction of Ex-OR gate using NAND gate and using NOR gate.

5. Draw logic circuit and make truth table to prove the following Boolean theorems
   i) A.0=0 ii) (A.B)C=A(B.C)

6. Using rules of Boolean algebra, solve y = (x+z)(x'+y+z). Draw a logic circuit using suitable gates to implement the simplified equation.

*****

4

SOP AND POS REPRESENTATION OF LOGICAL EXPRESSIONS AND KARNAUGH MAP

Unit Structure

4.0 SOP AND POS REPRESENTATION OF LOGICAL EXPRESSIONS

- Any logic expression can be expressed in the following two standard forms:

i) Sum-Of-Products (SOP) form ii) Product of Sums (POS) form

Sum-Of-Products (SOP) form:

$$Y = AB + BC + AC$$

- The above expression is a Sum of three Product terms i.e.,
- A.B, B.C and A.C.
- Therefore such expressions are called expressions in Sum-Of-Products (SOP) form.
- In a SOP sums are logical OR functions while products are logical AND functions.
- In above expression A, B and C are literals or inputs while Y is output of a combinational circuit.
- Some more examples of SOP are as follows:
- $Y=ABC+A.\overline{B}.D+B.D+\overline{A}.\overline{B}.C$

- $X=PQ+P.\overline{Q}.R+\overline{P}.\overline{R}$
- Thus in an SOP in each product term there can be one or more than literals ANDed together and then these product terms are logically ORed.

Product-of-Sums (POS) form:

Y = (A+B), (B+C), (A+C)

- The above expression is a Product of three Sum terms i.e., (A+B), (B+C) and (A+C).
- Therefore such expressions are called expressions in Product-Of-Sum (POS) form.
- In a POS sums are logical OR functions while products are logical AND functions.
- In above expression A, B and C are literals or inputs while Y is output of a combinational circuit.
- Some more examples of POS are as follows
- $X=(A+B+C).(A+\overline{C}+D).(\overline{A}+\overline{B}+C)$
- $M=(X+Y).(X+\overline{Y}+Z).(\overline{X}+\overline{Y})$
- Thus in a POS in each sum term there can be one or more than literals ORed together and then these Sum terms are logically ANDed
- **Standard SOP and POS forms:**

  In a standard SOP or POS each term may contain one, two or any number of literals. It is not necessary that each term should contain all the literals.
- **Canonical SOP and POS forms:**

  In a canonical SOP or POS each term contains all the literals in their complimented or uncomplemented forms**.**

Sr. No.	Expressions	Types
1.	$Y = A.B + A.B.\overline{C} + \overline{A}.B.C$	Standard SOP
2.	$Y = A.B + A.\overline{B} + \overline{A}.\overline{B}$	Canonical SO
3.	$Y = (A + C).(A + B + \overline{C}).(\overline{A} + B + C)$	Standard POS
4.	$Y = (A + \overline{B} + C).(A + B + \overline{C})(\overline{A} + B + C).$	Canonical POS

4.1 CONCEPT OF MINTERM AND MAXTERM

- Each individual term in a canonical SOP form is called as Minterm.
- Each individual term in the canonical POS is called as Maxterm

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

Each individual term is called Minterm

Canonical POS $Y = \underbrace{(A+B)}.\ \underbrace{(A+\overline{B})}$

Each individual term is called Maxterm

- The following table shows the minterms and maxterms for a three variable logic function Y=F (A, B, C). Let Y be the output and A, B, C be the inputs.
- The number of minterms and maxterms is $2^3 = 8$ In general for 'n' number of variables the number of minterms or maxterms will be 2^n.
- Each minterm is represented by mi and each maxterm is represented by Mi where i 0, 1,… ,2n – 1.

In this case i 0, 1, …, 7

Variables			Minterms	Maxterms
A	**B**	**C**	m_i	M_i
0	**0**	**0**	$\overline{A}.\overline{B}.\overline{C} = m_0$	$A+B+C = M_0$
0	**0**	**1**	$\overline{A}.\overline{B}.C = m_1$	$A+\overline{B}+C = M_1$
0	**1**	**0**	$\overline{A}.B.\overline{C} = m_2$	$A+\overline{B}+C = M_2$
0	**1**	**1**	$\overline{A}.B.C = m_3$	$A+\overline{B}+\overline{C} = M_3$
1	**0**	**0**	$A.\overline{B}.\overline{C} = m_4$	$\overline{A}+B+C = M_4$
1	**0**	**1**	$A.\overline{B}.C = m_5$	$\overline{A}+B+\overline{C} = M_5$
1	**1**	**0**	$A.B.\overline{C} = m_6$	$\overline{A}+\overline{B}+C = M_6$
1	**1**	**1**	$A.B.C = m_7$	$\overline{A}+\overline{B}+\overline{C} = M_0$

As we can see, minterm is product of given variables where logic 0 is represented by complemented variable, while logic 1 is represented by uncomplemented variable. Whereas maxterm is sum of given variables

where logic 1 is represented by complemented variable, while logic 0 is represented by uncomplemented variable.

4.2 MINIMIZATION WITH KARNAUGH MAPS

- Karnaugh map technique which provides a systematic method for simplifying and manipulating Boolean expressions.
- In this technique, the information contained in a truth table or available in POS or SOP form is represented on Karnaugh map (K-map).
- This is perhaps the most extensively used tool for simplification of Boolean functions.
- Although the technique may be used for any number of variables, it is generally used up to six variables beyond which it becomes very cumbersome.
- Below figures shows the K-maps for two, three and four variable.

- **2 variables Karnaugh map**

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

- **3 variables Karnaugh map**

C \ AB	00	01	11	10
0	$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\overline{C}$
1	$\overline{A}\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$

- **4 variables Karnaugh map**

CD \ AB	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$AB\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$
01	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}B\bar{C}D$	$AB\bar{C}D$	$A\bar{B}\bar{C}D$
11	$\bar{A}\bar{B}CD$	$\bar{A}BCD$	$ABCD$	$A\bar{B}CD$
10	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}BC\bar{D}$	$ABC\bar{D}$	$A\bar{B}C\bar{D}$

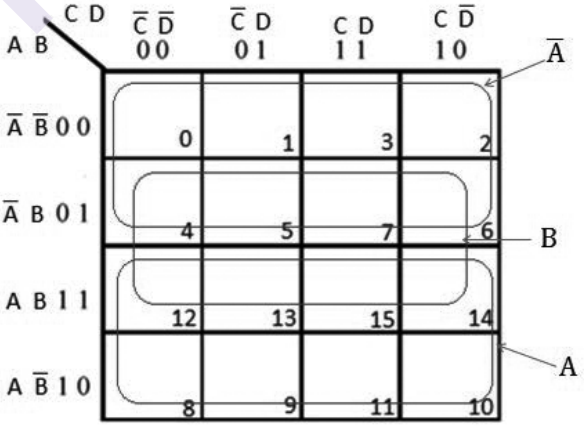
4.2 WAY OF GROUPING (PAIRS, QUADS, OCTETS)

- **Pairs:** group of two adjacent 1's in SOP K-map or two adjacent 0's in OS K-map)
- **Quads:** A group of four adjacent 1's in SOP K-map or four adjacent 0's in (POS K-map)
- **Octets:** group of eight adjacent 1's in SOP K-map or eight adjacent 0's in (POS K-map)

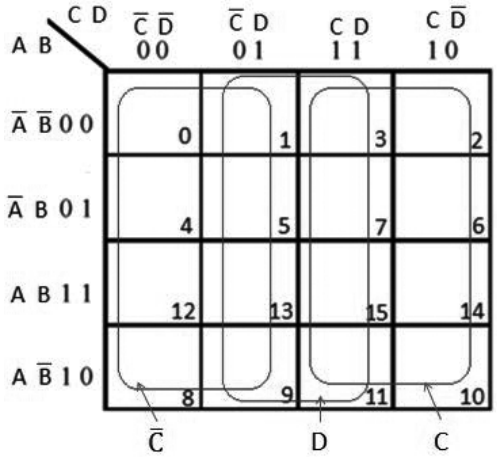
Grouping for eight adjacent Ones (Octets):

- Group the adjacent 1s in the given K-map and write the common variables eliminating uncommon variables. In reduction technique by forming octet three variables gets eliminated.
- Possible Octets in 4-variable K-map: (Note: octet is not possible in 2 variables K-map. While in 3-variable K map octet will select all the blocks making it equal to 1)

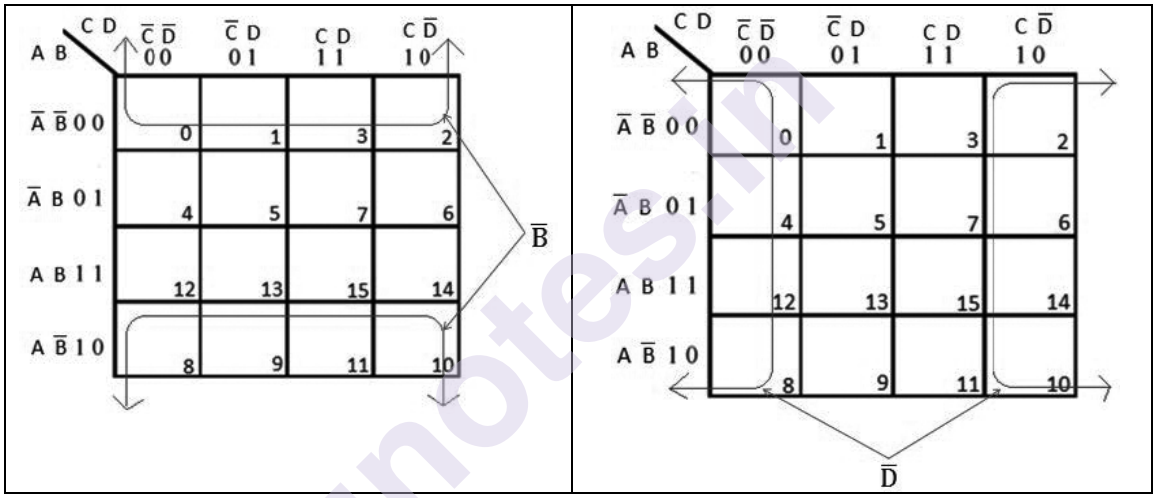
Adjacent Horizontal:



Adjacent vertical:



Fold-back:

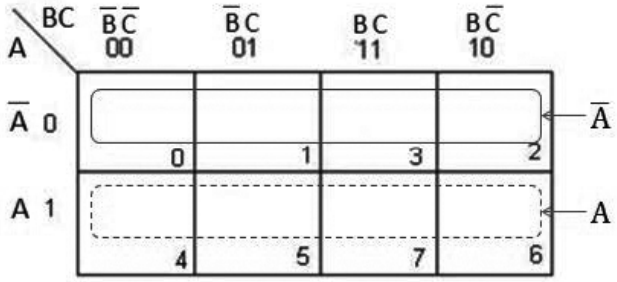


Grouping for four adjacent Ones (Quads):

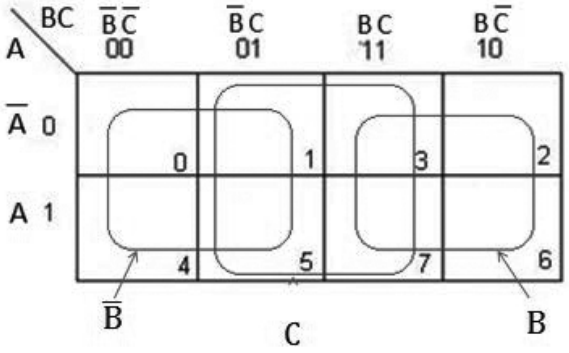
- Group the adjacent 1s in the given K-map and write the common variables eliminating uncommon variables. In reduction technique by forming pair two variables gets eliminated.

Possible Quads in 3-variable K-map:

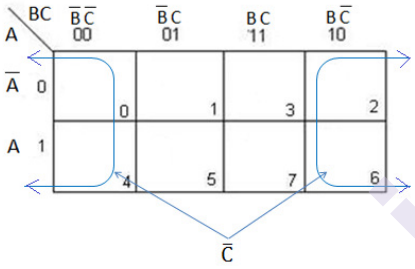
Adjacent Horizontal:



Adjacent squares:

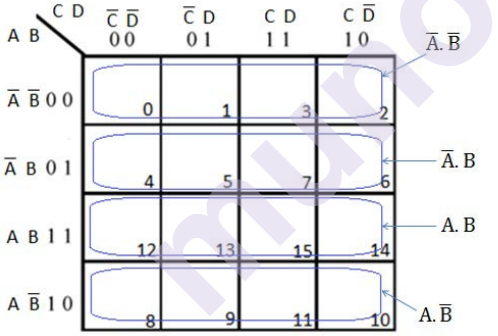


Fold-back:

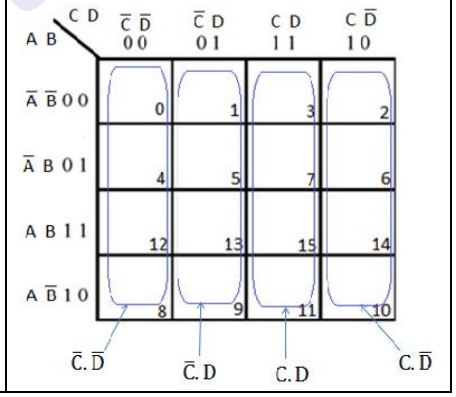


Possible Quads in 4-variable K-map:

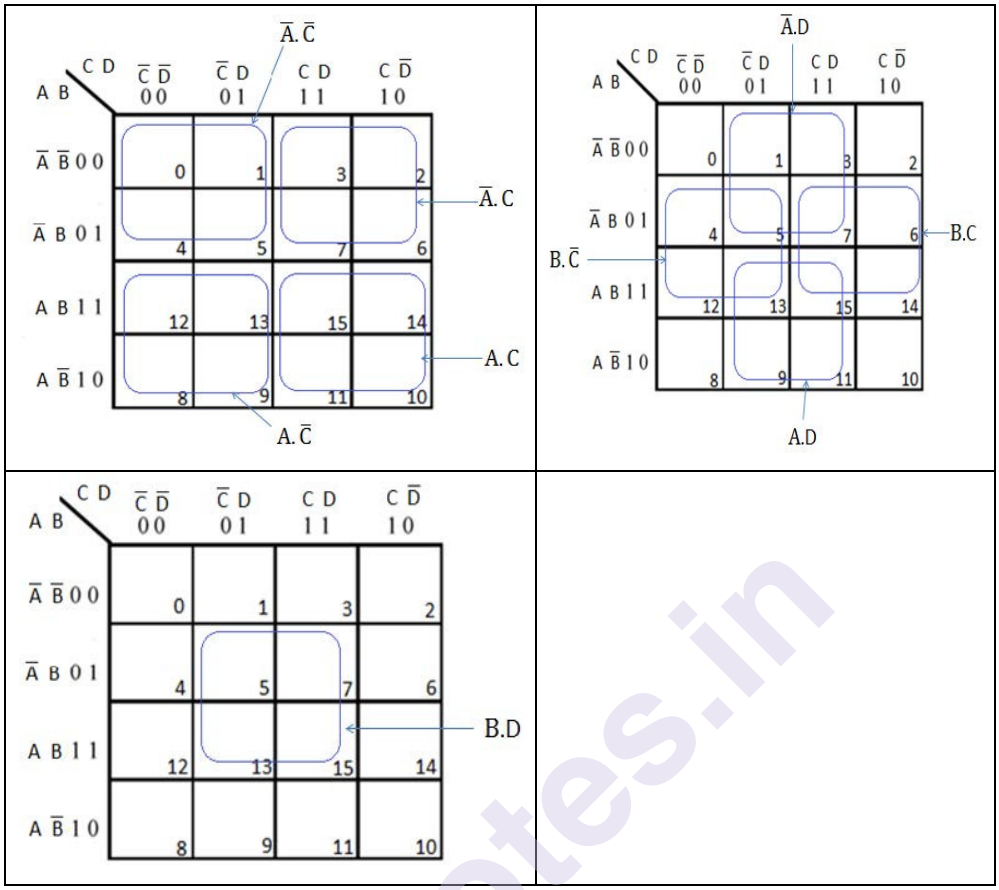
1. Adjacent Horizontal



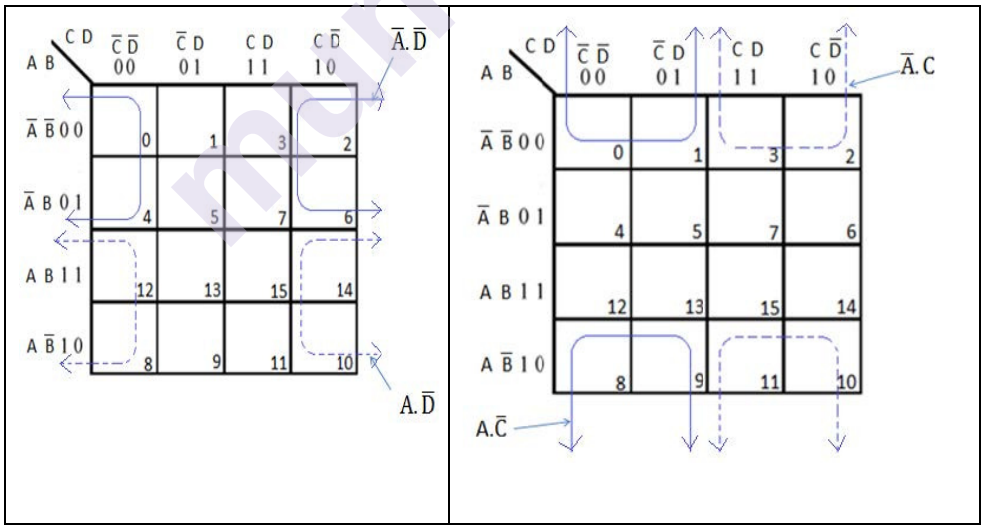
2. Adjacent Vertical

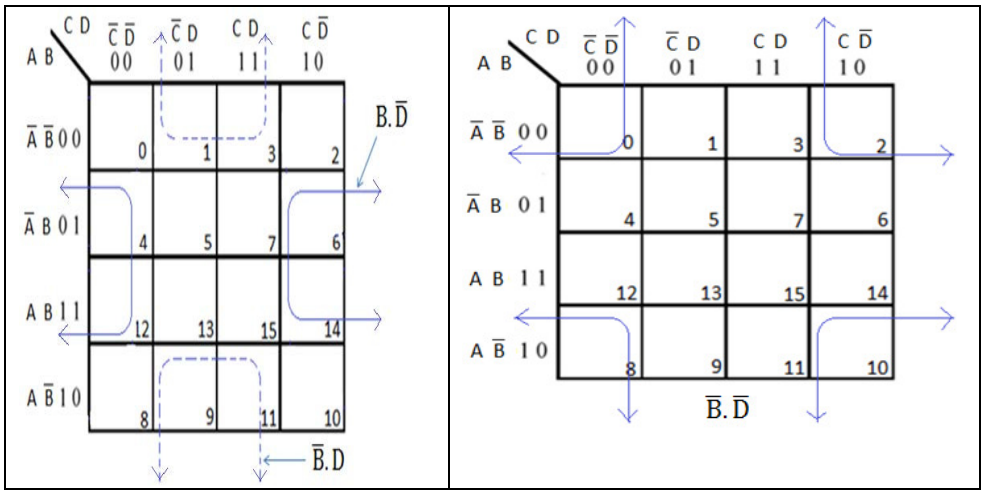


Adjacent squares:



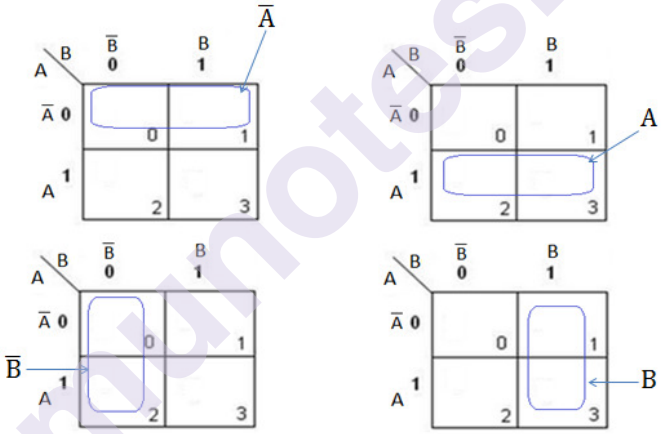
Fold-backs:





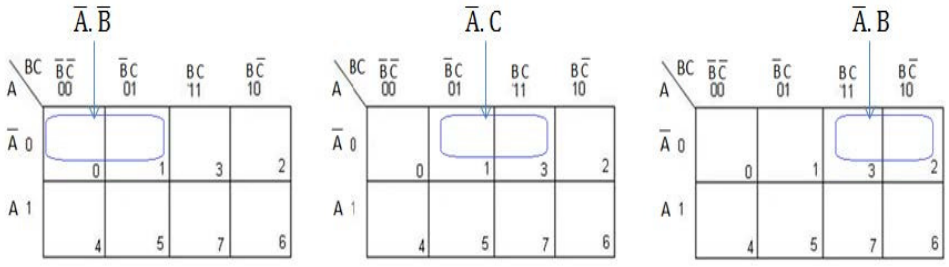
Grouping for two adjacent Ones (Pairs):

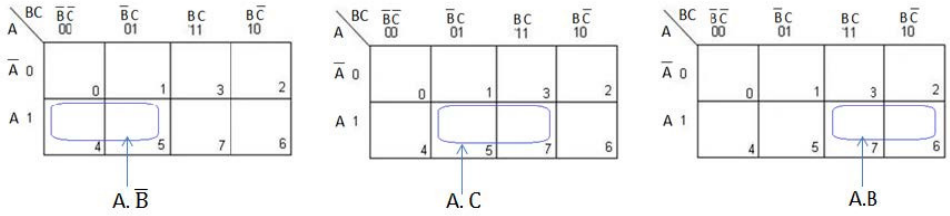
- Group the adjacent 1s in the given K-map and write the common variables eliminating uncommon variables. In reduction technique by forming pairs one variable gets eliminated.
- Possible pairs in 2-variable K-map:



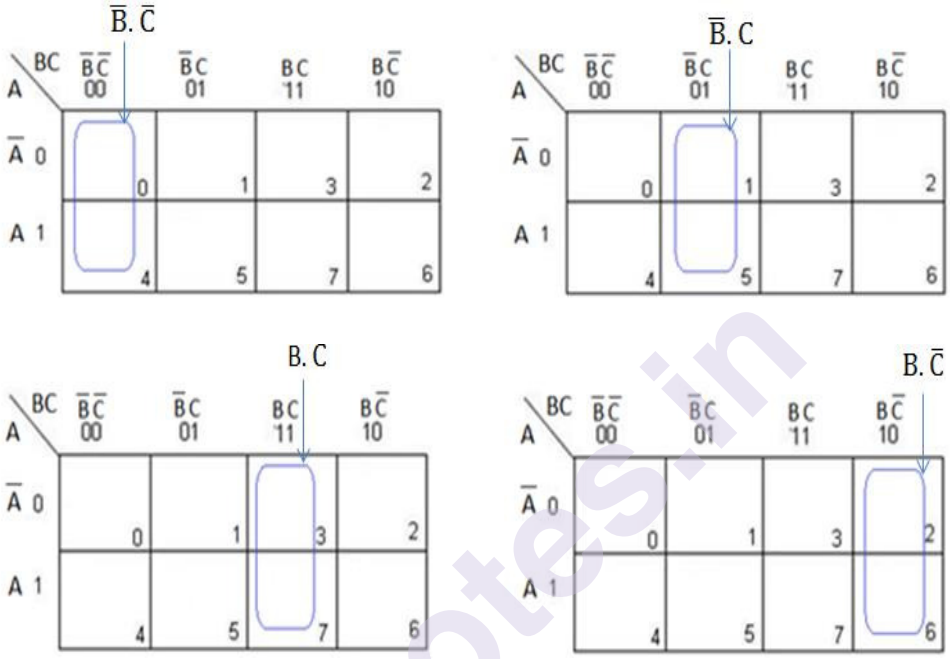
Possible pairs in 3-variable K-map:

Horizontal Adjacent:

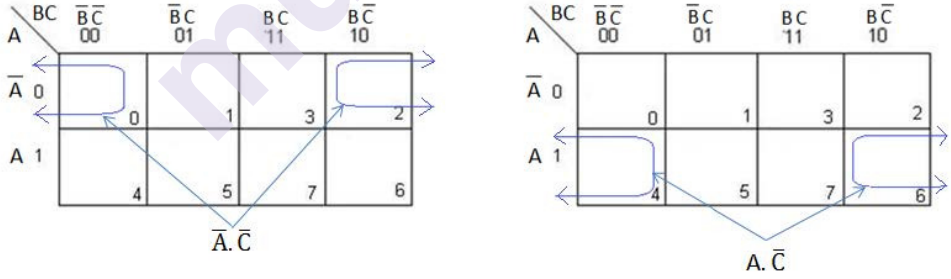




2. Vertical adjacent:

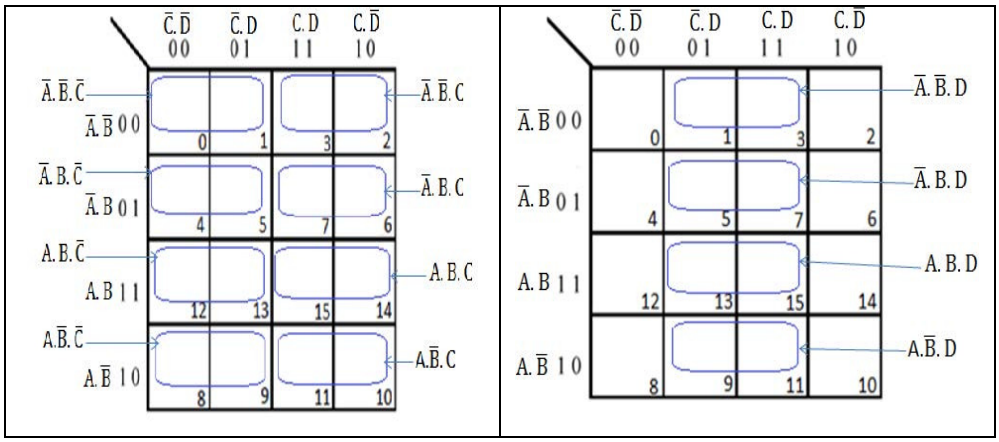


3. Fold-backs :

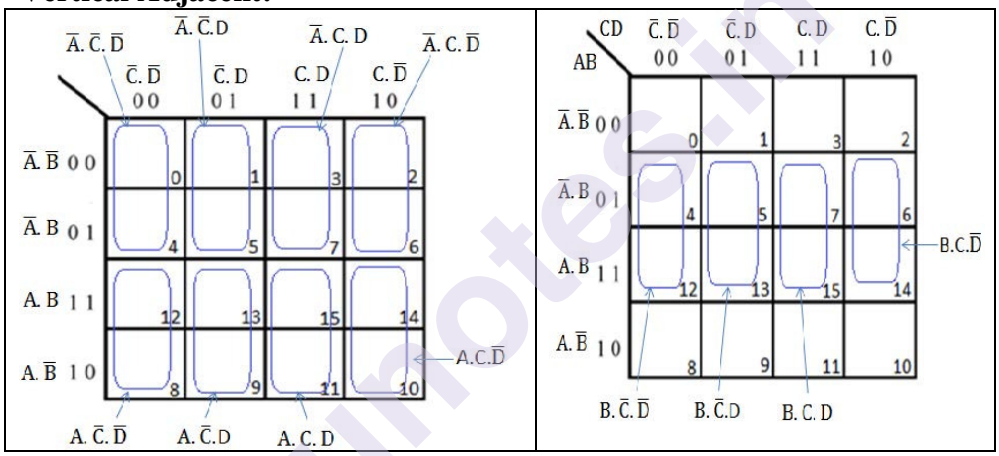


Possible pairs in 4-variable K-map:

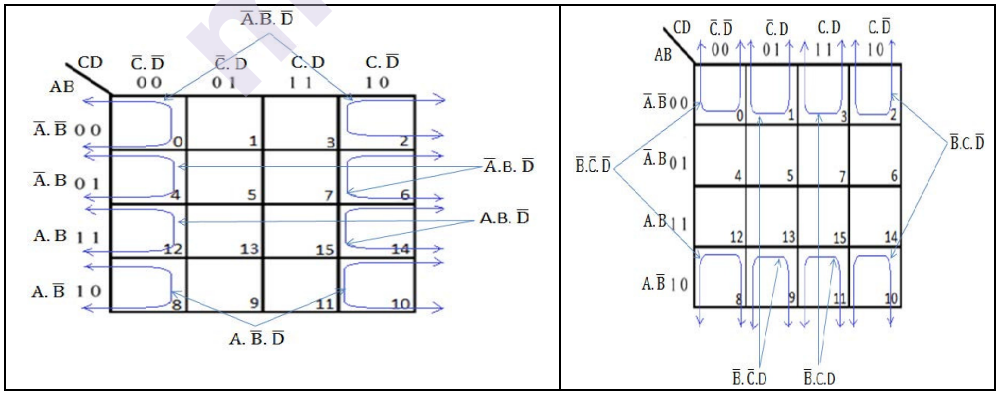
Horizontal Adjacent:



Vertical Adjacent:



Fold-backs:



Example 1:

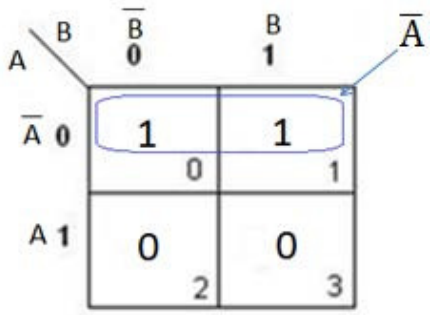
Solve:

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

Solution:

Hence $Y = m_0 + m_1$

We need 2-variable K-map



Here pair variable "B" is eliminated

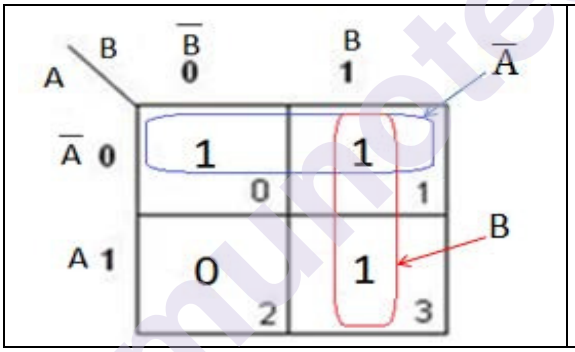
Hence, $Y = \overline{A}$

Example 2:

Solve: $Y = A.B+\overline{A}.\overline{B}+\overline{A}.B$

Solution:

Hence $Y = m_0 + m_1 + m_3$



(Variable B is eliminated)

(Variable A is eliminated)

Hence $Y = \overline{A}.B$

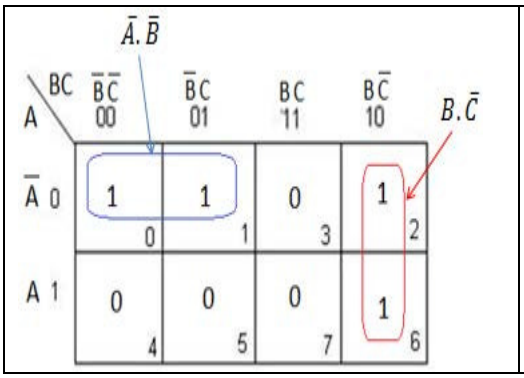
Example 3:

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

Solution :

$Y = m_0 + m_1 + m_2 + m_6$

Hence we need 3-variable (8 places) K-map.



(Variable A is eliminated)

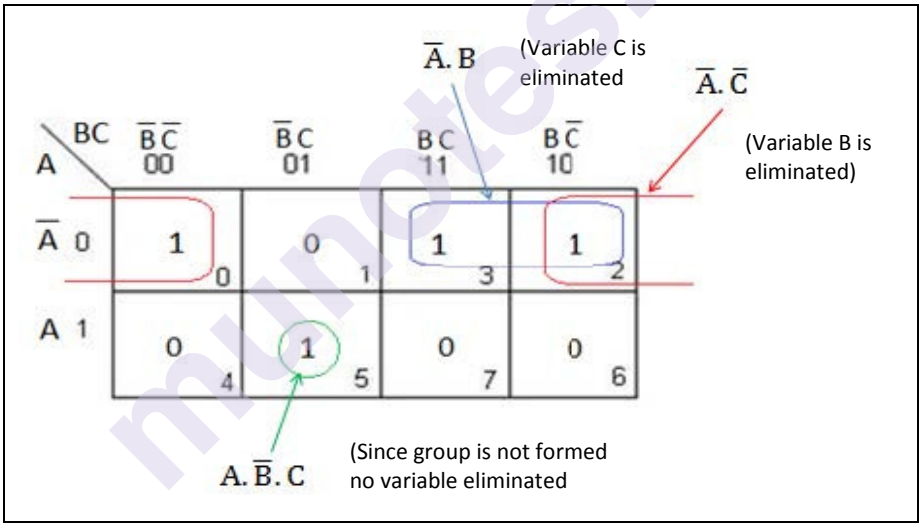
Hence $Y = \bar{A}.\bar{B}+B.\bar{C}$

Example 4:

Solve: $Y = \bar{A}.\bar{B}.\bar{C}+\bar{A}.B.\bar{C}+\bar{A}.B.C + A.\bar{B}.C$

Solution :

$Y = m_0 + m_2 + m_3 + m_5$



Hence $Y = \bar{A}.B+\bar{A}.\bar{C} + A.\bar{B}.C$

Example 5:

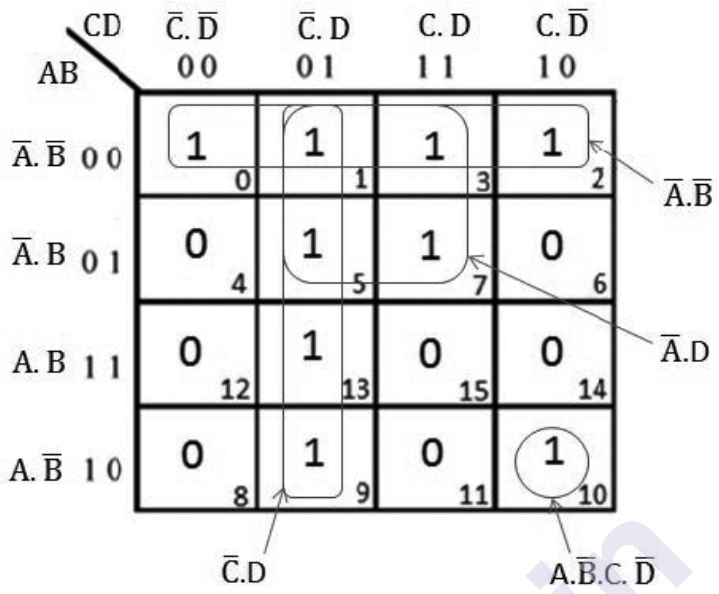
Solve:

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

Solution:

$Y = m_0 + m_1 + m_2 + m_3 + m_5 + m_7 + m_9 + m_{10} + m_{13}$

we need 4-variable (16 places) K-map.



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

4.2 MINIMIZATION OF LOGICAL FUNCTIONS NOT SPECIFIED IN MINTERMS/MAXTERMS

If the function is specified in one of the two standard forms, its K-map can be prepared and the function can be minimized. Now we consider the cases where the functions are not specified in standard forms. In such cases, the equations can be converted into standard forms using the techniques, the K-maps obtained and minimized. Alternately, we can directly prepare K-map using the following algorithm:

i. Enter ones for minterms and zeros for maxterms.

ii. Enter a pair of ones/zeros for each of the terms with one variable less than the total number of variables.

iii. Enter four adjacent ones/zeros for terms with two variables less than the total number of variables.

iv. Repeat for other terms in the similar way.

Once the K-map is prepared the minimization procedure is same as discussed earlier.

The following examples will help in understanding the above procedure:

Example 1 : Minimize the four variable logic function

$$f(A,B,C,D) = AB\bar{C}D+\bar{A}.B.C.D+\bar{A}.\bar{B}.\bar{C}+\bar{A}.\bar{B}.\bar{D}+\bar{A}.C+\bar{A}.B.C+\bar{B}$$

Solution :

The method for obtaining K-map is

i. Enter 1 in the cell with A = 1, B = 1, C = 0, D = 1 corresponding to the minterm $AB\overline{C}D$

ii. Enter 1 in the cell with A = 0, B = 1, C = 1, D = 1 corresponding to the minterm ABCD .

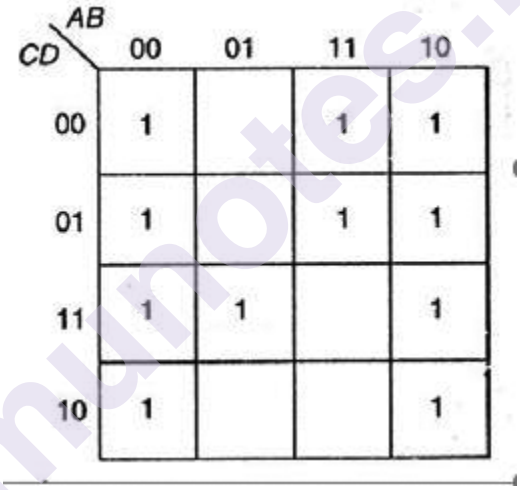
iii. Enter 1's in the two cells with A = 0, B =0

iv. Enter 1's in the two cells with A = 0, B = 0, D = 0 (one of these is already entered) corresponding to the term. $\overline{A}\overline{B}\overline{D}$

v. Enter 1's in the two cells with A = 1, B = 0, C = 1 corresponding to the term ABC

vi. Enter 1's in the four cells with A = 1, C = 0 (one of them is already entered) corresponding to the term AC

vii. Enter 1's in the eight cells with B = 0 (all of them except one have already been entered) corresponding to the term B.



4.3 QUINE-Mc CLUSKEY MINIMIZATION TECHNIQUE

Modem digital systems are designed using complex programmable logic devices (CPLDs), field-programmable gate arrays (FPGAs), and other very large scale integrated circuits that can be configured by the end user. These devices are highly complex and therefore, the techniques required for designing digital systems using these devices have to be computer driven rather than manual. A logic minimization technique which has the following characteristics is therefore, required:

1. It should have the capability of handling large number of variables.

2. It should not depend on the ability of a human user for recognising prime-implicants.
3. It should ensure minimized expression.
4. It should be suitable for computer solution.

The Quine-McCluskey minimization technique satisfies the above requirements and hence can be effectively used for the design of logic circuits. The K-map technique is not suitable for handling the design of complex digital systems because of the following disadvantages:

1. Minimization of logic functions involving more than six variables is unwieldy.
2. Recognition of prime-implicants that may form part of the simplified function relies on the ability of the human user making it difficult to be sure whether the best selection has been made.

The Quine-McCluskey method consists of two parts:

1. To find by an exhaustive search all the prime-implicants that may form part of the simplified function.
2. To identify essential prime-implicants obtained from part 1 and choose among the remaining prime-implicants those that give an expression with the least number of literals

The method can be best understood with the help of examples. This method is also known as Tabular method.

Example 1: Simplify the logic function $Y(A,B,C,D)=\sum m(0,1,3,7,8,9,11,15)$ the Quine-McCluskey minimization technique

Solution: $Y(A,B,C,D)=\sum m(0,1,3,7,8,9,11,15)$

Step 1: The logic function to be minimized here is in the minterm form, therefore, we go to step 2.

Step 2: Arrange all the minterms of the function in binary representation form in a Table according to the number of ones contained and form the groups containing no ones, one 1, two 1s, three 1s and so on. The groups are separated by horizontal lines.

Below Table shows the arrangement of groups, minterms, and the variables. Here group 0 contains no 1s {0}; group 1 contains minterms having a single 1 {1, 8}; group 2 contains minterms with two 1s {3, 9}; group 3 contains minterms with three 1s {7, 11}; and group 4 contains minterms with four 1s {15}.

Group	Minterm	Variables				Setup3
		A	B	C	D	√
0	0	0	0	0	0	√
1	1	0	0	0	1	√
	8	1	0	0	0	√
2	3	0	0	1	1	√
	9	1	0	0	1	√
3	7	0	1	1	1	√
	11	1	0	1	1	√
4	17	1	1	1	1	√

Step 3 : The Boolean algebraic theorem $A+\bar{A} = 1$ (Tneorem 1.7) is applied to pairs of minterms in which only one variable is different and all the other variables are same. This kind of relationship will be applicable only to the minterms belonging to adjacent groups of minterms. For this search, compare each minterm in group (n+1) with each minterm in group and identify the matched pairs. Put a check (√) on each matched pair as shown in Table 5.15.

The detailed procedure for comparison and matched pairs for Table 5.16 is given below and another Table 5.17 is prepared,

(i) The minterm 0 from group 0 is compared with the minterm 1 in the adjacent group 1. The three variables A, B, C are same in both with value 0 and the variable D is 0 in minterm 0 and 1 in minterm 1. Check (√) marks are placed on both the terms in Table 5.16 and a new term is generated as a result of the matching of these two terms which will contain A = 0, 8 = 0. C = 0 and a dash (-) mark is placed in D. Since the variable D differs and hence it gets eliminated and the resulting combination of these two terms is ABC.

(ii) Next the minterm 0 is compared with the minterm 8. The minterms match and their combination results in a term BCD. Check mark is placed on minterm 8 in Table 5.16, check mark on minterm 0 has already been placed. A - is placed under variable A.

(iii) Similarly, comparison of minterm 1 with 3 results in A = 0, B = 0, C = -, and D = 1 The minterm 3 is checked in Table 5.16.

(iv) Comparison of minterm 1 with minterm 9 yields A = -, B = 0, C = 0, and D = 1 and the minterm 9 is checked

(v) Now compare 8 with 3 and 9. The minterms 8 and 3 do not match. The comparison of minterms 8 and 9 results in A = 1, B = 0, C = 0, and D = -

(vi) Next compare the minterms 3 and 7, it results in A = 0, B = -, C = 1, and D = 1 and the minterm 7 is checked in Table 5.16.

(vii) Comparison of the minterms 3 and 11 results in A = -, B = 0, C = 1, and D = 1 and the minterm 11 is checked in Table 5.16

(viii) The minterms 9 and 7 do not match.

(ix) Comparison of the minterms 9 and 11 results in A = 1, B = 0, C = -, and D = 1

(x) Next compare the minterms 7 and 15, the result is A = -, B = 1, C = 1, and D = 1 and the minterm 15 is checked in Table 5.16

(xi) Comparison of the minterms 11 and 15 results in A = 1, S = -, C = 1, and D = 1

(xi) Table 5.17 lists the results of all the above matchings and all of the minterms in each group have been compared to those in the next higher group.

Table 5.17 : Combination of minterm groups of two

Group	Minterm	Variables				Setup3
		A	B	C	D	√
0	0,1	0	0	0	-	√
	0,8	-	0	0	0	√
1	1,3	0	0	-	1	√
	1,9	-	0	0	-	√
	8,9	1	0	0	1	√
2	3,7	0	-	1	1	√
	3,11	-	0	1	1	√
	9,11	1	0	-	1	√
3	7,15	-	1	1	1	√
	11,15	1	-	1	1	√

Step 4 : Next all the minterms in the adjacent groups in Table 5.17 are compared to see if groups of four can be made by matching. For this the dashes must be in the same bit position in the groups of two and only one variable must differ (0 in one group and 1 in the other). The matched pairs of minterms are checked in Table 5.17 and a new Table 5-18 is created.

In Table 5.18 we observe that in each group the two terms are same, therefore, only one is to be taken in each group.

Table 5.18 : Combination of minterm groups of four

Group	Minterm	Variables			
		A	B	C	D
0	0,1,8,9	-	0	0	-
	0,8,1,9	-	0	0	-
1	1,3,9,11	-	0	-	1
	1,9,3,11	-	0	-	1
2	3,7,11,15	-	-	1	1
	3,11,7,15	-	-	1	1

Step 5: Repeat the Process of Grouping of 8 miniterms. In this case both dashes must be in the same bit position and only one other variable must be different for matching. Since, there is no matching possible here, therefore, the process is complete. In general, this same process is repeated until no further combinations of minterm groups is possible.

Step 6: All nonchecked minterm groups in Tables 5.16, 5.17, and 5.18 are the prime-implicants of the function. The function can now be written as

$$Y(A,B,C,D) = BC+B\overline{D}+CD.....(5.52)$$

Step 7: Next a prime-implicant table is prepared listing each of the minterms contained in the original function, PI terms, and the decimal numbers of minterms that make up the PL Put cross (x) marks in the table in each row under the minterms contained in that PI. Table 5.19 is the PI table for the logic equation Eq. (5.21). Find the minterms that contain only one x in its column. These x's are encircled and (he corresponding PI terms are checked ( )

Table 5.19 : PI table

PL Terms	**Decimal No**	**Minterms**							
		0	1	3	7	8	9	11	15
BC	0,1,8,9	⊗	×			⊗	×		
BD	1,3,9,11		×	×			×	×	
BD	3,7,11,15			×	⊗			×	⊗

The minterms 0 and 8 are contained in only one PI $\overline{B}\overline{C}$ the minterms 7 and 15 are contained in only PI CD. Therefore, the prime-implicants BC and CD are essential prime-implicants. Now observe the other mfnterms and see whether these are contained in EPIs or not. Here, the minterms 1 and 9 are contained in $\overline{B}\overline{C}$ and 7 and 11 are contained in CD. Therefore, all the minterms of the original function are included in the two EPIs and the minimized expression will be

$$Y=(A,B,C,D)= \overline{B}\overline{C} + CD \text{ …..5.53}$$

Practice Questions:

1. Using Karnaugh's map simplify the following SO function and implement it with basic gates

   F(A,B,C,D)=(2,3,6,7,8,10,11,12)+d(14,15)

2. Realize the given Boolean expression using NOR gates only.

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

3. Obtain product of sum expression for the following function and implement it using NOR gates F(P,Q,R,S)=(1,3,5,6,7,12,13)

4. $F(A,B,C,D)=\sum m(0,1,2,5,6,7,12,13,15)$ Draw k-map and find minimized Boolean expressions.

5. What is meant by don't care conditions? Explain how they are used in simplifying an expression using a k-map. Use the following example- F(A,B,C,D)= Σm (1,4,8,12,13,15) d (3,14)

6. What are disadvantages of k-map? Explain the Q-M method. Discuss the terms 'prime impeccant', 'code word' and 'reduction table

*****

Unit 3

5

COMBINATIONAL LOGIC CIRCUITS

Unit Structure

5.0 INTRODUCTION COMBINATIONAL LOGIC CIRCUITS

Combinational Logic Circuits are circuits designed by using different types of logic gates. A logic gate is a fundamental building block of any electronic circuit. In other word Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs. Means these circuits do not make use of any memory or storage device.

The digital system consists of two types of circuits as -
a) Combinational circuits b) Sequential circuits 2

Combinational circuit consists of logic gates whose output at any time is determined from the present combination of inputs. The logic gate is the most basic building block of combinational logic circuit.

A combinational circuit consists of:

- Input variables
- Logic gates
- Output variables

The logic gates accept signals from inputs and output signals are generated according to the logic circuits working in it. Binary information from the given data transforms to preferred output data in this process. Both input and output are apparently the binary signals, i.e. both the input and output signals are of two probable states, logic '1' and logic '0'. Logical function performed by a combinational circuit is completely defined by a set of Boolean expressions.

The function implemented by combinational circuit is depend upon the Boolean expressions. Below figure shown the combinational circuit having 'n' inputs and 'm' outputs. The 'n' number of inputs shows that there are 2^n possible combinations of bits at the input.

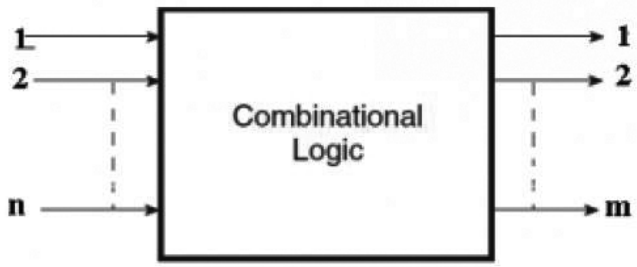


Fig. : Combinational circuit having 'n' inputs and 'm'outputs

Combinational logic circuits may be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classified as universal gates

Design Procedure:

At all combinational circuit can be designed by the following steps of design process- 3

1. Stated problem
2. Ascertain the input and output variables
3. The input and output variables are allocated letter symbols
4. Construction of a truth table to encounter input -output requirements
5. Writing Boolean expressions for numerous output variables in relations of input variables.
6. The basic Boolean expression is acquired by any method of minimization — algebraic, or tabulation method. orKarnaugh map method.
7. A logic diagram is understood from the simplified Boolean expression using logic gates.

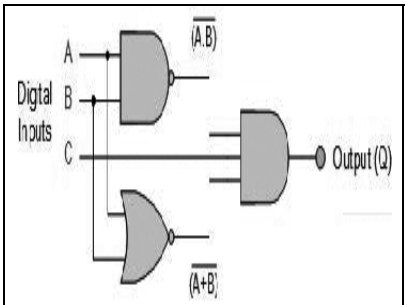
The logic gates are combined in such a way that the output state depends completely on the input states. Combinational logic circuits have no memory, timing or feedback loops, there operation is prompt. A combinational logic circuit performs an operation assigned logically by a Boolean expression or truth table.

The three main methods of specifying the function of a combinational logic circuit as-

1. **Boolean Algebra:** This forms the algebraic expression viewing the operation of the logic circuit for each input variable whicheverTrue or False that results in a logic ''1'' output.

2. **Truth Table:** A truth table expresses the function of a logic gate by providing a brief list that shows all the output states in tabular form for each possible combination of input variable that the gate could meet.

3. **Logic Diagram :** This is a graphical representation of a logic circuit that displays the wiring and connections of each individual logic gate, signified by a specific graphical symbol that implements the logic circuit. These logic circuit representations are shown as-



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

C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Fig. : Combinational Circuit Diagram, Boolean Expression & truth Table

As combinational logic circuits are made up from discrete logic gates only, they can also be considered as decision making circuits and combinational logic is nearby combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. General combinational circuits made up from individual logic gates that carry out a anticipated application comprise Full and Half Adders Multiplexers, De-multiplexers, Encoders, Decoders etc.

Classification of Combinational Logic:

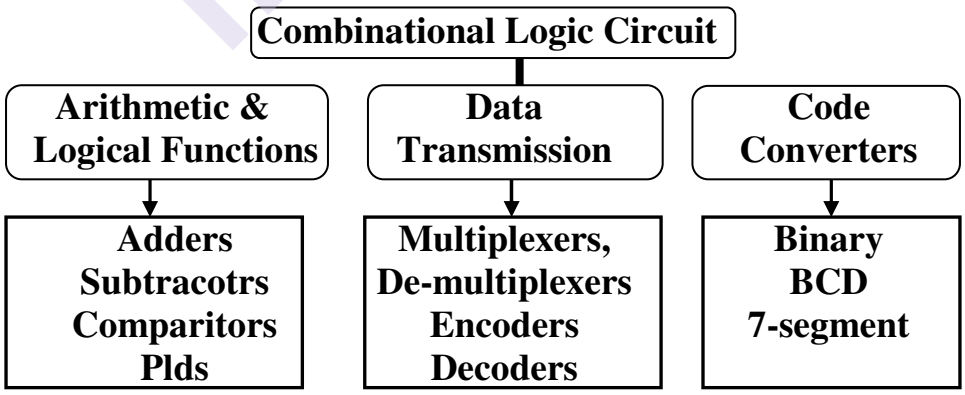


Fig. Classification of Combinational Logic

One of the greatest uses of combinational logic is in Multiplexer and De-multiplexer type circuits. Here, multiple inputs or outputs are connected to a common signal stripe and logic gates are used to decode an address to select a single data input or output switch.

5.1 MULTI INPUT COMBINATIONAL CIRCUIT

In processor designs, transistors are protected down for specific functions. To overcome the restriction of fixed structures of static architecture in the next generation of computer is an important issue to the real-world applications. A reconfigurable technique with dynamic architecture has made it possible to break through the stable limitations of the current computer systems. In dynamic architecture, systems can flexibly change their hardware configurations during the course of computation according to the demands of various functions. Multi input gates can be made by constructing gates of the same type with fewer inputs. The figure below shown how a three input AND gate can be made out of two input AND gates. The same logical standard applies - the output goes ‘‘low’’ (0) if any of the inputs are made “high” (1).

Combinational Logic Circuits are made up from basic logic NAND, NOR or NOT gates that are connected together to produce more complex switching circuits. These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit-Decoder, which converts the binary code data present at its input into a number of diverse output lines, one at a time producing an equivalent decimal code at its output.

Fig. Multi Input Combinational Circuit

The number of possible input states is equal to two to the power of the number of inputs: 6

$$\text{Number of possible input states} = 2^n$$

Where,

n = Number or inputs

This increase in the number of possible input states obviously allows for more complex gate performance. Instead of simply inverting a single “high” or “low” logic level, the output of the gate will be determined by whatsoever *combination* of 1‘s and 0‘s is present at the input stations.

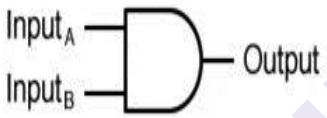
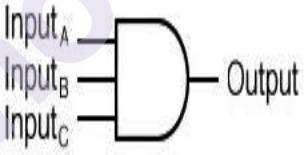
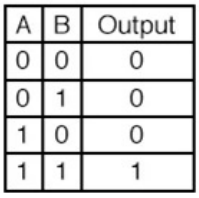
Since many combinations are possible with just a limited input stations, there are many different types of multiple-input gates, unlike single-input gates which can simply be inverters. Each basic gate type will be shown as its standard symbol, truth table, andoperation.

Types of different Multi Input Combinational Circuit

a) AND Gate
b) NAND Gate
c) Negative AND Gate
d) OR Gate
e) NOR Gate
f) Negative OR Gate
g) Exclusive OR Gate
h) Encoder
i) Multiplexer

a) AND Gate:

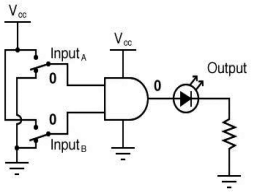
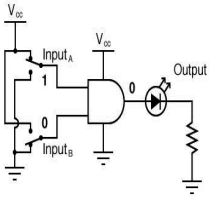
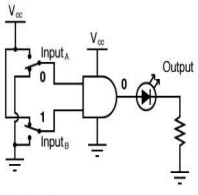
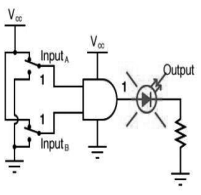
The simple multiple-input gates to recognize is the AND gate, the output of this gate will be ‘‘high’’ (1) if and only if *all* inputs are “high” (1). If any input(s) is ‘‘low’’ (0), the output is certain to be in a “low” state.

Two input AND gate	Three input AND gate	Truth Table
		

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate Circuit Operation:

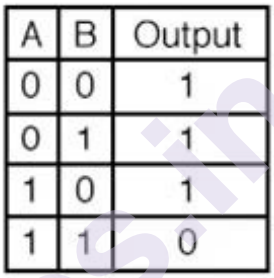
The truth table means in practical terms is shown in the following sequence as, with the 2-input AND gate exposed to all possibilities of input logic levels. An LED provides visual signal of the output logic level.

Circuit Diagram			
			
Input$_A$ = **0** Input$_B$ = **0** Output = **0** *(no light)*	Input$_A$ = **1** Input$_B$ = **0** Output = **0** *(no light)*	Input$_A$ = **0** Input$_B$ = **1** Output = **0** *(no light)*	Input$_A$ = **1** Input$_B$ = **1** Output = **1** *(light!)*

It is only with all inputs raised to ‘‘high’’ logic levels that the AND gate’s output goes ‘‘high,’’ thus stimulating the LED for only one out of the four input combination states.

b) NAND Gate:

A distinction on the idea of the AND gate is called the NAND gate. The term ‘‘NAND’’ is a verbal contraction of the words NOT and AND. Fundamentally, a NAND gate performs the same as an AND gate with a NOT gate connected to the output terminal. To represent this output signal inversion, the NAND gate symbol has a bubble on the output line. The truth table for a NAND gate is as one might expect, precisely opposite as that of an AND gate

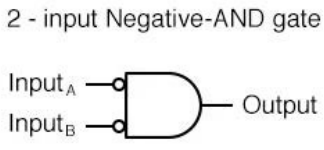
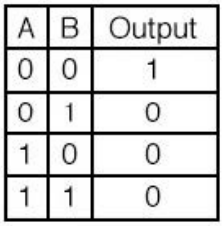
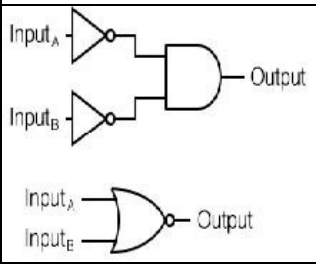
NAND gate	Truth Table
	

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

As with AND gates, NAND gates are made with more than two inputs. In such cases, the same over-all standard relates: the output will be “low” (0) if and only if all inputs are ‘‘high’’ (1). If any input is ‘‘low’’ (0), the output will go “high” (1).

c) Negative AND Gate:

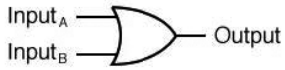
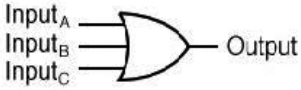
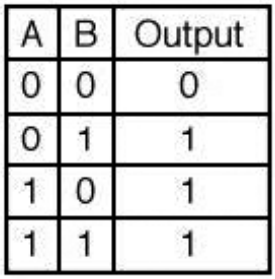
A Negative-AND gate purposes the same as an AND gate with all its inputs inverted (connected through NOT gates). In trust with standard gate symbol convention, these inverted inputs are showed by bubbles. Opposing to most publics‘ first nature, the logical conduct of a Negative-AND gate is not the same as a NAND gate. Its truth table, essentially, is matching to a NOR gate:

Negative AND gate	Truth Table	Circuit Diagram
		

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

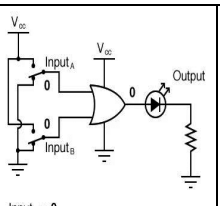
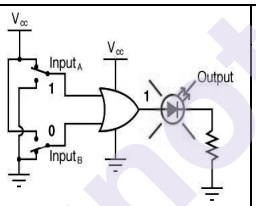
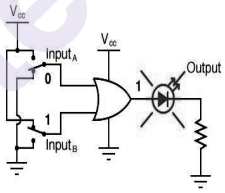
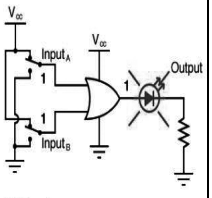
d) OR Gate:

Next gate to study is the OR gate, so-called because the output of this gate will be ‘‘high’’ (1) if any of the inputs are ‘‘high’’ (1). The output of an OR gate goes ‘‘low’’ (0) if and only if all inputs are ‘‘low’’ (0).

Two input	Three Input	Truth Table
		

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

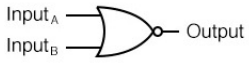
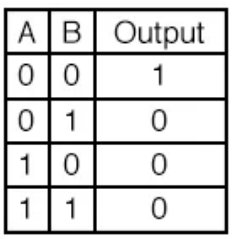
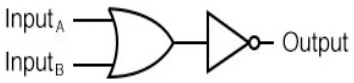
A two-input OR gate’s truth table appearances like the following sequence of drawings proves the OR gate’s function, with the 2-inputs undergoing all possible logic levels. An LED delivers visual indication of the gate’s output logic level

Circuit Diagram			
			

A condition of any input being outstretched to a ‘‘high’’ logic level makes the OR gate’s output go ‘‘high,’’ thus stimulating the LED for three out of the four input combination states.

e) NOR gate:

The NOR gate is an OR gate with its output inverted, just like a NAND gate is an AND gate with an inverted output

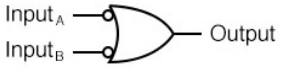
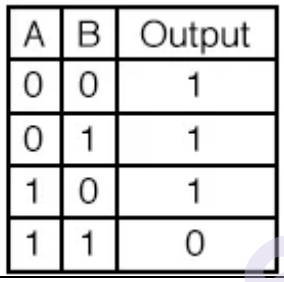
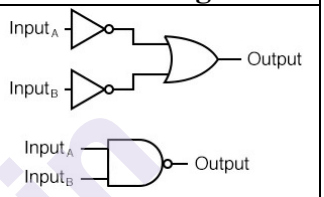
NOR Gate	Truth Table	Circuit Diagram
2 - input NOR gate 		

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

NOR gates, like all the other multiple-input gates realized thus outlying, can be synthetic with more than two inputs. Still, the same logical standard applies. The output goes ‘‘low’’ (0) if any of the inputs are made ‘‘high’’ (1). The output is ‘‘high’’ (1) only when all inputs are ‘‘low’’ (0)

f) Negative-OR Gate:

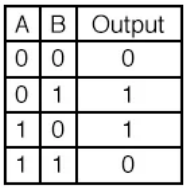
A Negative-OR gate functions the same as an OR gate with all its inputs inverted. In possession with standard gate symbol convention, these inverted inputs are showed by bubbles. The performance and truth table of a Negative-OR gate is the same as for a NAND gate:

NOR Gate	Truth Table	Circuit Diagram
		

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

g) Exclusive OR Gate:

This gate is direct variations on three basic functions- AND, OR, and NOT. The Exclusive-OR gate, however, is approximately fairly different. Exclusive-OR gates output a ‘‘high’’ (1) logic level if the inputs are at different logic levels, either 0 and 1 11 inputs are at the same logic levels. or 1 and 0. Contrariwise, the output a ‘‘low’’ (0) logic level if the The Exclusive-OR (XOR) gate has both a symbol and a truth table pattern that is distinctive:

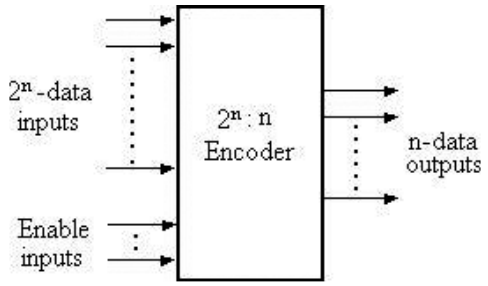
Exclusive OR Gate	Truth table
	

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

There are equivalent circuits for an Exclusive-OR gate made up of AND, OR, and NOT gates, impartial as there were for NAND, NOR, and the negative-input gates. A somewhat direct method to simulating an Exclusive-OR gate is to start with a regular OR gate, then add additional gates to avoid the output from going ‘‘HIGH’’ (1) when both inputs are ‘‘HIGH’’ (1):

h) Encoder:

An encoder is a digital circuit that achieves the inverse operation of a decoder. Therefore, the opposite of the decoding process is called encoding. An encoder is a combinational circuit that converts binary information from 2^n input lines to a maximum of 'n' exclusive output lines.



It has 2^n input lines, only one ''1'' is active at any time and 'n' output lines. It encodes one active inputs to a coded binary output with 'n' bits. In an encoder, the number of outputs is less than the number of inputs.

i) Octal-to-Binary Encoder:

It has eight inputs and the three outputs that produce the equivalent binary number. It is assumed that only one input has a value of '1' at any given time

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

Table: Input / Output of Octal-to-'Binary Encoder'

The encoder can be implemented with OR gates whose inputs are determined straight from the truth table. Output z is equal to 1, when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7 and the output is 1 for digits 4, 5, 6 or 7. These conditions can be expressed by the following output Boolean functions:

$z = D_1 + D_3 + D_5 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$x = D_4 + D_5 + D_6 + D_7$

The encoder can be applied with three OR gates. The encoder defined in the below table, has the restriction that only one input can be active at any given time. If two inputs are active concurrently, the output produces an approximate combination.

For eg. if D_3 and D_6 are 1 concurrently, the output of the encoder may be 111. This does not represent either D_6 or D_3. To resolve this problem, encoder circuits must establish an input priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers and if D_3 and D_6 are 1 at the same time, the output will be 110 because D_6 has higher priority than D_3.

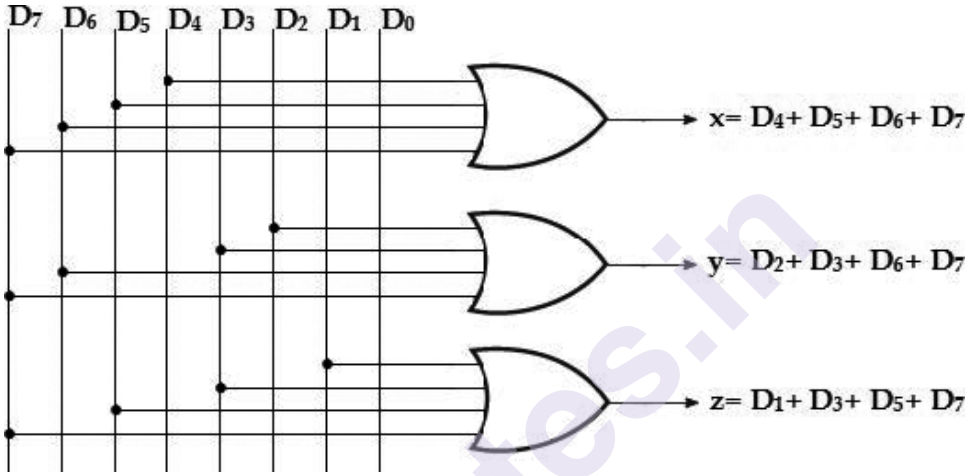


Fig. Circuit diagram of Octal-to-Binary Encoder

Another issue in the octal-to-binary encoder is that an output with all 0's is produced when all the inputs are 0; this output is same as when D_0 is equal to 1. The inconsistency can be resolved by providing one more output to indicate that at least one input is equal to 1.

i) Multiplexer: (Data Selector):

A multiplexer or MUX, is a combinational circuit with more than one input line, one output line and more than one selection line. A multiplexer selects binary information present from one of many input lines, depending upon the logic status of the selection inputs, and directions it to the output line. Generally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. The multiplexer is labelled as MUX in block diagrams. A multiplexer is also called a data selector, since it selects one of many inputs and leads the binary information to the output line.

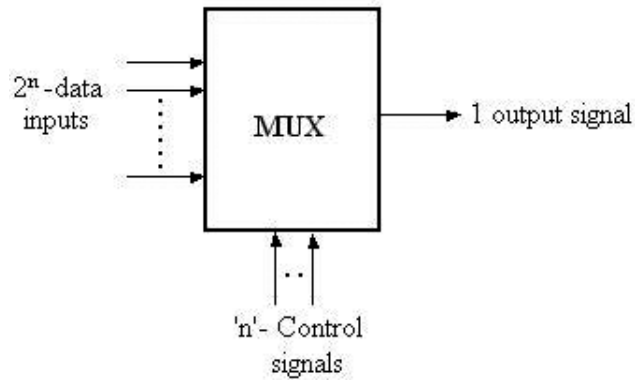


Fig. multiplexer

j) 2-to-1- line Multiplexer:

This circuit has two data input lines, one output line and one selection line. When S= 0, the upper AND gate is enabled and I_0 has a path to the output.

When S=1, the lower AND gate is enabled and I_1 has a path to the output

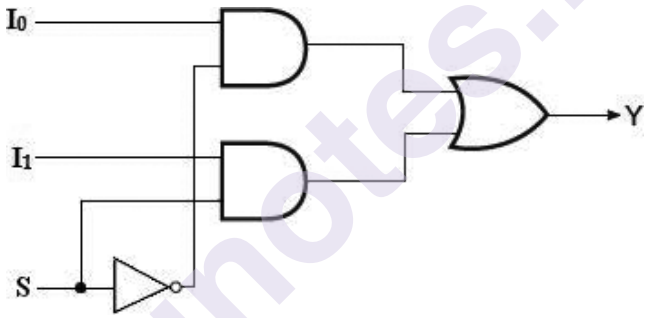


Fig.Circuit diagram 2-to-1- line Multiplexer

The multiplexer acts like an electronic switch that selects one of the two sources.

S	Y
0	I_0
1	I_1

Table: Truth table

ii) 4-to-1-line Multiplexer:

A 4-to-1-line multiplexer has four (2n) input lines, two (n) select lines and one output line. It is the multiplexer comprising of four input channels and information of one of the station can be selected and transmitted to an output line according to the select inputs combinations. Selection of one of the four input station is possible by two selection inputs.

Each of the four inputs I_0 through I_3, is applied to one input of AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.

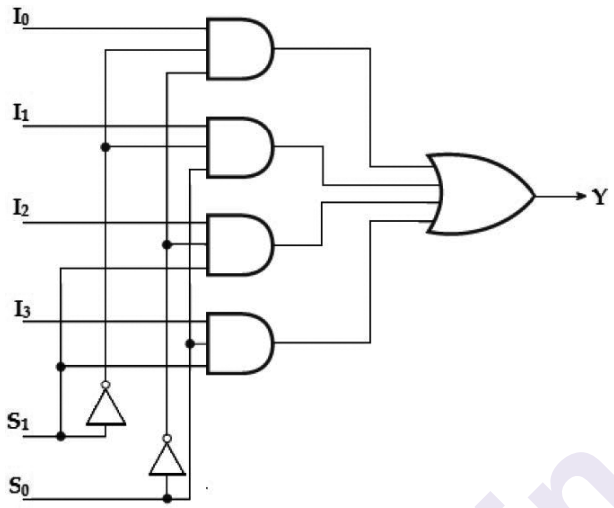


Fig. Circuit diagram of 4-to-1-line Multiplexer

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table: Truth table 4-to-1-line Multiplexer

To establish the circuit operation, consider the case when $S_1S_0 = 1_0$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I_2, providing a path from the selected input to the output.

The data output is equal to I0 only if $S_1 = 0$ and $S_0 = 0$; $Y = I_0S_1'S_0'$.
The data output is equal to I_1 only if $S_1 = 0$ and $S_0 = 1$; $Y = I_1S_1'S_0'$.
The data output is equal to I_2 only if $S_1 = 1$ and $S_0 = 0$; $Y = I_2S_1S_0$.
The data output is equal to I_3 only if $S_1 = 1$ and $S_0 = 1$; $Y = I_3S_1S_0$.

When these terms are ORed, the total expression for the data output is,
$Y = I_0S_1'S_0' + I_1S_1'S_0 + I_2S_1S_0' + I_3S_1S_0$.

In decoder, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the

outputs are disabled, and when it is in the active state, the circuit functions as multiplexer.

5.2 MULTI OUTPUT COMBINATIONAL CIRCUIT

A combinational circuit has output values that depend only on the current input values irrespective of presence or absence of responses.Circuits that implement these functions may be combined into expensive single circuit with multiple outputs by sharing some gates desirable in the implementation of the single functions. Boolean expressions are used to output a Boolean function of number of variables. There are circuits which have multiple outputs and multiple inputs. Conventional combinational circuits are normally acyclic but these circuits can have feedbacks (cycles) which will give more minimized expressions as compared to usual combinational circuits. Thoughtful integration of such cycles or feedbacks in usual combinational circuits eventually results in reduction in number of literals in the expression of the combinational circuits. The reduction in literal counts decreases the number of gates required to implement the expressions of the combinational circuits. Hence, the decrease in number of gates leads to reduction in transistor counts. A cyclic combinational circuit is defined as the circuit whose output depends on present inputs only, but at the same time contains one or more feedbacks (cycles)

Types of multiple-output circuits:

i) Decoder	a) 2 to 4 decoder	b) 3- to-8 Line Decoder
ii) Demultiplexer	a) 1 to 4 Demultiplexer	b) 3 to 8 Demultiplxer

i) Decoder:

A decoder is a combinational circuit that converts binary information from 'n' input lines to a maximum of ''2^n'' unique output lines. The general structure of decoder circuit is 17

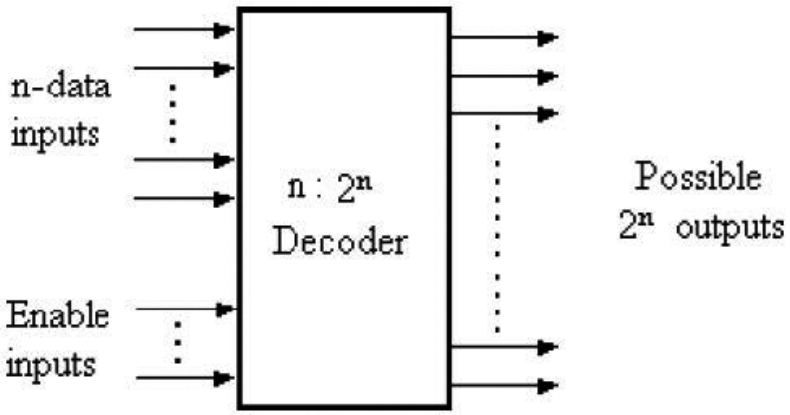


Fig. Decoder block diagram

The encoded information is presented as 'n' inputs producing 2n possible outputs. The 2^n output values are from 0 through 2^{n-1}. A decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled

a) Binary Decoder (2 to 4 decoder):

A binary decoder has 'n' bit binary input and a one activated output out of 2n outputs. A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n-bit input value.

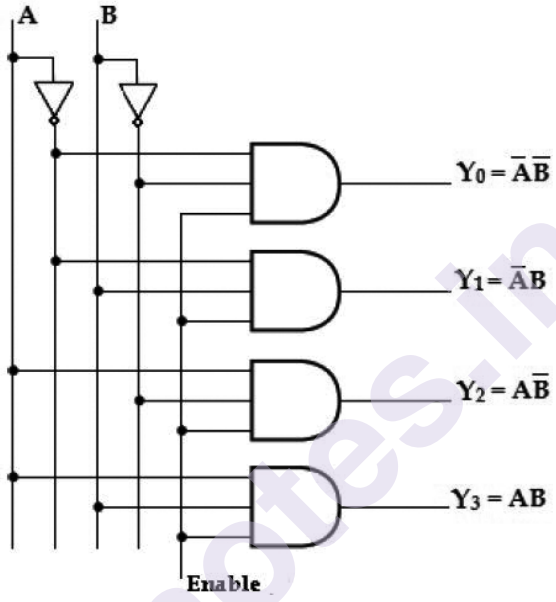


Fig. 2 to 4 decoder circuit diagram

Here the 2 inputs are decoded into 4 outputs, each output signifying one of the minterms of the two input variables.

Inputs			Outputs			
Enable	A	B	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Fig.Truth table 2 to 4 decoder circuit

As shown in the truth table, if enable input is 1 (EN= 1) only one of the outputs ($Y_0 - Y_3$), is active for a given input.

The output Y_0 is active, i.e., Y_0= 1 when inputs A= B= 0, Y_1 is active when inputs, A= 0 and B= 1,

b) 3- to-8 Line Decoder:

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y_0-Y_7). Based on the 3 inputs one of the eight outputs is selected

The three inputs are decoded into eight outputs, each output signifying one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion. The input variables may represent a binary number and the outputs will signify the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 signifies the minterm equivalent of the binary number currently accessible in the input lines.

Inputs			Outputs							
A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Fig.Truth table 3- to-8 Line Decoder

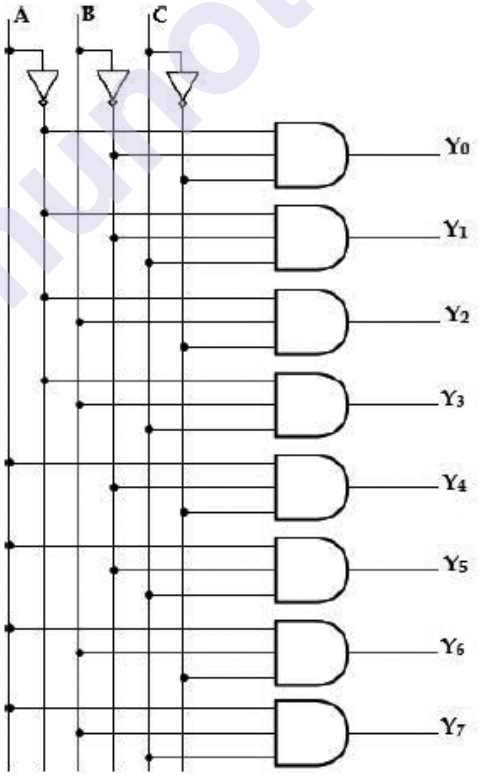


Fig. 3- to-8 Line Decoder

ii) Demultiplexer:

Demultiplex means one into many. Demultiplexing is the process of taking information from one input and transmitting the same over one of several outputs. A demultiplexer is a combinational logic circuit that receives information on a single input and transmits the same information over one of several (2^n) output lines

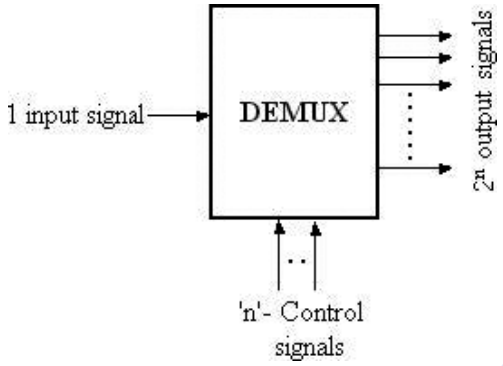


Fig. Demultiplexer circuit

The block diagram of a demultiplexer which is opposite to a multiplexer in its operation is shown above. The circuit has one input signal, 'n' select signals and 2^n output signals. The select inputs determine to which output the data input will be connected. As the serial data is changed to parallel data, i.e., the input produced to appear on one of the n output lines, the demultiplexer is also called a *—data distributer.*

i) 1-to-4 Demultiplexer:

A 1-to-4 demultiplexer has a single input, Din, four outputs (Y_0 to Y_3) and two select inputs (S_1 and S_0).

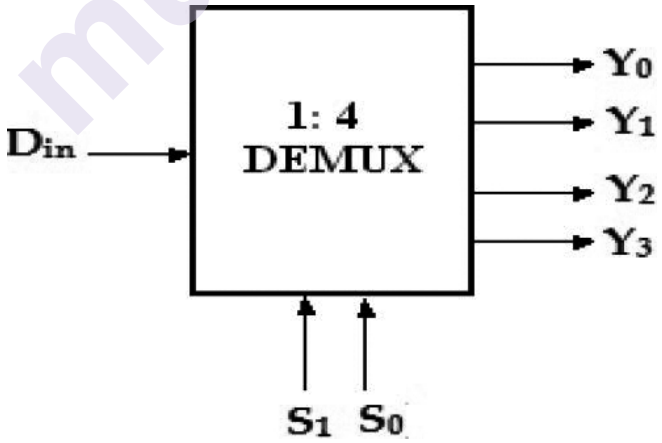


Fig. 1-to-4 Demultiplexer

The input variable Din has a path to all four outputs, but the input information is directed to only one of the output lines. The truth table of the 1-to-4 demultiplexer is shown below.

Enable	S_1	S_0	D_{in}	Y_0	Y_1	Y_2	Y_3
0	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Table: Input /Output of 1-to-4 Demultiplexer

From the truth table, it is clear that the data input, Din is connected to the output Y_0, when S_1= 0 and S_0= 0 and the data input is connected to output Y_1 when S_1= 0 and S_0= 1. Similarly, the data input is connected to output Y_2 and Y_3 when S_1= 1 and S_0= 0 and when S_1= 1 and S_0= 1, respectively. From the truth table, the expression for outputs can be written as follows,

$Y_0 = S_1 ' S_0 ' D_{in}$

$Y_1 = S_1 ' S_0 D_{in}$

$Y_2 = S_1 S_0 ' D_{in}$

$Y_3 = S_1 S_0 D_{in}$

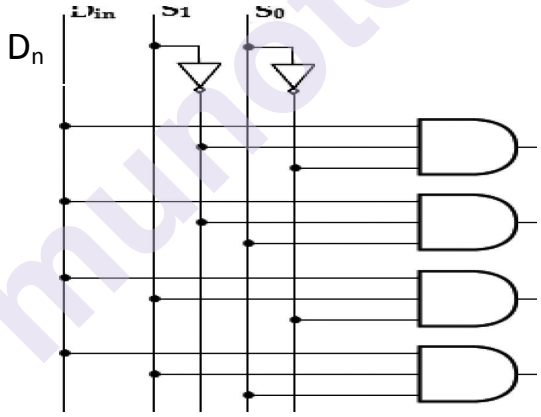


Fig. Circuit diagram 1-to-4 Demultiplexer

Now, using the above expressions, a 1-to-4 demultiplexer can be applied using four 3-input AND gates and two NOT gates. Here, the input data line Din, is connected to all the AND gates. The two select lines S_1, S_0 enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.

b)1-to-8 Demultiplexer :

A 1-to-8 demultiplexer has a single input, Din, eight outputs (Y_0 to Y_7) and three select inputs (S_2, S_1 and S_0). It distributes one input line to

eight output lines based on the select inputs. The truth table of 1-to-8 demultiplexer is shown below.

D_{in}	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table: Input/output of 1-to-8 Demultiplexer

From the above truth table, it is clear that the data input is connected with one of the eight outputs based on the select inputs. Now from this truth table, the expression for eight outputs can be written as follows

$Y_0 = S_2'S1'S0'$ D_{in} $Y_4 = S_2 S_1'S_0' D_{in}$

$Y_1 = S_2'S1'S0$ D_{in} $Y_5 = S_2 S_1'S_0' D_{in}$

$Y_2 = S_2'S1S0'$ D_{in} $Y_6 = S_2 S_1 S_0' D_{in}$

$Y_3 = S_2'S1S0$ D_{in} $Y_7 = S_2 S_1 S_0' D_{in}$

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can be drawn as shown below. Here, the single data line, Din is connected to all the eight AND gates, but only one of the eight AND gates will be enabled by the select input lines. For example, if $S_2S_1S_0$= 000, then only AND gate-0 will be enabled and thereby the data input, Din will appear at Y_0. Similarly, the different combinations of the select inputs, the input Din will appear at the respective output.

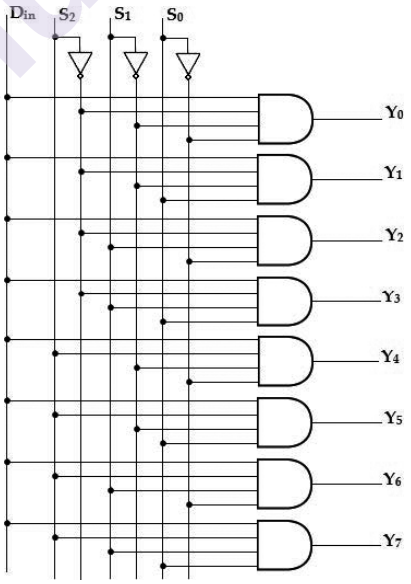


Fig.Circuit diagram 1-to-8 Demultiplexer

5.3 CODE CONVERTERS DESIGN AND IMPLEMENTATION

Introduction:

The converters, which convert one code to another code are called as code converters. These code converters basically consist of Logic gates. The availability of a large variety of codes for the same distinct elements of information outcomes in the use of different codes by different digital systems. It is necessary to use the output of one system as the input to the other. Thus a conversion circuit must be introduced between the two systems if each uses different codes for the same information. A code converter is a circuit that makes the two systems well-matched even though each uses the different codes. Code converters are used for protecting private information from detectives. They are also used to enhance data portability and tractability. Code converters have found applications in algorithm generation and communication. Some of the major codes are as follows:

Binary Code: A symbolic representation of data/information is called code. The base or radix of the binary number is 2. Hence, it has two independent symbols. The symbols used are 0 and 1. A binary digit is called a bit. A binary number consists of sequence of bits, each of which is either a 0 or 1. Each bit carries a weight based on its position comparative to the binary point. The weight of each bit position is one power of 2 greater than the weight of the position to its immediate right.

Code Converters Design and Implementation done with the help of following way as below-

a) Binary code to Gray code converter
b) Parity Bit Generator
    i) Even Parity Generator
    ii) Odd Party Generator
c) Parity Checker
    i) Even Parity Checker
    ii) Odd Party Checker

a) Binary code to Gray code converter:

Let us implement a converter, which converts a 4-bit binary code WXYZ into its equivalent Gray code ABCD.

The following table shows the Truth table of a 4-bit binary code to Gray code converter.

Binary code WXYZ	WXYZ Gray code ABCD
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Fig. -bit binary code to Gray code converter

Boolean Expression:
From Truth table, we can write the Boolean functions for each output bit of Gray code as below

A = Σm(8,9,10,11,12,13,14,15)

B = Σm(4,5,6,7,8,9,10,11)

C = Σm(2,3,4,5,10,11,12,13)

D = Σm(1,2,5,6,9,10,13,14)

Let us simplify the above functions using 4 variable K-Maps.

K-Map:

Let us simplify the above functions using 4 variable K-Maps.

The following figure shows the 4 variable K-Map for simplifying Boolean function, A.

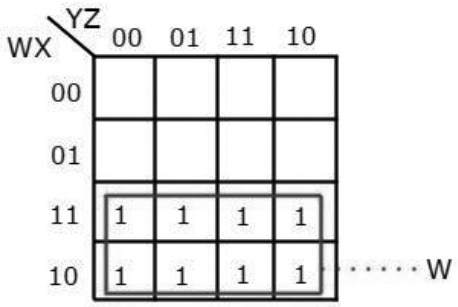


Fig. 4 variable K-Maps

By grouping 8 adjacent ones, we got A=WA=W.
The following figure shows the 4 variable K-Map for simplifying Boolean function, B.

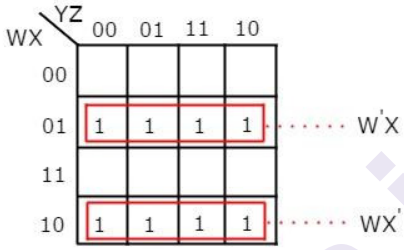


Fig. Simplification of 4 variable –K-Map

There are two groups of 4 adjacent ones. After grouping, we will get B as

B=W′X+WX′=W⊕XB=W′X+WX′=W⊕X

Similarly, we will get the following Boolean functions for C & D after simplifying.

C=X′Y+XY′=X⊕YC=X′Y+XY′=X⊕Y

D=Y′Z+YZ′=Y⊕ZD=Y′Z+YZ′=Y⊕Z

Circuit Diagram:

The following figure shows the circuit diagram of 4-bit binary code to Gray code converter.

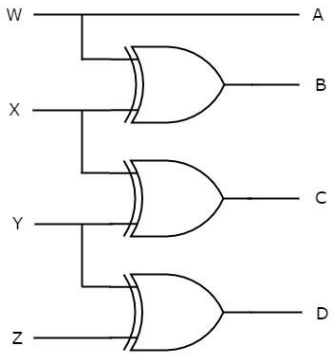


Fig. 4-bit binary code to Gray code converter

Since the outputs depend only on the present inputs, this 4-bit Binary code to Gray code converter is a combinational circuit. Similarly, you can implement other code converters.

b) Parity Bit Generator:

There are two types of parity bit generators based on the type of parity bit being generated. Even parity generator generates an even parity bit. Similarly, odd parity generator generates an odd parity bit.

i) Even Parity Generator ii) Odd Party Generator

i) Even Parity Generator:

Let us implement an even parity generator for a 3-bit binary input, WXY. It generates an even parity bit, P. If odd number of ones present in the input, then even parity bit, P should be '1' so that the resultant word contains even number of ones. For other combinations of input, even parity bit, P should be '0'. The following table shows the Truth table of even parity generator.

Binary Input WXY	**Even Parity bit P**
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1

Table. Even parity generator

From the above Truth table, we can write the Boolean function for even parity bit as

$P=W'X'Y+W'XY'+WX'Y'+WXYP=W'X'Y+W'XY'+WX'Y'+WXY$
$\Rightarrow P=W'(X'Y+XY')+W(X'Y'+XY)\Rightarrow P=W'(X'Y+XY')+W(X'Y'+XY)$
$\Rightarrow P=W'(X\oplus Y)+W(X\oplus Y)'=W\oplus X\oplus Y\Rightarrow P=W'(X\oplus Y)+W(X\oplus Y)'=W\oplus X\oplus Y$

The following figure shows the circuit diagram of even parity generator.

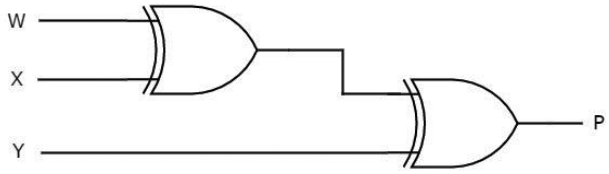


Fig. circuit diagram of even parity generator

This circuit consists of two Exclusive-OR gates having two inputs each. First Exclusive OR gate having two inputs W & X and produces an output W ⊕ X. This output is given as one input of second Exclusive-OR gate. The other input of this second Exclusive-OR gate is Y and produces an output of W ⊕ X ⊕ Y.

ii) Odd Parity Generator:

If even number of ones present in the input, then odd parity bit, P should be '1' so that the resultant word contains odd number of ones. For other combinations of input, odd parity bit, P should be '0'.Follow the same procedure of even parity generator for implementing odd parity generator. The circuit diagram of odd parity generator is shown in the following figure.

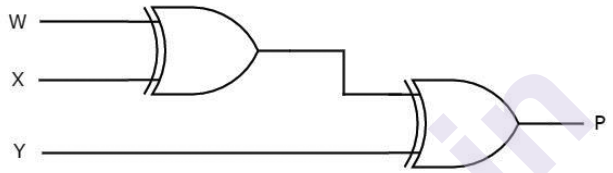


Fig: circuit diagram of odd parity generator

The above circuit diagram consists of Ex-OR gate in first level and Ex-NOR gate in second level. Since the odd parity is just opposite to even parity, we can place an inverter at the output of even parity generator. In that case, the first and second levels contain an Ex-OR gate in each level and third level consist of an inverter

c) Parity Checker:

There are two types of parity checkers based on the type of parity has to be checked. Even parity checker checks error in the transmitted data, which contains message bits along with even parity. Similarly, odd parity checker checks error in the transmitted data, which contains message bits along with odd parity

i) Even Parity Checker ii) Odd Party Checker

i) Even parity checker:

Let us implement an even parity checker circuit. Consider a 3-bit binary input, WXY is transmitted along with an even parity bit, P. So, the resultant word data contains 4 bits, which will be received as the input of even parity checker.

It generates an even parity check bit, E. This bit will be zero, if the received data contains an even number of ones. That means, there is no error in the received data. This even parity check bit will be one, if the received data contains an odd number of ones. That means, there is an error in the received data. The following table shows the Truth table of an even parity checker.

4-bit Received Data WXYP	Even Parity Check bit E
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

Table: Truth Table - even parity checker

From the above Truth table, we can perceive that the even parity check bit value is '1', when odd number of ones present in the received data. That means the Boolean function of even parity check bit is an odd function. Exclusive-OR function satisfies this condition. Hence, we can directly write the Boolean function of even parity check bit as

$=W \oplus X \oplus Y \oplus PE=W \oplus X \oplus Y \oplus P$

The following figure shows the circuit diagram of even parity checker.

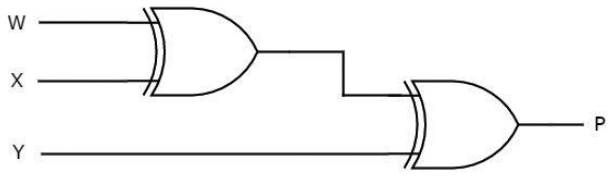


Fig: circuit diagram of even parity checker

This circuit consists of three Exclusive-OR gates having two inputs each. The first level gates produce outputs of W⊕XW⊕X & Y⊕PY⊕P. The Exclusive-OR gate, which is in second level produces an output of

W⊕X⊕Y⊕PW⊕X⊕Y⊕P

Odd Parity Checker:

Consider a 3-bit binary input, WXY is transmitted along with odd parity bit, P. So, the resultant word data contains 4 bits, which will be received as the input of odd parity checker.

It generates an odd parity check bit, E. This bit will be zero, if the received data contains an odd number of ones. That means, there is no error in the received data. This odd parity check bit will be one, if the received data contains even number of ones. That means, there is an error in the received data. Follow the same procedure of an even parity checker for implementing an odd parity checker. The circuit diagram of odd parity checker is shown in the following figure.

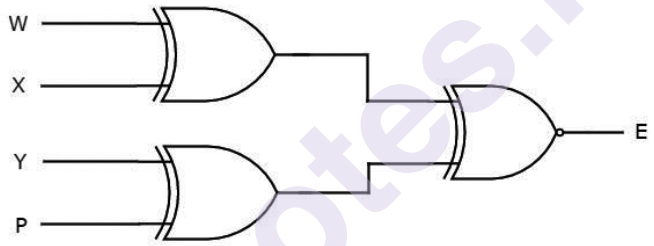


Fig: circuit diagram of odd parity checker

The above circuit diagram consists of Ex-OR gates in first level and Ex-NOR gate in second level. Since the odd parity is just opposite to even parity, we can place an inverter at the output of even parity checker. In that case, the first, second and third levels contain two Ex-OR gates, one Ex-OR gate and one inverter correspondingly.

*****

6

ARITHMETIC CIRCUITS

Unit Structure

6.1 INTRODUCTION

Combinational circuit is a circuit in which we have to combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are as below –

- The output of combinational circuit at any instant of time, depends only on the stages present at input stations.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an 'n' number of inputs and 'm' number of outputs.

Following are the various types of arithmetic circuits –

Adder, BCDAdder, Binarysubtractor, BCDsubtractor etc.

6.2 ADDER

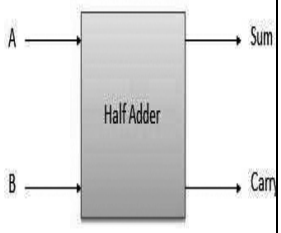
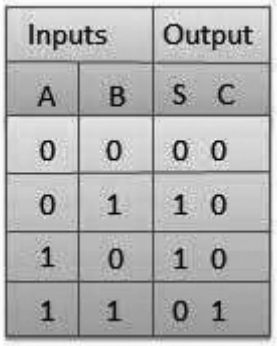
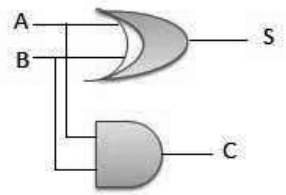
Adder arithmetic circuit can classified into two main types as –

a) Half Adder b) Full Adder

a) Half Adder:

Introduction:

Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two single bit numbers. This circuit has two outputs carry and sum.

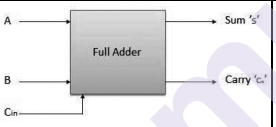
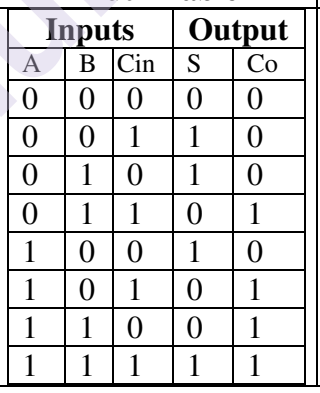
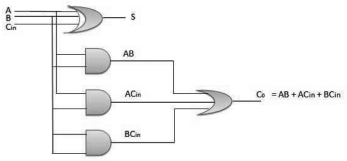
Block Diagram	Truth Table	Circuit diagram
		

Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

b) Full Adder:

Introduction:

Full adder is designed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry C. The full adder is a three input and two output combinational circuit

Block Diagram	Truth Table	Circuit diagram
		

Inputs			Output	
A	B	Cin	S	Co
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

N-Bit Parallel Adder:

Introduction:

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary

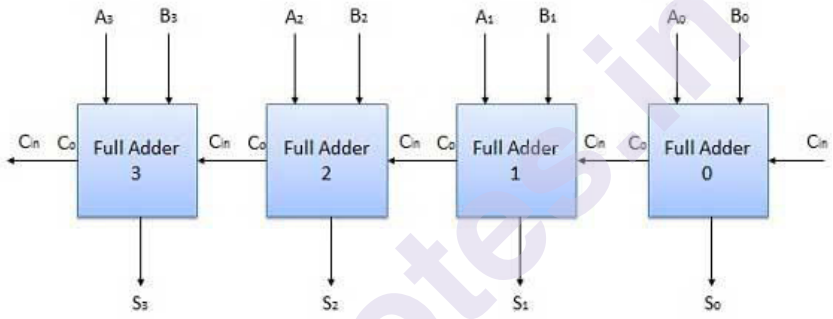
numbers which are much longer than just one bit. To add two n-bit binary numbers, we need to use the n-bit parallel adder. It uses a number of full adders in cataract. The carry output of the previous full adder is connected to carry input of the next full adder.

4. Bit Parallel Adder:

Introduction:

`In the block diagram, A_0 and B_0 represent the Least Significant Bits(LSB) of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are precisely same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

Block diagram:



6.4 BCD ADDER

Introduction:

A 4-bit binary adder that is accomplished of adding two 4-bit words having a BCD (binary-coded decimal) format. The result of the addition is a BCD-format 4- bit output word, demonstrating the decimal sum of the addend and augend, and a carry that is generated if this sum exceeds a decimal value of 9.

Use of BCD Adder:

A BCD 1-digit adder is a circuit that adds two BCD digits in parallel and also produces the Sum digit in BCD along with the essential correction logic. It can be seen that a 4-bit binary adder is used originally to add two BCD digits with a carry-input. When the binary sum is less than or equal to 9, it also correctly represents the sum in BCD. When the binary sum is greater than 9, however, it does not represent the correct BCD sum. The sum in BCD is to be attained by adding 6 to it.

In the BCD representation system each digit is encoded into its binary equivalent with four (4) bits.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table: BCD representation system

Observe that only 10 of the 16 possible bit-patterns are used in BCD. That means the remaining 6 patterns could be treated as don't-care cases. For the arithmetic addition of two decimal digits in BCD, the maximum value that may be produced as the result is 9 + 9 + 1 = 19 (two largest operands plus the carry). If we try to add two decimal digits in BCD with a 4-bit ripple-carry adder we will get a binary sum extending from 0 to 19. When the binary sum is less than or equal to 9, it may represents the sum in BCD. When the binary sum is greater than 9, however, it does not represent the correct BCD sum. The sum in BCD is to be obtained by adding 6 to it.

Perform the following addition in BCD as explained above

1) 7 + 4 2) 3 + 2 3) 9 + 9

The above examples should benefit you realize when the conversion is necessary and what should be done to perform the conversion correctly.

Here is a block diagram of a 1-digit BCD adder

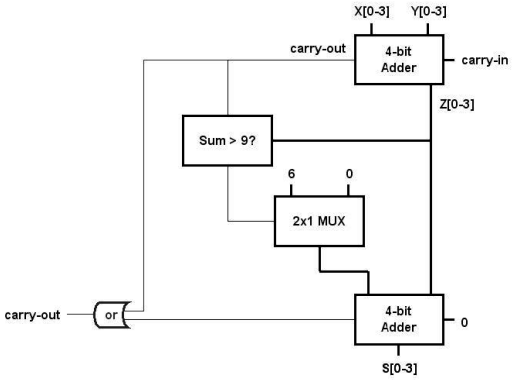


Fig. Block diagram of a 1-digit BCD adder

Complete the truth table below

Inputs		**Binary Sum**		**BCD Sum**		**Decimal Number**	
C_{in}	X	Y	C_{out}	Z	C_{out}	S	
0	0000	0000	0	0000	0	0000	0
0	0000	0001	0	0001	0	0001	1
0	0001	0001					
0	0101	0100					
0	0101	0101					
0	0101	0110					
0	1000	1000	1	0000	1	0110	16
0	1000	1001					
0	1001	1001					
1	1001	1001					

Table: Truth table - 1-digit BCD adder

Upon an examination of the 1-digit BCD adder block diagram shown above, you should notice that the only block you do not have a circuit for is the ―Sum > 9”? block. Having the circuit just excitingly appear on your circuit diagrams

3.4.4 Excess-3 Adder:

The excess-3 code is a non-weighted code used to express code used to express decimal numbers. It is a self-complementary Binary Coded Decimal (BCD) code and numerical system which has partial representation.

The primary benefit of excess-3 coding over non-biased coding is that a decimal number can be nines' complemented as easily as a binary number can be ones' complemented, just by inverting all bits.Excess-3 is a modified form of a BCD number. The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.

For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as-

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table: Truth Table - Excess-3 Adder

From the truth table, the logic expression for the Excess-3 code outputs can be written as,

$E_3 = \Sigma m\,(5, 6, 7, 8, 9) + \Sigma d\,(10, 11, 12, 13, 14, 15)$

$E_2 = \Sigma m\,(1, 2, 3, 4, 9) + \Sigma d\,(10, 11, 12, 13, 14, 15)$

$E_1 = \Sigma m\,(0, 3, 4, 7, 8) + \Sigma d\,(10, 11, 12, 13, 14, 15)$

$E_0 = \Sigma m\,(0, 2, 4, 6, 8) + \Sigma d\,(10, 11, 12, 13, 14, 15)$

K-map Simplification:

For E_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$E_3 = B_3 + B_2(B_1 + B_1)$

For E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$E_2 = B_2B_1'B_0' + B_2'(B_0 + B_1)$

For E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$E_1 = B_1'B_0' + B_1B_0$

$= B_1 \odot B_0$

For E_0

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

$E_0 = B_0'$

Circuit Diagram:

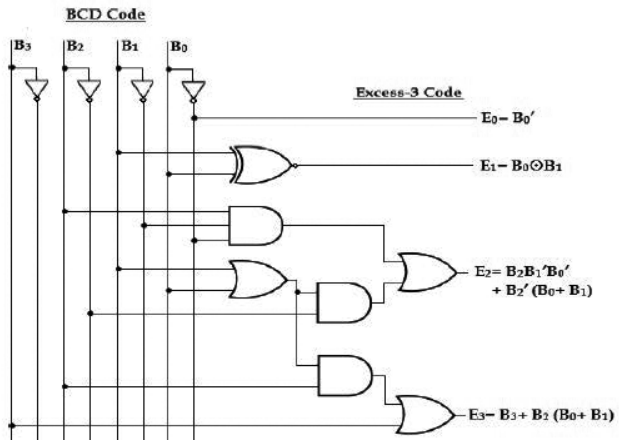


Fig.Excess-3 Adder circuit

6.6 BINARY SUBTRACTORS

Introduction:

The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction (A-B) by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4. Bit Parallel Subtractor:

Introduction:

The number to be subtracted (B) is first accepted through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S_3 S_2 S_1 S_0 represents the result of binary subtraction (A-B) and carry output C_{out} represents the polarity of the result. If A > B then C_{out} = 0 and the result of binary form (A-B) then C_{out} = 1 and the result is in the 2's complement form.

Block diagram:

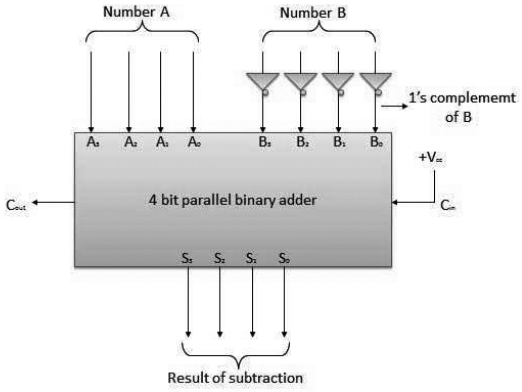


Fig. 4 Bit Parallel Subtractor

Types of Subtractors:

Subtractors circuit can classified into two main category as-

a) Half Subtractor b) Full Subtractors

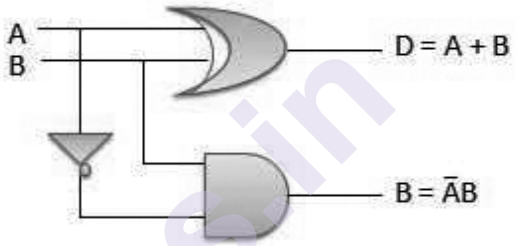
a) Half Subtractors:

Introduction:

Half subtractor is a combination circuit with two inputs and two outputs. It produces the difference between the two binary bits at the input and also produces an output to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Truth Table	Circuit Diagram

Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



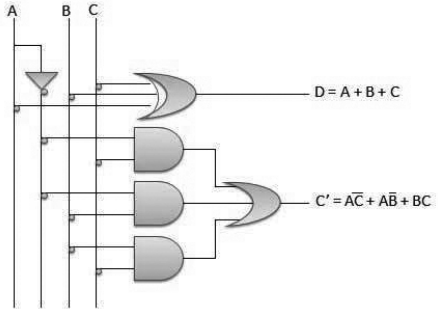
a) Full Subtractors

Introduction:

The drawback of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table	Circuit Diagram

Inputs			Output	
A	B	C	(A-B-C)	C'
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



6.7 BCD Subtractor

Introduction:

A Binary Coded Decimal (BCD) adder is a circuit which adds two 4-bit BCD numbers in parallel and produces a 4-bit BCD result. The block diagram of conventional BCD adder. The circuit must include the correction logic to produce valid BCD output.subtractor an electronic logic circuit for calculating the difference between two binary numbers, the minuend and the number to be subtracted, the subtrahend . A full subtractor performs this calculation with three inputs- subtrahend bit, minuend bit, and borrow bit

BCD Subtraction:

Addition of signed BCD numbers can be accomplished by using 9's or 10's complement methods. A negative BCD number can be expressed by taking the 9's or 10's complement.The BCD Subtraction using 9's Complement and BCD Subtraction using 10's Complement numbers and BCD Subtraction process using it.

9s Complement:

The 9's complement of a decimal number is found by subtracting each digit in the number from 9. The 9's complement of each of the decimal digits is as follows:

Digit	9's Complement
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

Table: 9's Complement

BCD Subtraction using 9's Complement:

In 9's Complement subtraction when 9's Complement of smaller number is added to the larger number carry is produced. It is essential to add this carry to the result. When larger number is subtracted from smaller one, there is no carry, and the result is in 9's complement form and negative. This is demonstrated in following ways as:

Regular subtraction **9's Complement Subtraction**

(a)
```
   8          8
 - 2        + 7      9's complement of 2
 ---        ---
   6      1   5
          ↳ + 1      Add carry to result
            ---
            6
```

(b)
```
   2  8       2 8
 - 1  3     + 8 6
 -------    -----
   1  5   1   1 4    9's complement of 13
          ↳ +   1
            -----
            1 5
```

(c)
```
   1  8       1 8
 - 2  4     + 7 5    9's complement of 24
 -------    -----
   -  6       9 3    9's complement of result
               ↓     (No carry indicates that
             - 0 6   the answer is negative and
                     in complement form)
```

The 10's complement of a decimal number is equal to the 9's complement plus 1.

BCD Subtraction using 10s Complement:

The BCD Subtraction using 10's Complement can be used to accomplish subtraction by adding the minuend to the 10's Complement of the subtrahend and dropping the carry. This is demonstrated in following ways as:

Regular subtraction **10's Complement Subtraction**

(a)
```
   8          8
 - 2        + 8      10's complement of 2
 ---        ---
   6        1 6      Drop carry
```

(b)
```
   2  8       2 8
 - 1  3     + 8 7    10's complement of 13
 -------    -----
   1  5     1 1 5    Drop carry
```

(c)
```
   1  8       1 8
 - 2  4     + 7 5    10's complement of 24
 -------    -----
   -  6       9 4    10's complement of result
               ↓     (No carry indicates that
               0 6   the answer is negative and
                     in complement form)
```

From the above examples we can review steps for 9's Complement BCD subtraction as follows:

- Find the 9's complement of a negative number
- Add two numbers using BCD addition
- If carry is generated add carry to the result otherwise find the 9s complement of the result.

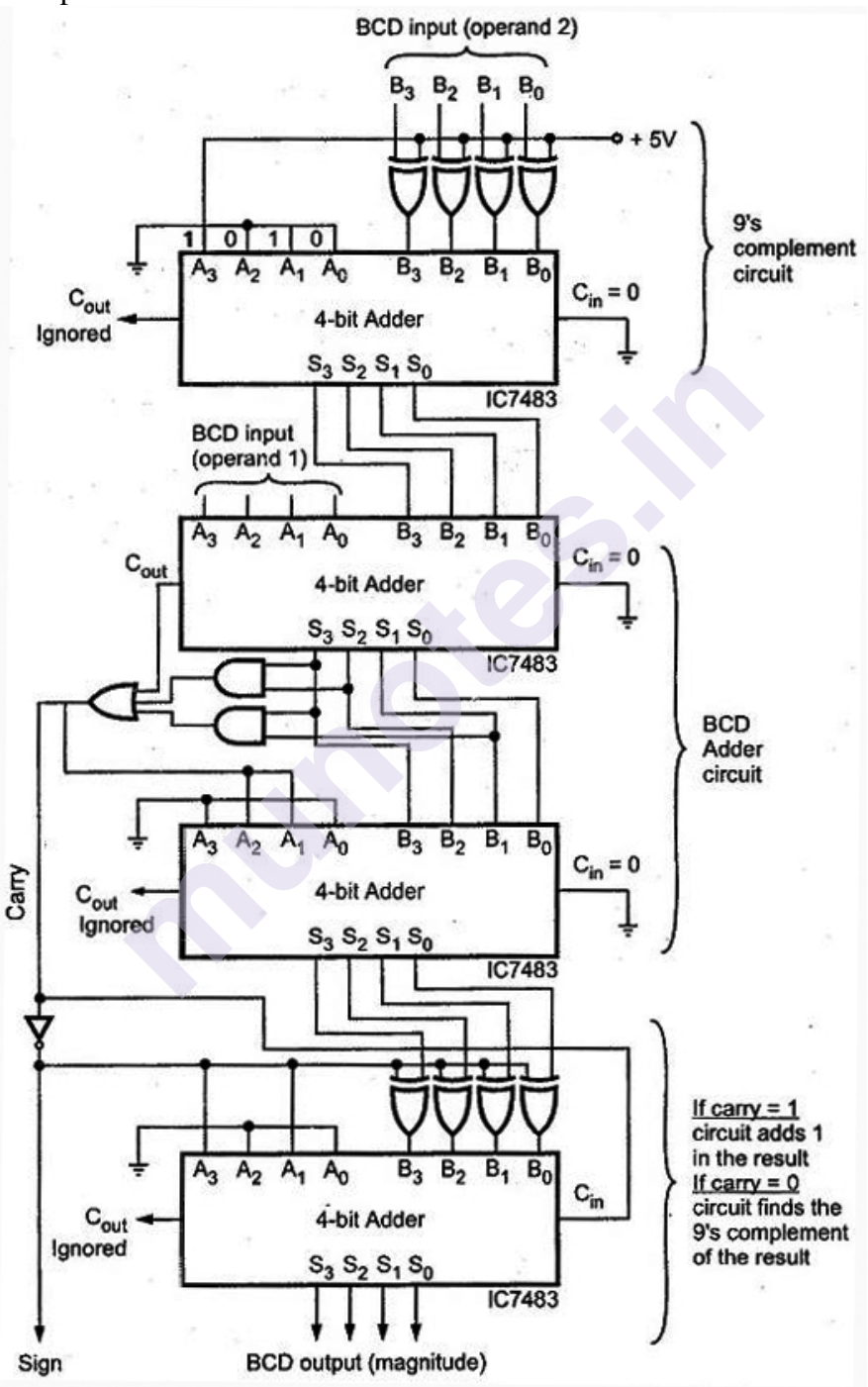


Fig.4 bit BCD substractor using 9's complement

Above figure demonstrate the logic diagram of the circuit to implement above stated steps to perform BCD subtraction using 9's Complement method. First binary adder finds the 9's Complement of the negative number. It does this by inverting each bit of BCD number and adding 10 (1 0 1 0_2) to it. Let us find the 9's complement of 2

		0	0	1	0	←BCD for 2
		1	1	0	1	←Inverting each bit
		1	0	1	0	←Add 10(1010_2)
Ignore carry →	1	0	1	1	1	←9's complement for 2

Next two 4-bit binary adders accomplish the BCD addition. The last adder finds the 9's Complement of the result if carry is not generated after BCD addition otherwise it adds carry in the result. From the above examples we can summarize steps for 10s complement BCD subtraction as:

- Find the 10's complement of a negative number
- Add two numbers using BCD addition
- If carry is not generated find the 10s Complement of the result.

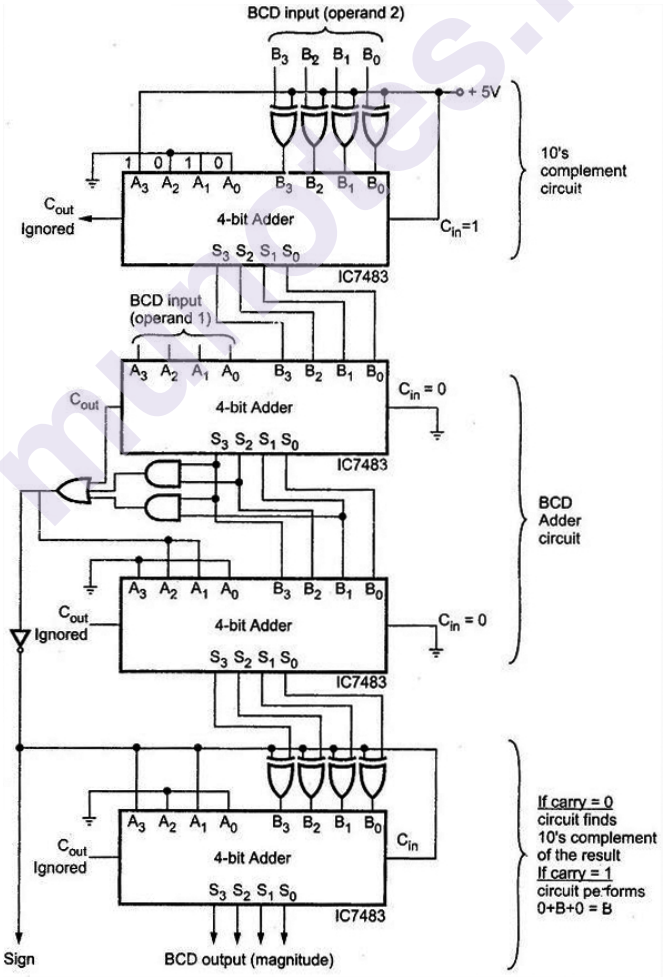


Fig.4 bit BCD substractorusing 10's complement

The logic diagram of the circuit to implement above stated steps to perform subtraction using 10s Complement method. First binary adder finds the 10's Complement of the negative number (9's complement + 1). Next two 4-bit binary adders perform the BCD addition. Lastly, last 4-bit binary adder finds the 10's complement of the number if carry is not generated after BCD addition.

*****

7

MULTIPLEXER

Unit Structure

7.1 INTRODUCTION

Multiplexer is a distinct type of combinational circuit. There are 'n' data inputs, one output and 'm' select inputs with 2m = 'n'. It is a digital circuit which selects one of the 'n' data inputs and routes it to the output. The selection of one of the 'n' inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of 'n' data sources is selected and transmitted to the single output 'Y'. 'E' is called the strobe or enable input which is useful for the cascading. It is commonly an active low terminal that means it will complete the required operation when it is low.

Block diagram:

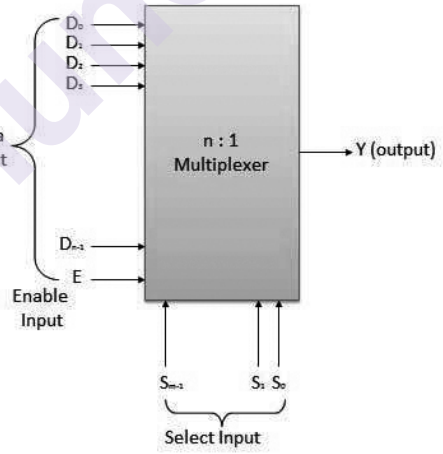


Fig: n:1 Multiplexer

Types of Multiplexer:

Multiplexers come in multiple variations

a) 2 : 1 multiplexer

b) 4 : 1 multiplexer

c) 16 : 1 multiplexer

d) 32 : 1 multiplexer

a) 2 : 1 multiplexer:

Block Diagram:

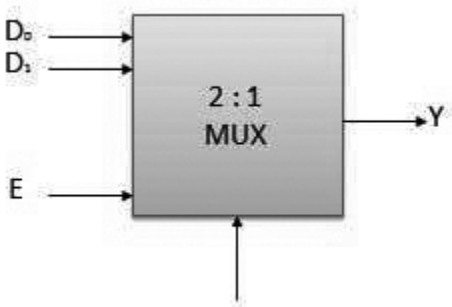


Fig.2:1 multiplexer

Truth Table:

Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

x = Don't care

Table: Truth Table

Demultiplexers:

Introduction:

A demultiplexer achieves the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.

Types of demultiplexer:

Demultiplexers comes in multiple variations

a) 1 : 2 demultiplexer

b) 1 : 4 demultiplexer

c) 1 : 16 demultiplexer

d) 1 : 32 demultiplexer

a) 1 : 2 demultiplexer

Block diagram:

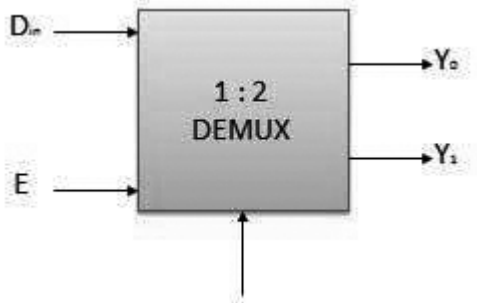


Fig. 1:2 Demux

Truth Table:

Enable	Select	Output	
E	S	Y_0	Y_1
0	X	0	0
1	0	0	D_{in}
1	1	D_{in}	0

Table : 1:2 Demux truth table

7.6 COMPARATOR

Introduction:

Comparator is a combinational logic circuit that compares the magnitudes of two binary quantities to determine which one has the greater magnitude. In other word, a comparator determines the association of two binary quantities. An ex-OR gate can be used as a basic comparator. Digital comparators are also called as binary or logic comparators.

Usage:

This logic circuit is used for testing whether the binary number at one input is greater than or less than or equal to another binary number. In other word, Comparator is a very useful combinational circuit capable of comparing two numbers as input in binary form and determines whether one number is greater than, less than or equal to other number. Comparators can also be used as window detectors. A comparator is used to compare two voltages and determine whether a given input voltage is under voltage or over voltage. Comparators are used in central processing unit (CPUs) and microcontrollers (MCUs).

Types of Comparators:

There are two types of comparators:

1. Equality or Identity Comparator
2. Magnitude or inequality comparator

1. Equality or Identity Comparator:

Identity Comparator is a digital comparator that has only one output. The circuit of the equality comparator is made up from an exclusive NOR gate (XNOR) per pair of input bits. If the two inputs are equal (both logic 1 or both logic 0) then a logic 1 is output

Consider two 4-bit binary numbers A and B so

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

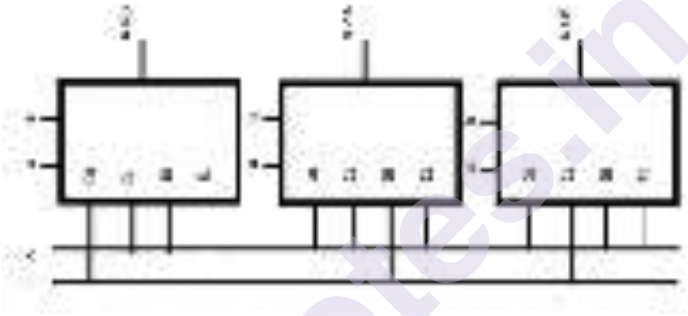


Fig. Identity Comparator

Here each subscript represents one of the digits in the numbers.

Equality:

The binary numbers A and B will be equal if all the pairs of significant digits of both numbers are equal, i.e.,

$$A_3=B_3,\ A_2=B_2,\ A_1=B_1 \text{ and } A_0 = B_0$$

Since the numbers are binary, the digits are either 0 or 1 and the Boolean function for equality of any two digits Ai and Bi can be expressed as

$$x_i = A_i B_i + \overline{A}_i + \overline{B_i}$$

we can also replace it by XNOR gate in digital electronics.

X_i is 1 *only if* A_i *and* B_i *are equal*

For the equality of A and B, all variables (for i=0,1,2,3) must be 1. So the equality condition of A and B can be implemented using the AND operation as

X_i is 1 *only if* A_i *and* B_i *are equal*

The binary variable (A=B) is 1 only if all pairs of digits of the two numbers are equal

2. Magnitude or inequality comparator:

To manually determine the greater of two binary numbers, you have inspect the relative magnitudes of pairs of significant digits, starting from the most significant bit, gradually proceeding towards lower significant bits until an inequality is found. When an inequality is found, if the corresponding bit of A is 1 and that of B is 0 then we conclude that A>B.

This sequential comparison can be expressed logically as:
Fig.2

$$(A > B) = A_3\overline{B_3} + x_3A_2\overline{B_2} + x_3x_2A_1\overline{B_1} + x_3x_2x_1A_0\overline{B_0}$$

$$(A < B) = \overline{A}_3B_3 + x_3\overline{A}_2B_2 + x_3x_2\overline{A}_1B_1 + x_3x_2x_1\overline{A}_0B_0$$

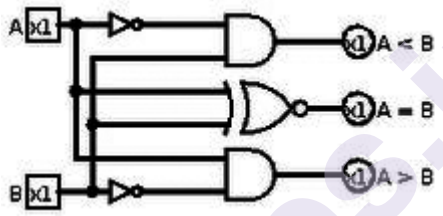


Fig.2 inequality comparator

(A>B) and (A < B) are output binary variables, which are equal to 1 when A>B or A<B respectively.

EXERCISE

1. List and Explain Applications of Encoder.
2. Explain in detail Binary Encoder

List of References

1. https://study.com/academy/lesson/basic-combinational-circuits-types-examples.html
2. https://www.electronics-tutorials.ws/combination/comb_1.html
3. https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm
4. https://www.tutorialspoint.com/digital_circuits/digital_arithmetic_circuits.htm
5. https://www.allaboutcircuits.com/textbook/digital/chpt-3/multiple-input-gates/

*****

Unit 4

8

MULTIPLEXER AND DEMULTIPLEXER

Unit Structure

8.0 OBJECTIVE

This chapter takes a comprehensive look at another class of building blocks used to design more complex combinational circuits, and covers building blocks such as multiplexers and demultiplexers and other derived devices such as encoders and decoders. Particular emphasis is given to the operational basics and use of these devices to design more complex combinational circuits.

8.1 MULTIPLEXER

- Multiplex means many into one.
- A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- It is also called a data selector is a device that selects between several analog or digital input signals and forwards it to a single output line

Block Diagram :

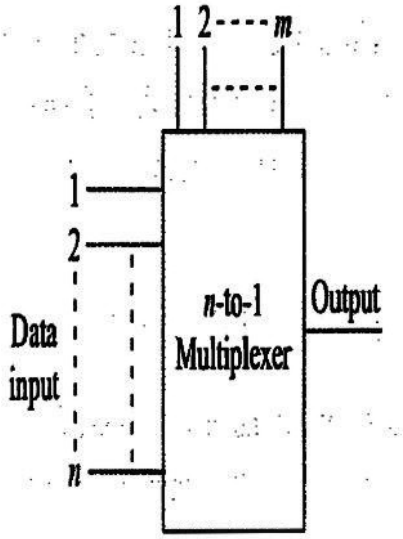


Fig.1 Block Diagram

The circuit has n input signals, m select signals and 1 output signal. Note that, m control signals can select at the most 2^m input signals thus n <=2^m 4x1 Multiplexer

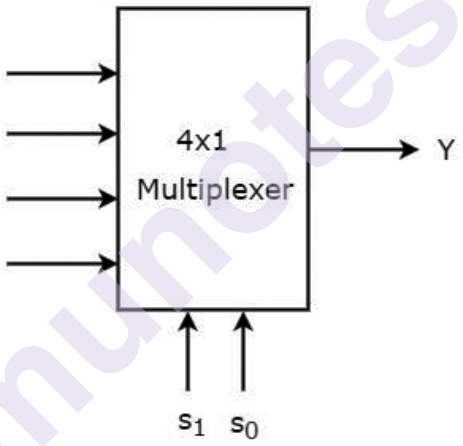


Fig.2 4x1 multiplexer block diagram

The circuit diagram of a 4-to-l multiplexer is shown in Fig. Depending on control inputs S_1 & S_0, one of the four inputs *Do* to *D3* is steered to output *Y*.

Let us write the logic equation of this circuit. Clearly, it will give a SOP representation, each AND gate generating a product term, which finally are summed by OR gate. Thus,

$Y=S_1'S_0'D_0 + S_1'S_0D_1 + S_1S_0'D_2 + S_1S_0D_3$

If S_1=0 and S_0= 0

$Y=0'\ 0'\ D_0 + 0'\ 0\ D_1 + 0\ 0'\ D_2 + 0\ 0\ D_3$

$Y=1.1\ D_0 + 1.0\ D_1 + 0.1\ D_2 + 0.0\ D_3$ (0'=1)

$Y= D_0$

In other words, for $S_1S_0 = 00$, the first AND gate to which D_0 is connected remains active and equal to D_0 and all other AND gate are inactive with output held at logic 0.

Thus, multiplexer output Y is same as D_0.

If D_0 =0, Y=0 and if D_0 = 1, Y= 1.

Control Signals		**Output**
S_1	**S_0**	**Y**
0	0	D0
0	1	D1
1	0	D2
1	1	D3

A logic diagram of 4-line to 1-line multiplexer is shown in figure 3

Each of the four input lines, D_0 to D_3, is applied to one input of an AND gate. Selection lines s_1 and s_0 are decoded to select a particular AND gate. The function table in the figure lists the input-to-output path for each possible bit combination of the selection lines.

To demonstrate the circuit operation, consider the case when s_1s_0 = 10. The AND gate associated with input D_2 has two of its inputs equal to 1 and the third input connected to D_2.

The other three AND gates have at least one input equal to 0, which makes their output equal to 0.

The OR-gate output is now equal to the value of D_2, thus providing a path from the selected input to the output.

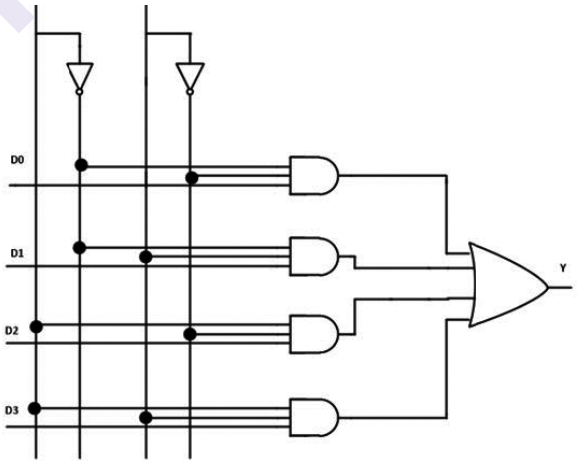


Fig.3 logic diagram of 4 to 1 multiplexer

8.2 DEMULTIPLEXER

- Demultiplex means one into many.
- A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2n possible output lines.
- The selection of a specific output line is controlled by the bit values of n selection lines.

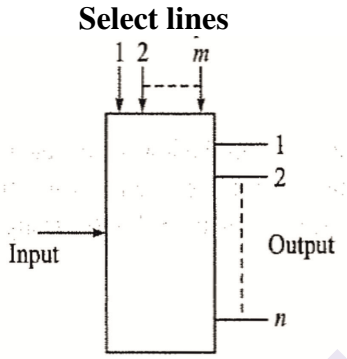


Fig.4. Demultiplexer Block diagram

Demultiplexer has single data input (D) and n outputs ($Y_0 - Y_{n-1}$).

While number of Select lines depends on number of outputs.

If 'n' is number of outputs and 'm' is number of select lines then the relation between them is given by $n = 2^m$.

1 to 4 Demultiplexer:

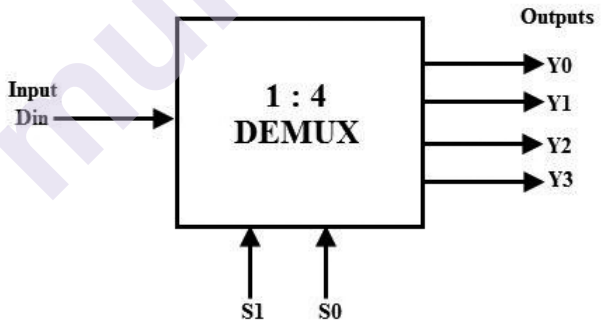


Fig. 5 Block diagram 1 to 4 Demultiplexer

The truth table of this type of demultiplexer is given below.

Input		Control Signals			Output	
S_1	S_0	Y_3	Y_2		Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

From the truth table it is clear that, when S_1=0 and S_0= 0, the data input is connected to output Y_0 and when S_1= 0 and S_0=1, then the data input is connected to output Y_1.

Similarly, other outputs are connected to the input for other two combinations of select lines.

$Y_0 = \overline{S_1}.\overline{S_0}D$

$Y_1 = \overline{S_1}.\overline{S_0}D$

$Y_2 = \overline{S_1}.\overline{S_0}D$

$Y_3 = \overline{S_1}.\overline{S_0}D$

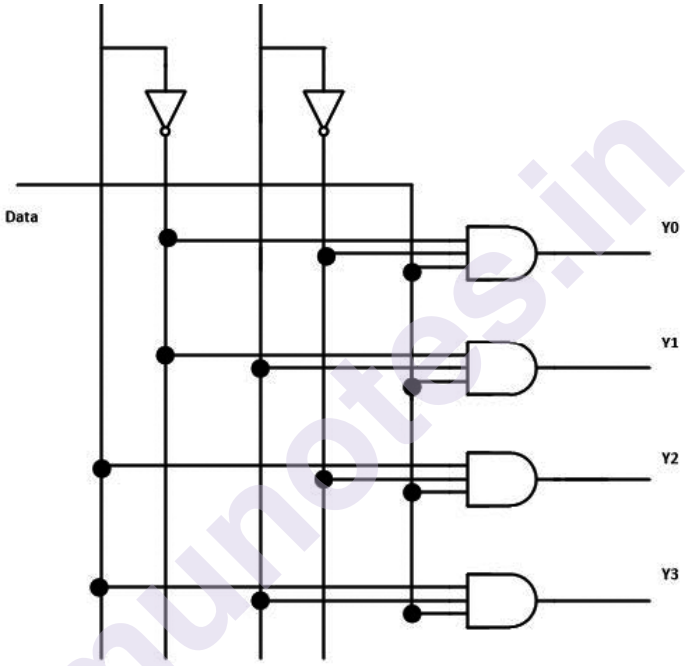


Fig.6 Logic Diagram 1 to 4 Demultiplexer

Decoder:

- A *decoder* is similar to a demultiplexer, with one exception-there is no data input.
- The only inputs are the Select signals.
- Decoder is a logic circuit that converts n-bit binary input code into m output lines.
- The decoders presented here are called n-to-m line decoders where m < 2n.
- Each output line will be activated for only one of the possible combination of inputs.

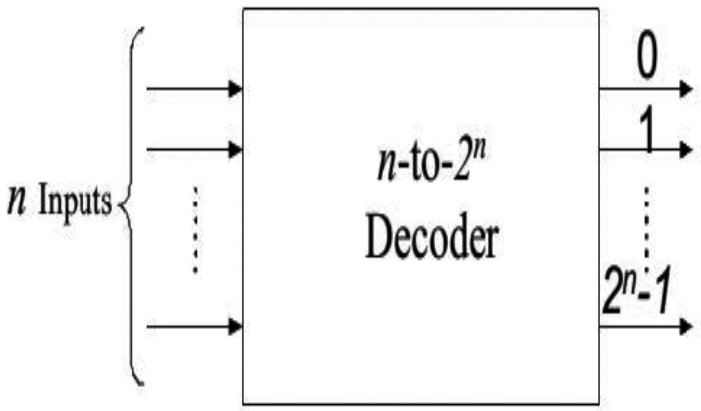
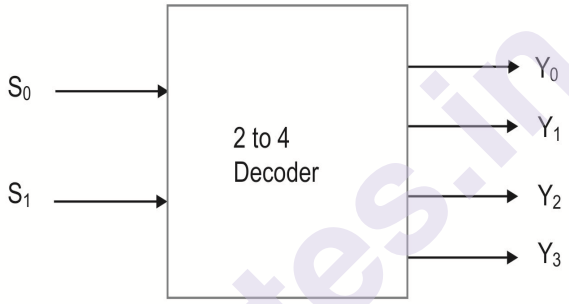


Fig. 7 Block diagram

2 to 4 Decoder:



The two inputs are decoded into four outputs, each output representing one of the minterms of the 2-input variables. The two inverters provide the complement of the inputs, and each one of the four AND gates generates one of the minterms. A particular application of this decoder would be a binary-to-octal conversion. The input variables may represent a binary number, and the outputs will then represent the four digits. However, a 2-to-4 line decoder can be used for decoding any 2-bit code to provide four outputs, one for each element of the code.

$Y_0 = \overline{S_1}.\overline{S_0}$

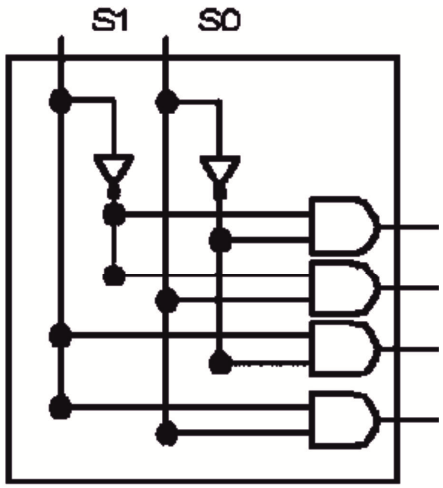
$Y_1 = \overline{S_1}.\overline{S_0}$

$Y_2 = \overline{S_1}.\overline{S_0}$

$Y_3 = \overline{S_1}.\overline{S_0}$

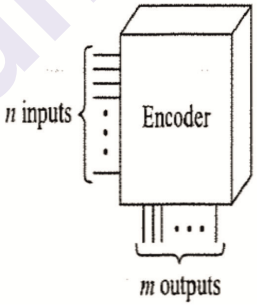
Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
X	X	0	0	0	0
0	0	0	0	0	1

0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Encoder:

- An Encoder is a combinational circuit that performs the reverse operation of Decoder.
- An encoder converts an active input signal into a coded output signal
- There are *n* input lines, only one of which is active.
- Internal logic within the encoder converts this active input to a coded binary output with *m* bits.



Decimal-to-BCD Encoder:

Figure shows a common type of encoder-the decimal-to-BCD encoder.
The switches are push-button switches like those of a pocket calculator.

When button 3 is pressed, the C and D OR gates have high inputs; therefore, the output is

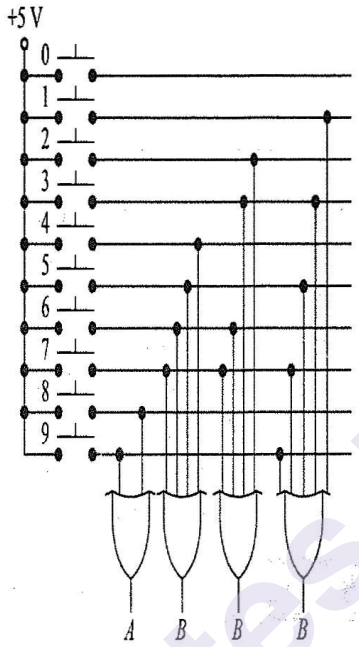
ABCD=0011

If button 5 is pressed, the output becomes

ABCD=0101

When switch 9 is pressed,

ABCD= 1001



Encoders	Decoders
Encoders may have more than one input line active and may have more than one output line active at any given time	Decoders may have more than one input line active at any given time but only one output line will be active
Number of input lines is more than number of output lines	Number of output lines is more than number of input lines
Number of input lines = 2Number of output lines	Number of output lines = 2Number of input lines
Encoder is logic device used to create binary code for given decimal input	Decoder is logic device used to decode binary input to give decimal output

Priority Encoder:

A *priority encoder* is a practical form of an encoder. The encoders available in IC form are all priority encoders. In this type of encoder, a priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded.

In previous cases we saw that if two or more input lines are activated then output code is invalid. Therefore we have to modify the circuit. The modification is we have to define the priority of given number. It means whenever two or more inputs are applied at a time, internal hardware will check this condition and if priority is set such that higher input is to be taken into account and remaining are considered as don't care then output code that will appear is of higher input.

Arithmetic Logic Unit (ALU):

The *arithmetic logic unit* (ALU) is a digital building block capable of performing both arithmetic as well as logic operations. Arithmetic logic units that can perform a variety of arithmetic operations such as addition, subtraction, etc., and logic functions such as ANDing, ORing, EX-ORing, etc., on two four-bit numbers are usually available in IC form. The function to be performed is selectable from *function select* pins. Functional details of these ICs are given in the latter part of the chapter under the heading of *Application-Relevant Information*.

More than one such IC can always be connected in cascade to perform arithmetic and logic operations on larger bit numbers.

8.5 QUESTIONS

1. Define Multiplexer.
2. Write short note on 4 –to- 1 Channel Multiplexer.
3. Explain 4- to -2 Channel Multiplexer With help of suitable diagram.
4. What is De-Multiplexer?
5. What is Digital Encoder? Explain 4- to -2 Bit Binary Encoder with help of suitable diagram.
6. What is Priority Encoder? Explain 8- to -3 Priority Encoder with the help of suitable diagram.

*****

9

FLIP-FLOPS

Unit Structure

9.0 OBJECTIVE

- After completing this chapter, you will be able to: Understand the basics of Sequential Logic Circuits.
- Learn different types of Flips –Flops, their working and applications with the help of suitable diagrams.

9.1 INTRODUCTION TO SEQUENTIAL CIRCUITS

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

Sequential logic elements perform as many different functions as combinational logic elements; however, they do carry out certain well-defined functions, which have been given names.

Latch:

A latch is a 1-bit memory element. You can capture a single bit in a latch at one instant and then use it later; for example, when adding

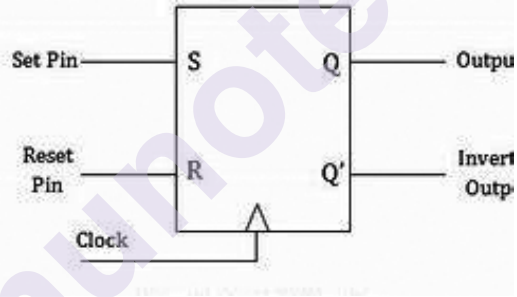
numbers, you can capture the carry-out in a latch and use it as a carry-in in the next calculation.

Register: The register is just m latches in a row and is able to store an m-bit word; that is, the register is a device that stores one memory word. A computer's memory is just a very large array of registers.

Flip-flop:

In the electronics world, a **flip-flop** is a type of circuit that has two states (i.e., on or off, 1 or 0). These circuits are often used to store state information. By sending a signal to the flip-flop, the state can be changed. Flip-flops are used in many electronics, including computers and communications equipment.

Flip-flops and latches are used as data storage elements. Such data storage can be used for storage of *state*, and such a circuit is described as sequential logic. When used in a finite-state machine, the output and next state depend not only on its current input, but also on its current state (and hence, previous inputs). It can also be used for counting of pulses, and for synchronizing variably-timed input signals to some reference timing signal. A flip-flop can be symbolically represented as shown below:



Generally, Set and Reset Pins are input pins; whereas Q and Q` are output pins.

When Set Pin= logic 1, the output Q is SET to 1. (Q`= 0)

When Reset Pin=logic 1, the output Q is RESET to 0. (Q`=1)

Note that Q` is complement of Q at all times.

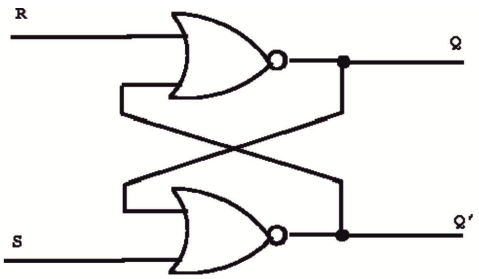
9.2 SR FLIP- FLOP

The R-S flip-flop is the most basic of all flip-flops. The letters ‘‘R’’ and ‘‘S’’ here stand for RESET and SET.

It is constructed by feeding the outputs of two NOR gates back to the

other NOR gates as shown below:

RS Flip-Flop composed of two NOR Gates:



To understand the operation of the RS-flip-flop (or RS-latch) consider the following scenarios:

- **S=1 and R=0:**
  - The output of the bottom NOR gate is equal to zero, Q'=0.
  - Hence both inputs to the top NOR gate are equal to one, thus, Q=1.
  - Hence, the input combination S=1 and R=0 leads to the flip-flop being set to Q=1.

- **S=0 and R=1:**
  - The output of the top NOR gate is equal to zero, Q=0.
  - Hence, the inputs to the bottom NOR gate are equal to 1, thus
  - Q`=1
  - We say that the flip-flop is reset, that is reset to Q=0.

- **S=0 and R=0:**
  - Assume the flip-flop is set (Q=0 and Q'=1), then the output of the top NOR gate remains at Q=1 and the bottom NOR gate stays at Q'=0.
  - Similarly, when the flip-flop is in a reset state (Q=1 and Q'=0), it will remain there with this input combination.
  - Therefore, with inputs S=0 and R=0, the flip-flop remains in its state.
- **S=1 and R=1**
  - We can summarize the operation of the RS-flip-flop by the following truth table.
  - This input combination must be avoided

We can summarize the operation of the RS-flip-flop by the following truth table.

R	S	Q	Q	Comment
0	0	Q	Q`	Hold State
0	1	1	Q`	Set
1	0	0	1	Reset
1	1	?	?	Avoid

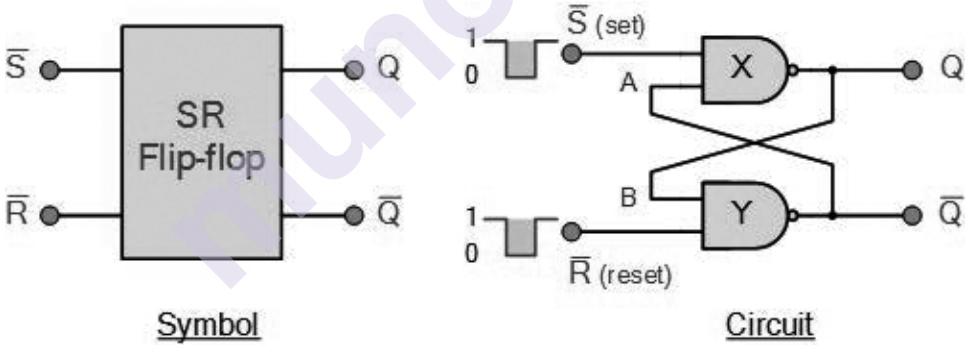
Note, the output Q' is simply the inverse of Q.
An RS flip-flop can also be constructed from NAND gates.

S-R Flip-Flop Using NAND Gate:

A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs. The term "Flip-flop" means that the output can be "flipped" into one logic Set state or "flopped" back into the opposing logic Reset state.

The NAND Gate SR Flip-Flop

Basic SR Flip-Flop Circuit:



The Set State:

To understand the operation of the RS-flip-flop (or RS-latch) consider the following scenarios:

S`=1 and R`=0:

- In a NAND gate, if one of the input =0, then output of NAND gate=1.
- Using this logic, the lower NAND gate Y will be Q'=1, assuming B=0.

- Let us consider the upper NAND gate X now. A=0 since Q'=1 from above, but S'=1. Therefore, output of X will be 0, that is Q=0. This is exactly what we had assumed when we said let B=0.
- Thus, the SR flip-flop is said to be RESET to 1. Q=0. Q'=1

S`=0 and R`=1:

- Using the same logic, the upper NAND gate X will be Q=1, assuming A=0.
- Let us consider the lower NAND gate Y now. B=1 since Q=1 from above, but R`=1. Therefore, output of Y will be 0, that is Q`=0. This is exactly what we had assumed when we said let A=0.
- We say that the flip-flop is set, that is set to Q=1. Q`=0.

S'=0 and R`=0:

- S`=0 and R`=0 is not desired or invalid condition. This must be avoided. This condition will cause both Q and Q` to go high together to logic 1. In such a case, the flip-flop will become unstable.

S`=1 and R`=1:

- When both S` and R` are equal to logic HIGH, the outputs Q and Q` can be at 1 or 0 logic level. This depends on the states of S` and R` before this input condition existed. Thus, S`=R`=1 results in no change of state of the outputs.

Truth Table for this Set-Reset Function

S`	R`	Q	Q`	State
1	1	Previous state	Previous state	No change
1	0	0	1	RESET
0	1	1	0	SET
0	0	?	?	Forbidden

9.3 D FLIP-FLOP

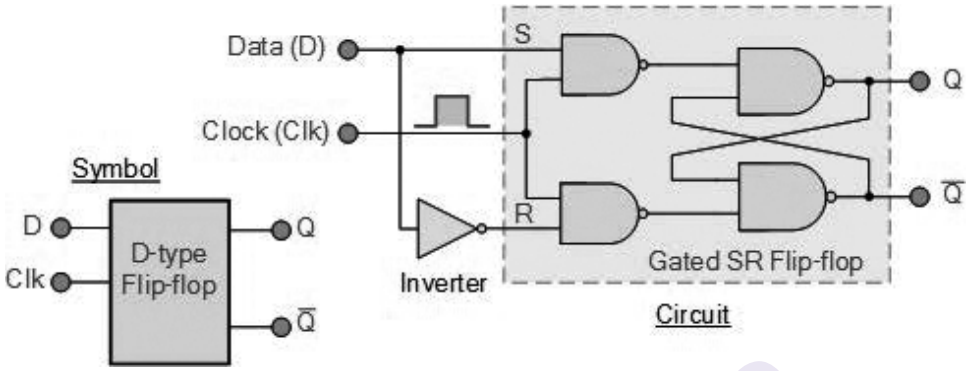
The Delay Flip-Flop or D Flip-flop is easily formed from SR flip-flop by adding a NOT gate(inverter) as shown in the logic diagram

We have observed that S'=0, R'=0 is a forbidden condition in SR NAND gate flip-flop. We can prevent this from happening by connecting a NOT gate between S and R inputs.

The D flip-flop ensures that S and R can never be simultaneously equal to each other.

The single input then is called the Data input or simply D input.

D-type Flip-Flop Circuit:



If D=1, then S=1 in the above circuit resulting in Q=1 and Q'=0

If D=0, then R=1 in the above circuit resulting in Q'=1 and Q=0.

Thus, the D input condition is copied to the output Q when the clock input is active. Once the clock input goes low, the output at Q and Q` will remain unchanged. We say that the output is "latched" at logic 0 or logic 1.

Truth Table for the D-type Flip Flop:

Clk	**D**	**Q**	**Q`**	**Description**
↓ » 0	X	Q	Q`	Memory no change
↑ » 1	0	0	1	Reset Q » 0
↑ » 1	1	1	0	Set Q » 1

Note that: ↓ and ↑ indicates direction of clock pulse as it is assumed D-type flip flops are edge triggered

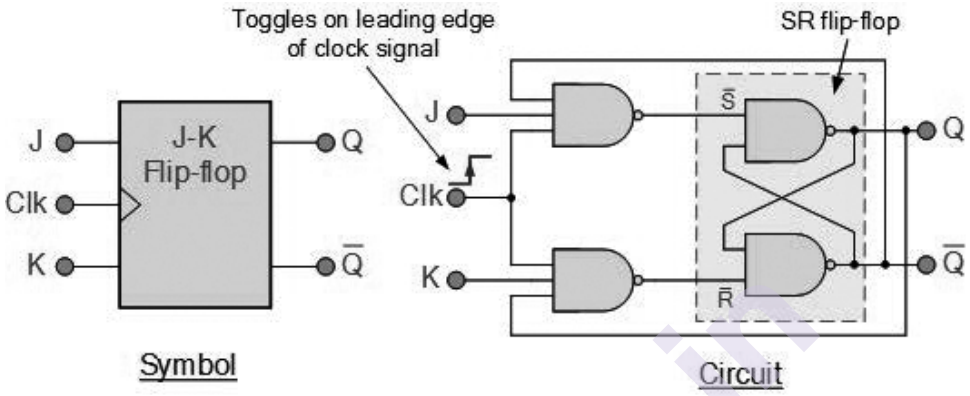
9.4 JK FLIP-FLOP

We have seen that S=0 and R=0 condition in SR NAND flip-flop is forbidden and should be avoided. Also, while Enable input is HIGH, the correct latching may not occur, if S or R changes state during this period. To overcome these two problems, the JK flip-flop was developed. The inputs J and K are derived from its inventor Jack Kilby.

The operation of JK flip-flop is same as the SR flip-flop for same S and R inputs. The difference is that it has no forbidden or invalid states of SR flip-flop.

The JK flip flop is obtained by adding two NAND gates to SR NAND gate flip-flop as shown below:

The Basic JK Flip-flop:



Both the S and the R inputs of the previous SR bi-stable have now been replaced by two inputs called the J and K inputs. Thus, J = S and K = R.

The two 2-input AND gates of the gated SR bi-stable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q'. This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked.

If the circuit is now "SET", then the J input is inhibited by the "0" status of Q through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q' are always different we can use them to control the input. When both inputs J and K are equal to logic "1", the JK flip flop toggles as shown in the following truth table

The Truth Table for the JK Function:

	Clock	Input		Output		Description
same as for the SR Latch	Clk	J	K	Q	Q"	
	X	0	0	1	0	Memory
	X	0	0	0	1	no change
	↓	0	1	1	0	Reset Q » 0
	X	0	1	0	1	
	↓	1	0	0	1	Set Q » 1
	X	1	0	1	0	
toggle action	↓	1	1	0	1	Toggle
	↓	1	1	1	0	

9.5 RACE-AROUND CONDITION

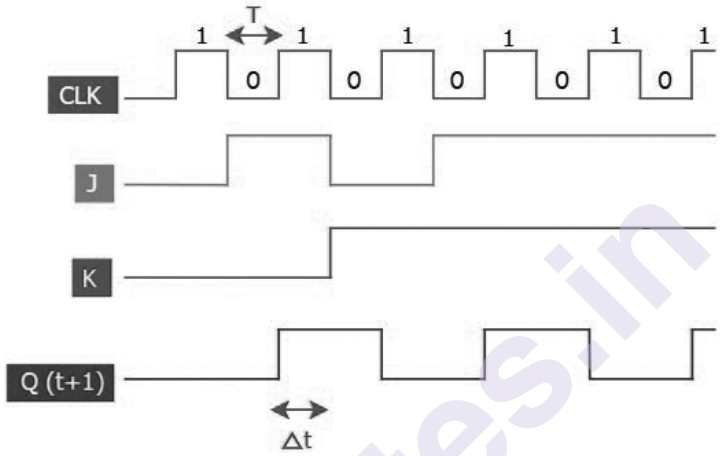
Before getting into the *race around condition*, let us have a look at the JK flip-flop's truth table.

	Input		**Outputs**				
	J	**K**	**Q**	**Q`**			
	0	X	X	Same as previous		Same as previous	No change
Clock Input					**comments**		
	1	0	0	Same as previous		Same as previous	No change

	1	0	1	0		1	Reset
	1	1	0	1		0	set
1	1	1	Opposite of previous		Toggle		

Here, Q is the present state and Q` is the next state. As you can see, when J, K and Clock are equal to 1, toggling takes place, i.e. The next state will be equal to the complement of the present state.

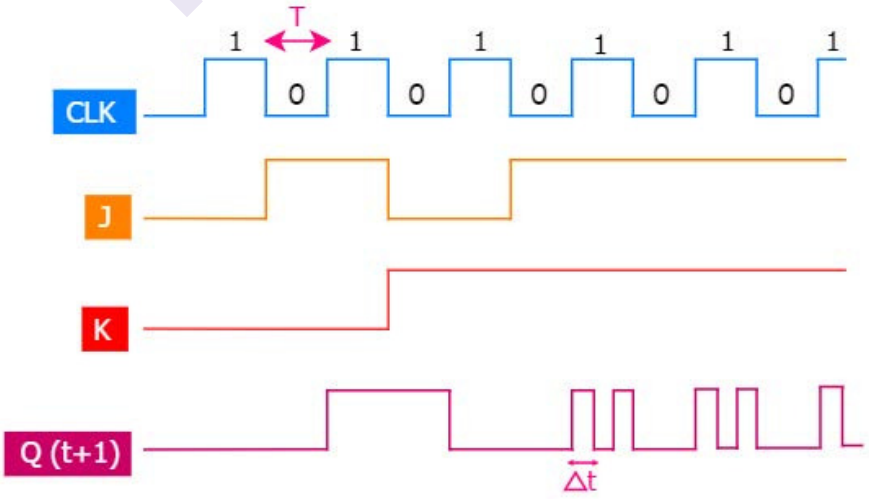
Now, let us look at the timing diagram of JK flip-flop.



Here, T is the time period of the clock whereas delta t is the propagation delay. The delay between input and output is called a propagation delay.

This is what was expected, but the output may not be like this all the time. This is where **Race around condition** comes into the play.

Let us look at the timing diagram of JK flip-flop when the race around condition is considered



When J, K and Clock are equal to 1, toggling takes place. Here, propagation delay has also been reduced, so the output will be given out at the instant input is given. So there is a toggling again. Therefore, whenever Clock is equal to 1 there are consecutive toggling. This condition is called as Race around condition. To put it in words, "For JK flip-flop if J, K and Clock are equal to 1 the state of flip-flop keeps on toggling which leads to uncertainty in determining the output of the flip-flop. This problem is called Race around condition. "'' This condition also exists in T flip-flop since T flip-flop also has toggling options.

9.6 MASTER-SLAVE JK FLIP-FLOP

- The master slave JK flip flop is a combination of a clocked JK flip-flop and a clocked SR flip-flop. The clocked JK flip-flop acts as the master and the clocked SR flip-flop acts as the slave.

- Master is positive level triggered and due to the presence of an inverter in the clock line, the slave is negative level edge triggered. Hence when clock=1, the master is active and slave is inactive. Vice versa happens when clock=0.

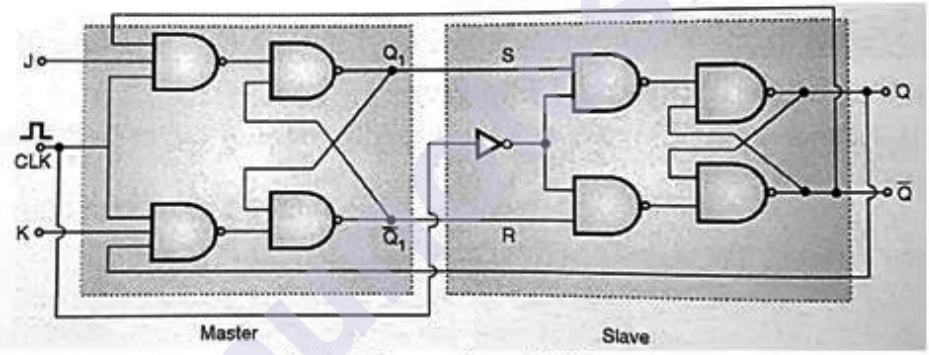


Fig6. Master slave JK FF

- The following is truth table of master slave flip flop.

Case	Input			Output		Remark
	CLK	J	K	Q_{n+1}	$\bar{Q}_{n+1}$	
I	X	0	0	Q_n	$\bar{Q}_n$	No Change
II	⎍ (1)	0	0	Q_n	$\bar{Q}_n$	No Change
III	⎍ (1)	0	1	0	1	Reset
IV	⎍ (1)	1	0	1	0	Set
v	⎍ (1)	1	1	$\bar{Q}_n$	Q_n	Toggle

Fig 7 Truth table of Master slave jk FF

Operation:

Case I: When clock is not given, both master and slave are inactive and there will be no change in outputs.

Case II: For clock=1, master is active, slave inactive. As J=K=0, output of master Q and Q' will not change. As soon as clock goes to 0, slave becomes active, and master is inactive. But since input to slave S and R is same, output of slave will also remain same.

Case III: For clock=1, master is active and slave is inactive. When J=0 and K=1, outputs of master will be Q=0, Q'=1, which will be inputs to slave. When clock=0, slave becomes active and takes inputs 0,1 to give output Q=0, Q'=1. This output will not change if clock is again made 1and then 0. Hence we get a stable output from master and slave.

Case IV: For clock=1, master is active and slave is inactive. When J=1 and K=0, outputs of master will be Q=1, Q'=0, which will be inputs to slave. When clock=0, slave becomes active and takes inputs 1,0 to give output Q=1, Q'=0. This output will not change if clock is again made 1 and then 0. Hence we get a stable output from master and slave.

Case V: When clock =1, J=K=1, master output will toggle. So S and R will invert. But slave remains inactive all this time since clock is 1. As soon as clock becomes 0, slave becomes active and master becomes inactive. So slave will also toggle. These changed outputs are returned through feedback to the master, but master does not respond to them because clock is now 0 and master is inactive. Thus, in one clock period, master and slave both toggle only once, avoiding race condition caused by multiple toggling

9.7 T FLIP-FLOP

We can construct a T flip – flop by any of the following methods

(1) Connecting the output feedback to the input, in SR flip – flop.

(2) Connecting the XOR of T input and Q PREVIOUS output to the Data input, in D flip – flop.

(3) Wiring the J and K inputs together and connecting it to T input, in JK flip – flop. This is illustrated in the figures below

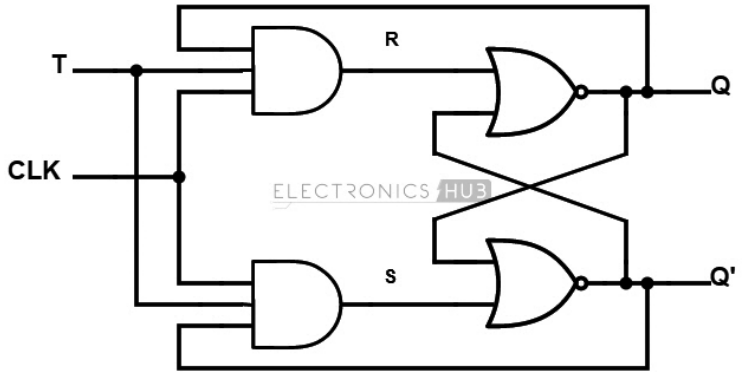


Figure (1) From SR flip flop

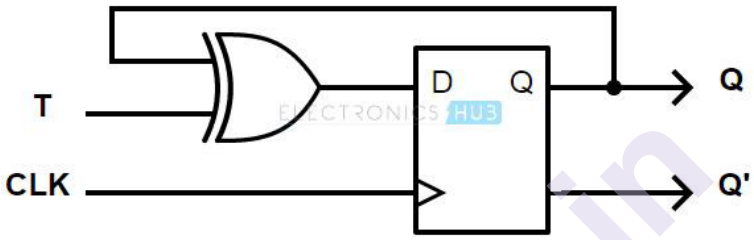


Figure (2) From D Flip flop

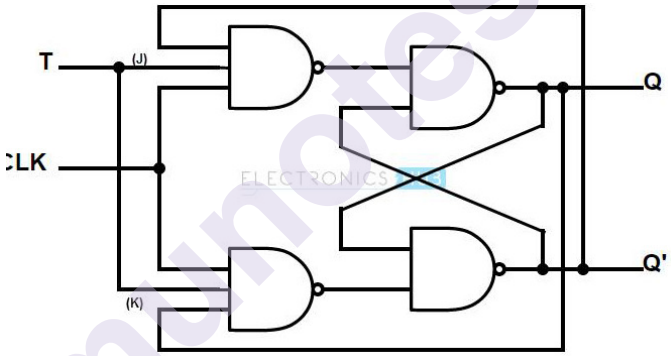


Figure (3) From JK Flip flop

Working:

Toggle Flip flop changes its output whenever it is edge-triggered. What it means is that whenever the clock changes its state from low to high or high to low.

Truth Table of T flip – flop

	Previous		Next	
T	Q_{Prev}	$Q`_{Prev}$	Q_{Next}	$Q`_{Next}$
0	0	1	0	1
0	1	0	1	0
1	0	1	1	0
1	1	0	0	1

The disadvantage of T flip flop is that the state of the flip-flop at an applied trigger is known only when read with the previous state.

T flip flops are not available as Integrated Circuits(ICs). But, they can be easily constructed using SR flip-flop, JK flip-flop or D flip-flop as shown above.

9.8 CONVERSION OF FLIP-FLOP FROM ONE TYPE TO ANOTHER

We have discussed four flip-flops-SR, D, JK and T flip-flops so far. It is possible to convert one flip-flop from one type to another very easily. The steps required for such conversion are:

1. Consider the truth table of the desired flip-flop.
2. Fill in the excitation values of the flip-flop in hand for each combination of present and next state.
3. Get a simplified expression for each input using Karnaugh Maps.
4. Draw the logic circuit diagram of the flip-flop to be formed according to the simplified expression. Use flip-flop in hand and logic gates to achieve this

Let us understand this with an example. Let us consider converting SR flip- flop to a D flip flop.

Step 1. Prepare truth table of desired flip flop, that is D flip flop.

Input	Present state	Next state
D	Q (t)	Q (t+1)
0	0	1
0	1	0
1	0	1
1	1	1

Step 2. SR flip-flop has two inputs S and R. Write the excitation values of SR flip flop for each combination of present state and next state. The above table will now be modified to

Input	Present state	Next state	SR flip-flop inputs	
D	**Q(t)**	**Q(t+1)**	**S**	**R**
0	0	1	0	X

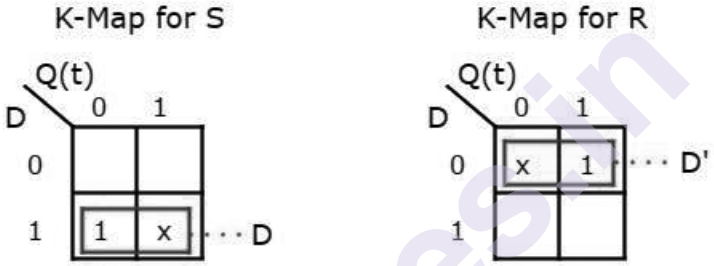
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Step 3.

From the above table, we can write the **Boolean functions** for each input as below.

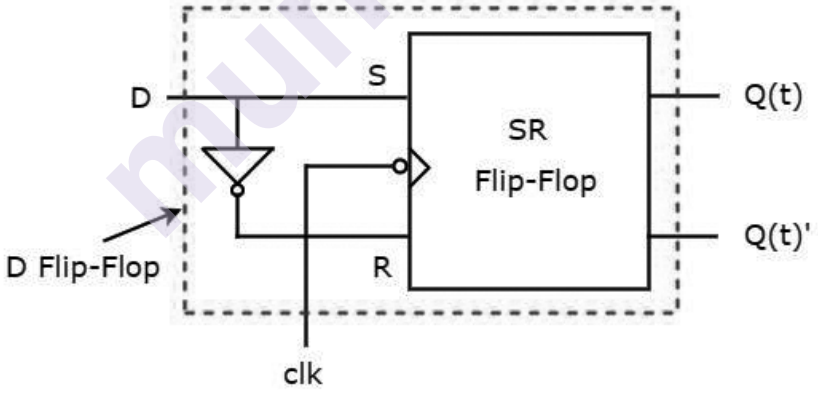
S=m2+d3S=m2+d3
R=m1+d0R=m1+d0

We can use 2 variable K-Maps for getting simplified expressions for these inputs. The **k-Maps** for S & R are shown below



So, we got S = D & R = D' after simplifying

Step 4.
The **circuit diagram** of D flip-flop is shown in the following figure



This circuit contains a NOT gate connected in addition to SR flip-flop.

Other conversions can be similarly worked out.

9.9 APPLICATIONS OF FLIP-FLOP

Flip-flops are used in numerous applications, such as

(1) Registers

(2) Counters

(3) Event Detectors

(4) Data Synchronizers

(5) Frequency Divider

1) Registers:

Registers are storage devices used to store memory. Each flip-flop can store a single bit. Figure 1 shows cascading three D flip-flops to store 3-bit information.

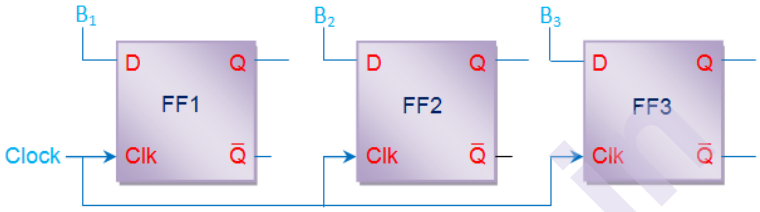


Figure 1 3-bit register formed by cascading three D flip-flops

The data can be shifted within registers in/out of the register by applying clock pulses. These registers are called shift registers and can be pictorially represented as in Figure 2.

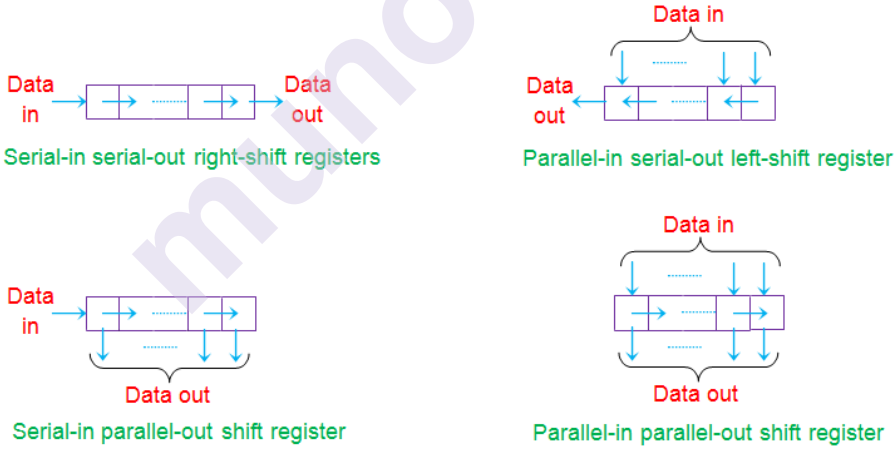


Figure 2 Various kinds of shift registers

2) Counters:

Counters are used for the purpose of counting. A series of flip – flops are cascaded to form counters. These counters can be synchronous or asynchronous. They can be positive-edge triggered or negative-edge triggered. Counters are used as up-counter, down counter, ring counter,

Johnson counter etc. Figure 3 shows a 3-bit asynchronous positive edge triggered up-counter.

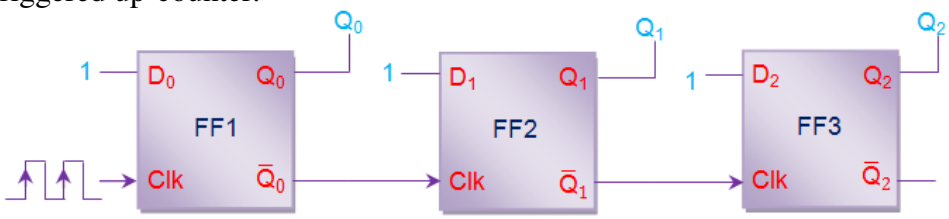


Figure 3 3-bit asynchronous positive edge triggered up-counter

3) Event Detectors:

These are circuits which are used to find occurrence of a particular event. Flip-flops do not change their state unless triggered. This can be used to detect and store occurrence of an event. Figure 4a shows one such event detector which detects the event of switching ‘‘ON’’ of light. The working is illustrated in Figure 4b.

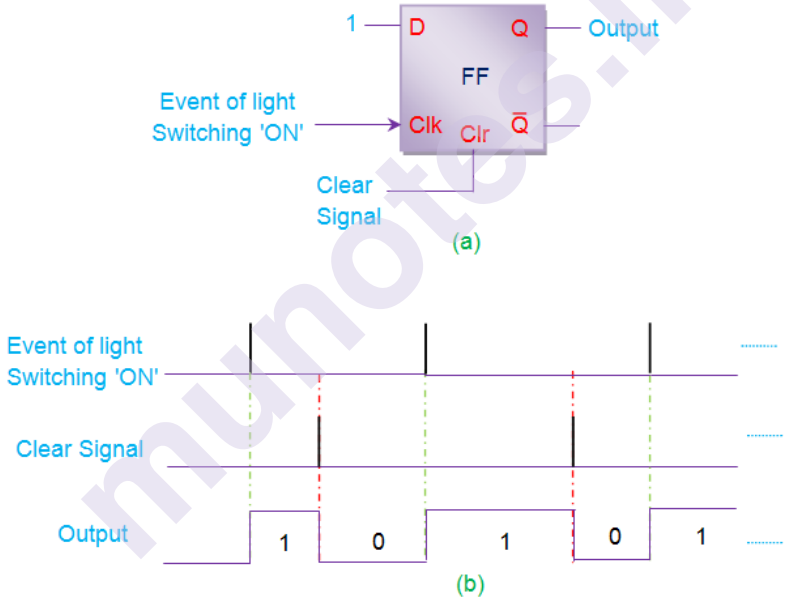


Figure 4 Event detector (a) Circuit (b) Timing diagram

4) Data Synchronizers:

Outputs of a particular combinational circuit should change their states simultaneously. Using Data synchronizers, this can be easily achieved as illustrated in Figure 5 below

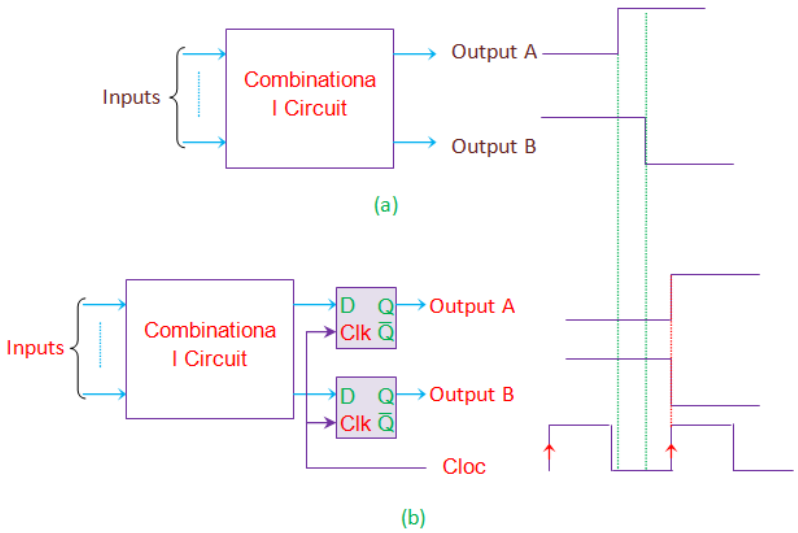


Figure 5 Data synchronizing application of flip-flops

5) Frequency Divider:

Consider a positive edge triggered JK flip-flop as shown in Figure 6 below

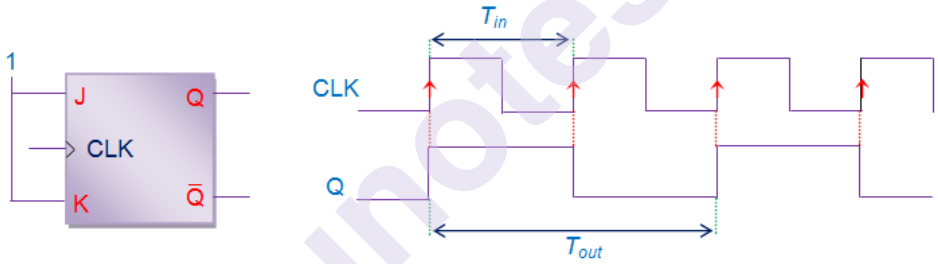


Figure 6 JK flip-flop as a frequency divider

The output of JK flip-flop will toggle for each positive-edge of the clock. It is clearly seen from the waveforms that if Tin is the input clock period, then Tout= 2Tin. In other words, fout=fin/2. This is how frequency division takes place using flip-flops.

*****

Unit 5

10

COUNTERS

Unit Structure

10.0 OBJECTIVES

This chapter would make you understand the following concepts

What is counter? Different types of counters – Asynchronous and synchronous counter. Terms related to counters. IC 7493 (4-bit binary counter) Synchronous counter, Bushing, Type T Design, Type JK Design.

Presettable counter, IC 7490, IC 7492. Synchronous counter ICs, Analysis of counter circuits.

10.1 INTRODUCTION

In the design of counters and registers FFs (flip-flops) are most widely used. The FF is the basic building block of any sequential logic system. FF and combinational circuit are used in the design of any sequential system.

Counter is a sequential circuit. A counter is a device that stores the number of times a particular event or process has occurred, often in relationship to CLK (a clock) pulse. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit.

10.1.1 Definition:

A digital circuit which is used for counting pulses is known *counter.* Counter is the widest application of FFs. It is a group of FFs with a CLK pulse applied. Counter is a register that goes through a prescribed series of states. Counter is a circuit which cycle through state sequence.

10.1.2 Types of Counters:

A digital circuit which is used for counting pulses is known *counter.* Counter is the widest application of FFs. It is a group of FFs with a CLK pulse applied. Counter is a register that goes through a prescribed series of states. Counter is a circuit which cycle through state sequence.

10.1.3 Types of Counters:

The number of FFs used and the way in which they are connected determines the number of states and also the specific sequence of states that the counter goes through during each complete cycle. Counters are classified according to the way they are clocked: *Asynchronous or ripple counters* and *Synchronous counters*

Asynchronous counter	Synchronous counter
1. It is also called as serial counter. 2. Simple and straight forward in operation. 3. Slower than synchronous. 4. Next FF is triggered by previous FF. 5. Problem of glitch. 6. Settling time is more.	1. It is also called as parallel counter. 2. Complex in operation as compared to asynchronous. 3. Faster than asynchronous. 4. All FFs are triggered simultaneously by external CLK. 5. No problem of glitch. 6. Settling time is less.

Synchronous / asynchronous counter can be further divided in to following sub-type:

1. **Regular Counter:** FFs are used to build the regular counter, in which the number of FFs determines the number of states means there is direct relation between number of FFs used and number of states of counter.

   Suppose **'N'** is number of States in counter and **'m'** is number of FFs then the relation is **N** (number of States) = $\mathbf{2}^{\mathbf{m\ (\#\ of\ FFs)}}$

   Let's say **m (# of FFs) = 3** then $\mathbf{N = 2^m = 2^3 = 8}$

   $\therefore$ Number of States (N) in counter = 8

   Consider one additional variable **'n'**, which indicates the number of actual states of counter. In regular counter, number of actual states of counter (**n**) and number of states of counter (**N**) are equal i.e. **n = N**.

2. **Truncated counter:** in this counter, number of actual states of counter (n) are always less than number of states of counter (N) i.e. $n < N$. If FFs are 3, then $N = 2^m = 8$ but $n < 8$.

3. **Sequential Counter:** in this counter, states of counter are sequential i.e. 0, 1, 2, 3, 4, 5, 6, … so on.

4. **Non-sequential Counter:** in this counter, states of counter are not sequential means states are irregular. e.g. 0, 3, 9, 8, 2, 1, 7.

10.2 ASYNCHRONOUS OR RIPPLE COUNTERS

Asynchronous counter is a cascaded arrangement of FFs where the output of one FF drives the CLK input of the following FF. The number of FFs in the cascaded arrangement depends upon the number of different logic states that it goes through before it repeats the sequence, a parameter known as the modulus of the counter.

Asynchronous counter, also called ripple counter or a serial counter, the CLK input is applied only to the first FF, also called the input FF, in the cascaded arrangement. The CLK input to any subsequent FF comes from the output of its immediately preceding FF. For instance, the output of the first FF acts as the CLK input to the second FF, the output of the second FF feeds the CLK input of the third FF and so on. In general, in an arrangement of 'n' FFs, the CLK input to the nth FF comes from the output of the $(n-1)^{th}$ FF for $n>1$.

Figure 10.2 shows the generalized block diagram of an n-bit binary ripple counter. As a natural consequence of this, not all FFs change state at the same time. The second FF can change state only after the output of the first FF has changed its state. That is, the second FF would change state a certain time delay after the occurrence of the input CLK pulse owing to the fact that it gets its own CLK input from the output of the first FF and not from the input CLK. This time delay here equals the sum of propagation delays of two FFs, the first and the second FFs. In general, the nth FF will change state only after a delay equal to n times the propagation delay of one FF. The term 'ripple counter' comes from the mode in which the CLK information ripples through the counter. It is also called an 'asynchronous counter' as different FFs comprising the counter do not change state in synchronization with the input CLK. In a counter like this, after the occurrence of each CLK input pulse, the counter has to wait for a time period equal to the sum of propagation delays of all FFs before the next CLK pulse can be applied. The propagation delay of each FF, of course, will depend upon the logic family to which it belongs.

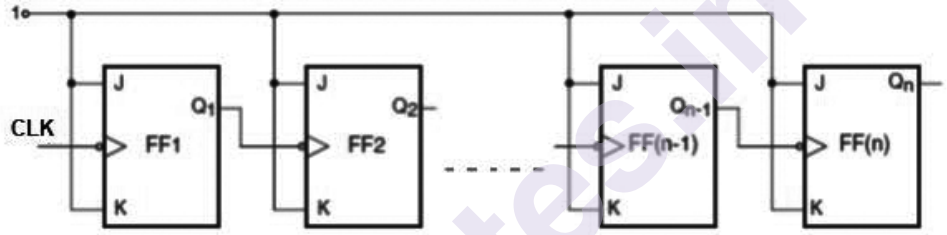


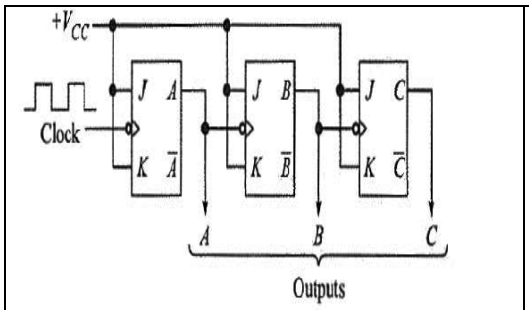
Figure 10.2: Block diagram of an n-bit binary ripple counter

10.2.3 Bit Ripple Counter:

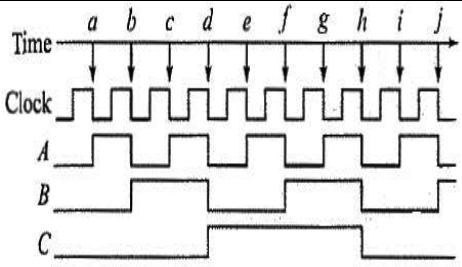
A binary ripple counter can be constructed using clocked JK FFs. Figure 10.2.1(a) shows three negative edge – triggered, JK FFs connected in cascade. The system clock, a square wave, drives FF A. The output of A drives FF B, and the output of B drives FF C. All the J and K inputs are tied to $+V_{CC}$. This means that each FF will change state (toggle) with a negative transition at its clock input.

When the output of a FF is used as the clock input for the next FF, we call the counter a ripple counter, or asynchronous counter. The A FF must change state before it can trigger the B FF, and the B FF has to change state before it can trigger the C FF. The triggers move through the FFs like a ripple in water. Because of this, the overall propagation delay time is the sum of the individual delays.

For instance, if each FF in this three- FF counter has a propagation delay time of 10 ns, the overall propagation delay time for the counter is 30 ns.



(a) Three-bit binary ripple counter



(b) Waveforms

Negative clock transitions	C	B	A	State or count
---	0	0	0	0
a	0	0	1	1
b	0	1	0	2
c	0	1	1	3
d	1	0	0	4
e	1	0	1	5
f	1	1	0	6
g	1	1	1	7
h	0	0	0	0

(c) Truth table

Figure 10.2.1: 3 - bit Ripple Counter

The waveforms given in Fig. 10.2.1(b) show the action of the counter as the clock runs. Let's assume that the FFs are all initially reset to produce 0 outputs. If we consider A to be the least-significant bit (LSB) and C the most-significant bit (MSB), we can say the contents of the counter is CBA = 000.

Every time there is a clock NT (Negative Transition), FF A will change state. This is indicated by the small arrows (↓) on the time line. Thus at point a on the time line, A goes high, at point b it goes back low, at c it goes back high, and so on. Notice that the waveform at the output of FF A is one-half the clock frequency.

Since A acts as the clock for B, each time the waveform at A goes low, FF B will toggle. Thus at point b on the time line, B goes high; it then goes low at point d and toggles back high again at point f Notice that the waveform at the output of FF B is one-half the frequency of A and one-fourth the clock frequency.

Since B acts as the clock for C, each time the waveform at B goes low, FF C will toggle. Thus C goes high at point d on the time line and goes back low again at point h. The frequency of the waveform at C is one-half that at B, but it is only one-eighth the clock frequency.

10.3 TERMS RELATED TO COUNTERS

Before we proceed further, we would like to study few things related to counter.

10.3.1 State Diagram:

State diagram means graphical representation of the states of counter circuit. For example state diagram for 3 bit binary ripple up counter and 3 bit binary ripple down counter can be as shown in Figures 10.3.1(a) and 10.3.1(b) respectively.

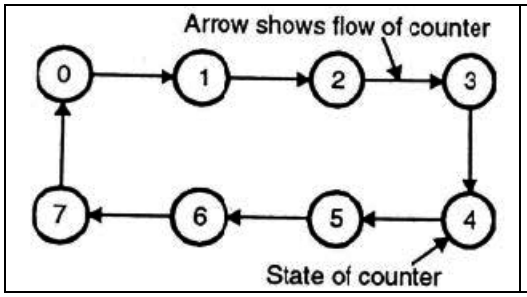


Figure 10.3.1(a): 3 bit binary ripple up counter

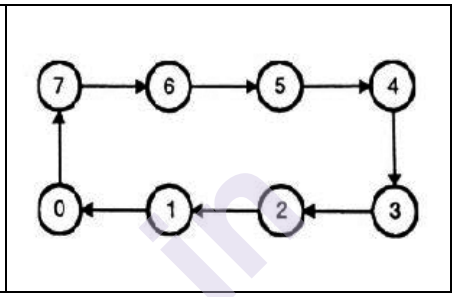


Figure 10.3.1(b): 3 bit binary ripple down Counter

10.3.2 Module N Counter:

If it is desired to have modulo N (mod – N) counter, the number of FFs required is determined by,

$N \leq 2^m$ where **m = number of FFs**

Let say m = 4, therefore N = 16. But if we require only 10 states out of 16, it is called as modulus – 10 (Mod – 10) counter, but required FFs will be 4 only.

Mod – N can be achieved by resetting the FF. This should be done at the N state. In Mod – 10, counter will count from 0 to 9, and at 10th state it will reset back to 0. Mod – 10 counter is also called as **decade**

counter.

Up till now we have seen that counter is sequential and number of states provided are $N = 2^m$. Now let us take a case of truncated ripple counter when n (number of actual state) < N.

Example 1: Design **mod – 3 ripple counter**

Solution:

1) **State diagram = 3 states → 0 to 2**

2) No of actual states are **n = 3**
n < N = 2^m 3 ≤ 2^m therefore **m = 2**
Hence **2 FFs** are required.

3) Here FF used should have clear terminal for resetting or clearing.

4) **Assumption:** considering FF and gates are having 0 propagation delay.

5) Reset logic circuit required to terminate count after 2.

6) 2 FFs are used therefore N = 4

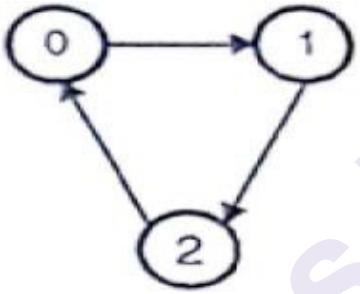


Figure 10.3.2.1(a): state diagram

QB	QA	
0	0	0
0	1	1
1	0	2
1	1	3

Valid States

Invalid State so whenever this state occurs countershould beCLEARED(Reset)

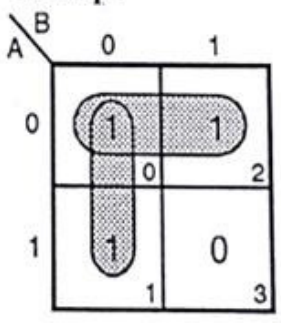
Normally CLR terminal is ACTIVE LOW therefore Design a reset logic combinational circuit such that output Y should be 1 when valid states are there and Y = 0 when Invalid states are present.

Truth Table:

Q_B	Q_A	Y O/P
0	0	1
0	1	1
1	0	1
1	1	0

→ Reset FF

K-Map :



$Y = \overline{A} + \overline{B}$ ...(a)
$= \overline{AB}$ ...(b)

Circuit Diagram:

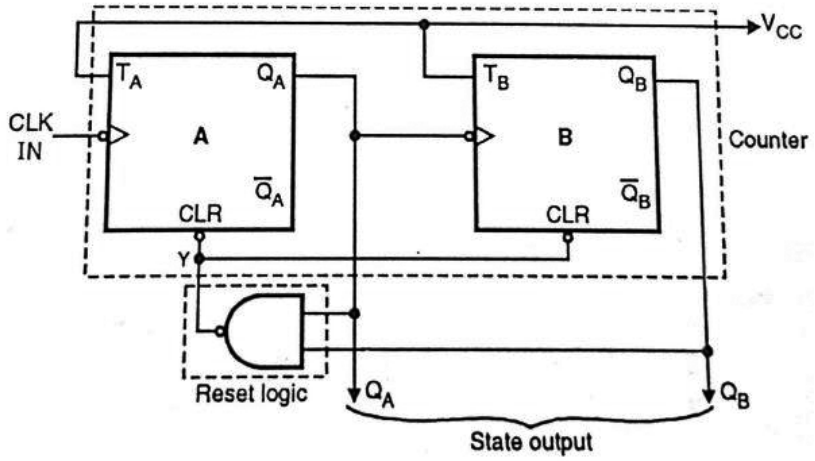


Figure 10.3.2.1(b): mod – 3 ripple counter: circuit diagram

Counter is up counter, therefore FF should be –ve edge triggered, cascade Q O/P and final O/P is taken from Q. The final circuit diagram is therefore shown in figure 10.3.2.1(b). Wave form for the same is shown in following figure 10.3.2.1(c).

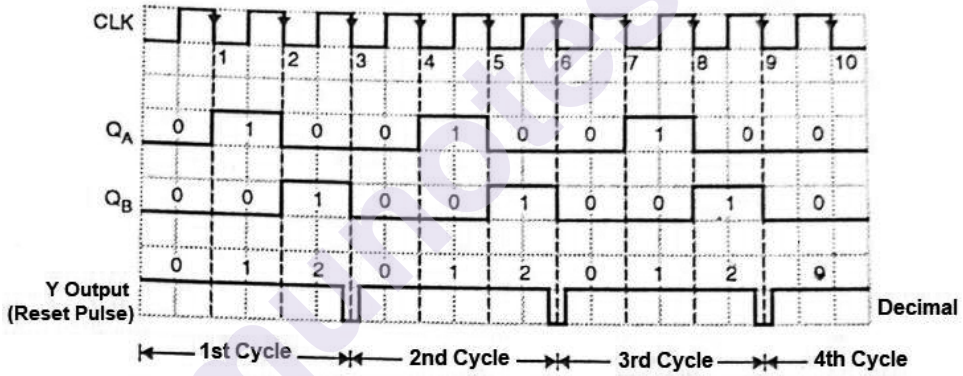


Figure 10.3.2.1(c): mod – 3 ripple counter: wave form

Working: to check the working refer the figures 10.3.2.1(b) and 10.3.2.1(c) (**QB QA = B A**)

1) Initially all FFs are cleared.
   $\therefore$ BA = 00; $\therefore$ Y = 1

2) When 1st negative CLK edge hits, A toggles
   $\therefore$ A = 1, $\therefore$ BA = 01; Y = 1

3) At 2nd hit of CLK edge, A toggles from 1 → 0 (negative edge),
   $\therefore$ B also toggles from 0 → 1.
   $\therefore$ BA = 10 $\therefore$ Y = 1

4) When 3rd CLK edge hits, A will toggle from 0 → 1(positive edge)
∴ B = unchanged ∴ BA = 11
∴ Y = 0. As Y = 0, all FFs will be cleared.
and BA = 00. As BA = 00, Y = 1 again.

The sequence from BA = 11 to BA = 00, is so fast (as propagation delay is considered 0 ηsec) that, it is not possible to observe 11 condition of waveform. Y waveform pulse is also very sharp and of very, very small duration. Thus counter will run 0 → 1 → 2 → 0.

Example 2: Design **mod – 6 ripple counter**

Solution:

1) State diagram = 6 states → 0 to 5

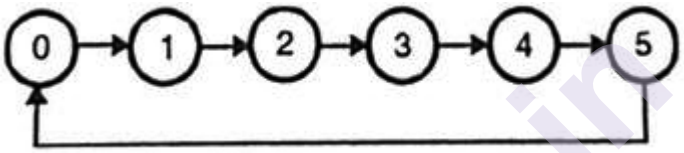


Figure 10.3.2.2(a): state diagram

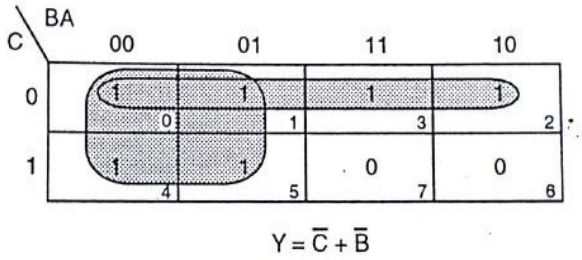
2) No. of states = 6 therefore $\mathbf{n < N = 2^m\ 6 < 2^m}$ therefore **m = 3**

3) Design of reset circuit. Truth table is,
$\mathbf{N = 2^m = 2^3 = 8,}$ but **n = 6**

Truth Table:

	C	B	A	Y O/P
Valid	0	0	0	1
	0	0	1	1
	0	1	0	1
	0	1	1	1
	1	0	0	1
	1	0	1	1
Invalid	1	1	0	0
	1	1	1	0

K-map:



Points:

1) Counter is up counter, therefore we have to cascade Q output, -ve edge triggered FF and final output from Q.
2) FF should have ACTIVE LOW reset (clear) terminal.
3) Propagation delay presently assumed 0 n sec.

Circuit diagram:

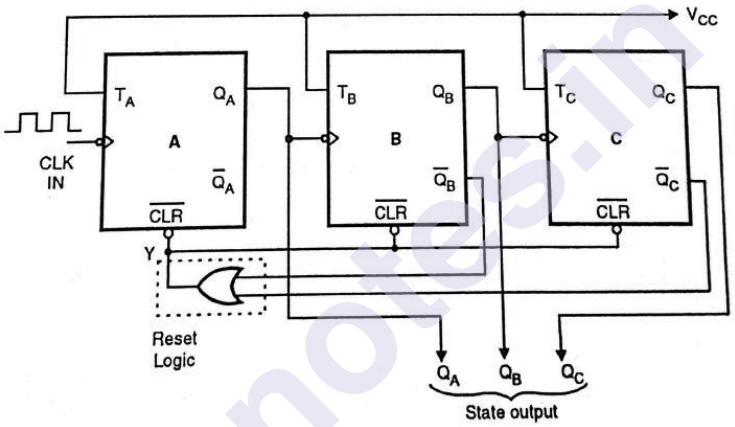


Figure 10.3.2.2(b): mod – 6 ripple counter - circuit diagram

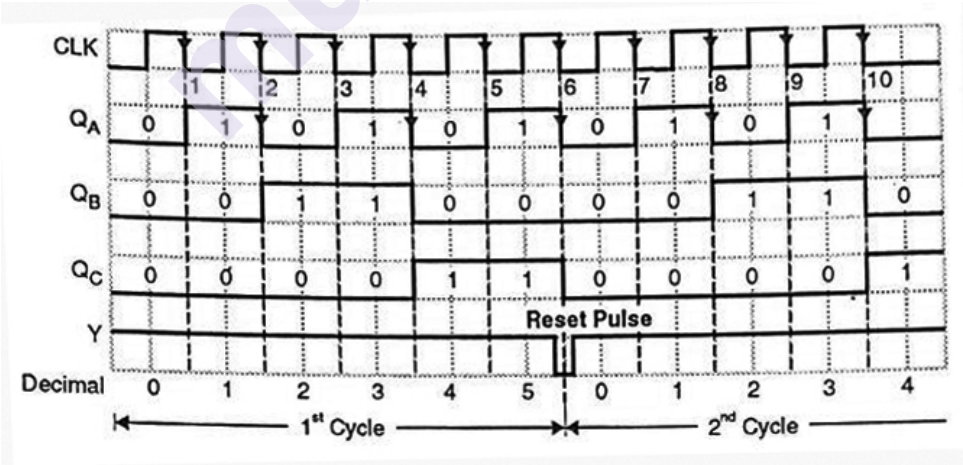


Figure 10.3.2.2(c): mod – 6 ripple counter - wave form

10.3.3 Problems involved in Ripple Counter:

Mainly there are two problems involved in ripple counter.

1) **Glitch operation** – propagation delay of Reset logic.
2) **Propagation delay** – propagation delay of FF

Glitch operation:

In example 1 assumption was made that propagation delay through FF and reset circuit is 0 ɳ sec. But practically this situation does not exist, therefore glitch operation occurs.

Let's consider propagation delay of reset logic. For analysis let's consider example 1 in which states are 0, 1, 2. Redraw the waveforms considering propagation delay of reset CLK.

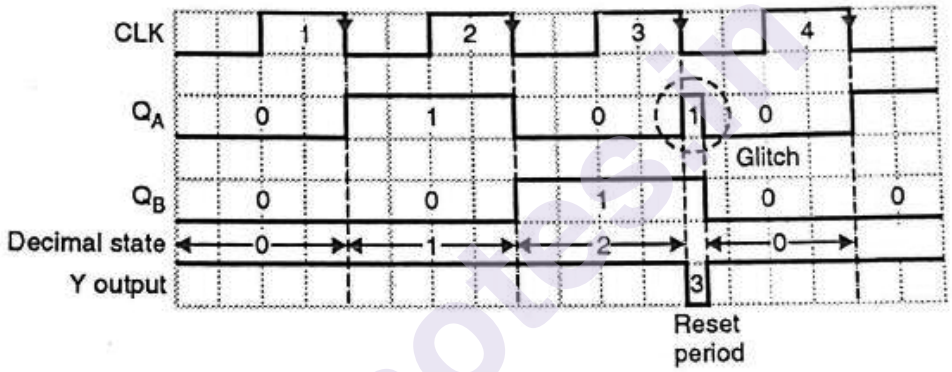


Figure 10.3.3(a): mod – 3 ripple counter - wave form

Working: Refer Figure 12.3.3(a) and example 1 (Q_A = A Q_B = B)

1) Initially all FFs are reset ∴ BA = 00 ∴ Y = 1
2) When 1st CLK edge hits, A will toggle ∴ BA = 01, ∴ Y = 1
3) At 2nd CLK edge, A will toggle from 1 → 0 (negative edge), B also toggles. ∴ BA = 10. ∴ Y = 1
4) When 3rd CLK edge hits, A will toggle from 0 → 1 (positive edge), ∴ B is unchanged. ∴ BA = 11

Reset circuit is designed in such a way that, when 11 occurs Y = 0. But appearing of logic 11 at NAND input, getting settled, then propagates through NAND and appear at output, then resetting the FF will take some time in ɳ sec. During this period BA = 11. This condition is invalid condition and as in Figure 10.3.3(a), Q_A produces unwanted short duration pulse called **glitch**.

Propagation Delay in Ripple counter:

Basic principle of operation of a synchronous counter is, each FF is triggered by the transition at the output of preceding FF. Each FF has internal propagation delay, means second FF will not respond unless and until propagation delay time, after the first FF receives an active clock transition. ∴ Third FF will not respond until a time equal to twice propagation delay (**2 x t_{pd}**) after that clock transition.

Thus propagation delay of FF accumulates so that **N^{th}** cannot change state unless and until, time equals to **N x t_{pd}** after the CLK transition occurs. To understand this refers waveforms in following figure 10.3.3(b).

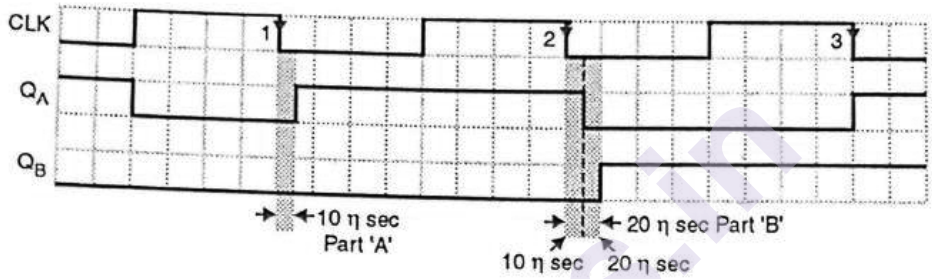


Figure 10.3.3(b): mod – 3 ripple counter: wave form

Refer figure 12.3.3(b), (Q_A = A, Q_B = B) propagation delay of one FF = 10 η sec.

1) Initially both output BA = 00

2) When the first CLK edge hits, A changes from 0 → 1. As shown QA changes states 10 η sec after CLK edge hits. (Part (a)).

3) In Second CLK edge QA changes from 1 → 0. Transition occurs, 10 η sec after CLK edge hits.

   Q_A provides negative transition (1 → 0) to B FF.

   ∴ Q_B changes the state from 0 → 1, 10 η sec after the transition of Q_A.

   ∴ With respect to CLK edge Q_B changes state after 20 η sec. (Part (b)).

   i.e. 2 FF x 10 η sec = 20 η sec

This propagation delay causes limitation on CLK frequency input, so for ripple counter, following equation provides relationship between CLK period, number of FFs used and propagation delay of FF.

Time period of CLK (TCLK) ≥ N x t_{pd}

Where, **t_{pd}=** propagation delay

N = No. of FFs

$$\therefore f_{\max} = \frac{1}{CLK} = \frac{1}{Nt_{pd}}$$

10.3.4 Power on Reset Circuit:

Whenever counter working was discussed, it was assumed that initially everything is reset. Whenever power supply is switched ON; one cannot predict the output state of FF. Therefore clear terminal is provided. But clear terminal requires some external circuitry to force it self 'LOW' so that it can provide Q = 0 (some known state). This external circuitry is nothing but power ON reset circuit. Refer following figure 10.3.4.

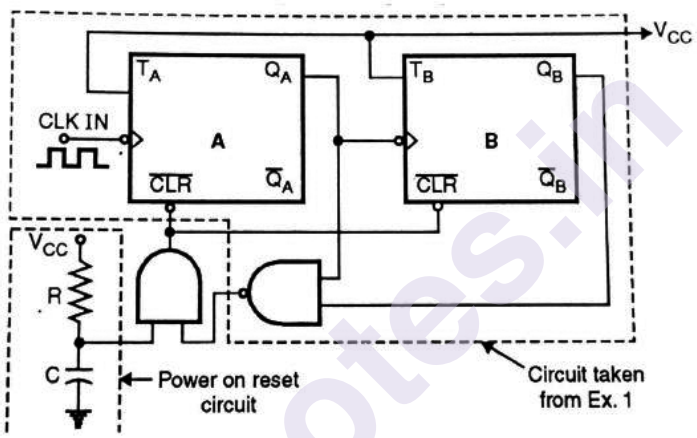


Figure 10.3.4: mod – 3 counter with power ON Reset circuit

As shown in figure 10.3.4 (counter states are 0, 1, 2). In addition one RC n/w with AND gate is used. So either of the input is LOW output of AND gate is LOW, FFs will be cleared

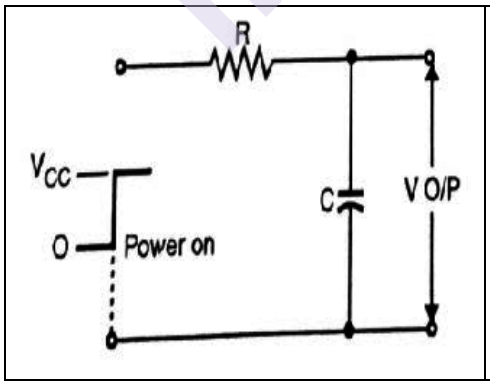


figure 10.3.4(a)

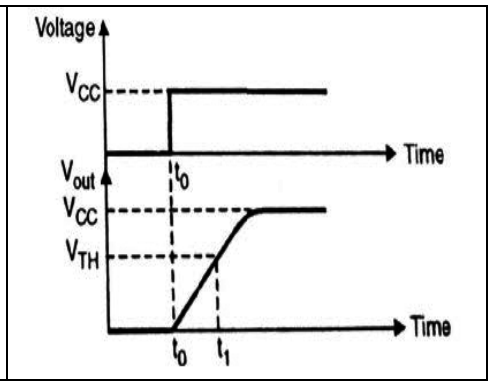


figure 10.3.4(b).

Working:

Refer figure 10.3.4(a) and 10.3.4(b).

When power supply is switched ON, V_{CC} will appear across RC n/w at time t_0. ∴ Capacitor will start charging. The charging slope depends upon RC time constant. When capacitor voltage (V_{output}) reaches to level V_{TH} (Threshold voltage), chip (IC) will consider it HIGH logic. Below threshold it is considered as LOW logic.

∴ For duration $t_1 - t_0$ output of AND gate is '0' ∴ FFs are cleared.
When counter steps through states and AB = 11, output of NAND is equal to 0.
∴ $\overline{CLEAR}.\left(\overline{CLK}\right) = 0$ as AND output is '0' ∴ FF will reset.

10.4 IC 7493 (4-BIT BINARY COUNTER)

Let's study TTC MSI circuit of 4 bit ripple counter implemented in IC 7493.

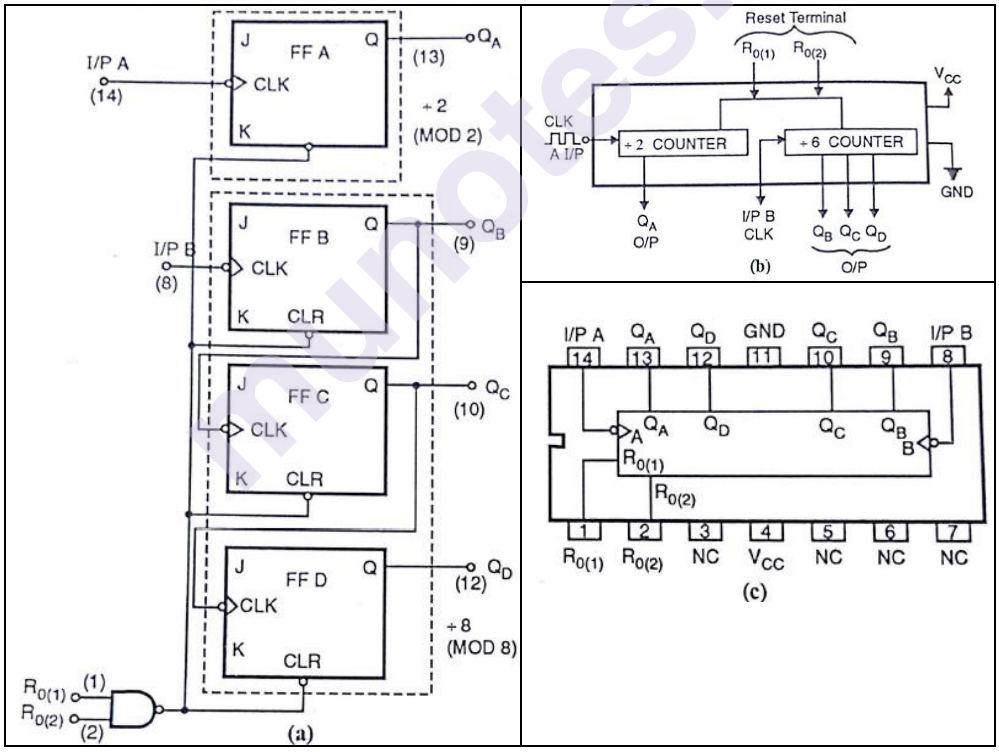


Figure 10.4.1: IC 7493

In **IC 7493,**

Q_D Q_C Q_B Q_A → Binary output of FF.

Input A → CLK IN of FFA

Input B → CLK IN of FFB

$R_{0\ (1)}$, $R_{0\ (2)}$ → Reset pin

When **$R_{0\ (1)} = R_{0\ (2)} = 1$** Reset will occur

Figure 10.4.1 shows logic diagram of IC 7493, It shows that FFA is independent i.e. output is not cascaded to next FF, But FF B, C and D are cascaded. Therefore we have individual, internally two counters, ÷2 (Mod 2) and ÷ 8 (Mod 8). If we combine both counters we get 2 x 8 = 16 state counter of mod 16 counter or ÷ 16 counter.

When we connect Q_A output to input B and CLK IN given to Input A only, it becomes simple 4 bit binary ripple counter. Truth table is as follows

R01/02	CLK	QD	QC	QB	QA
1	X	0	0	0	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	1
0	↓	0	1	1	0
0	↓	0	1	1	1
0	↓	1	0	0	0
0	↓	1	0	0	1
0	↓	1	0	1	0
0	↓	1	0	1	1
0	↓	1	1	0	0
0	↓	1	1	0	1
0	↓	1	1	1	0
0	↓	1	1	1	1

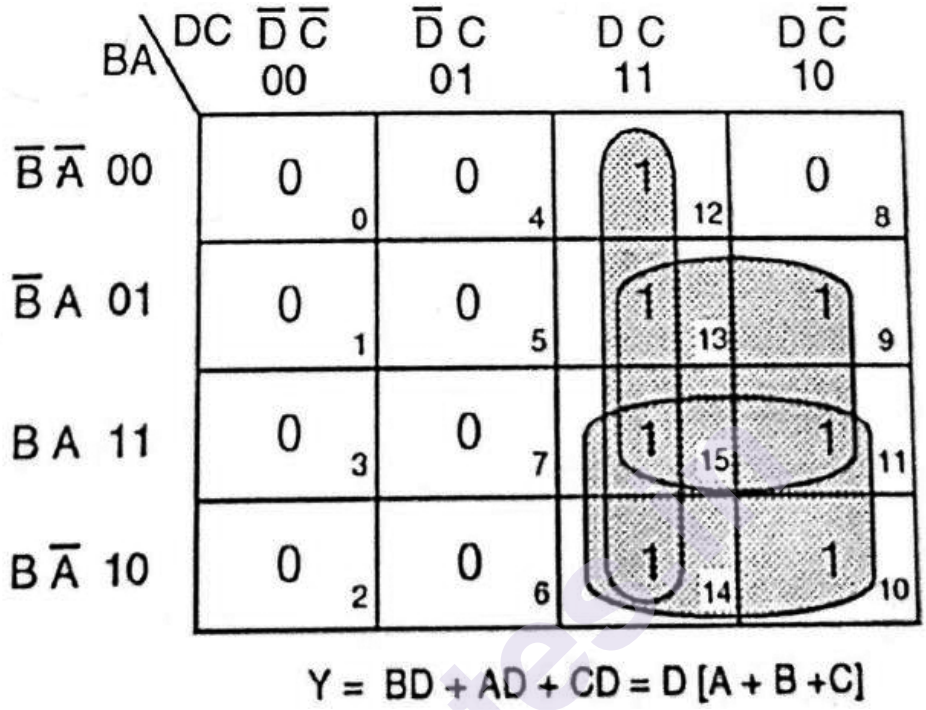
Example 3: Implement ÷ 9 counter using IC 7493.

Solution: Procedure is same as when we design mod – N ripple counter.

1) ÷ 9 means mod - 9, ∴ required states are 9 (0 to 8) ∴ FF used are 4.

2) Reset circuit should generate '0' output for valid states (0 to 8) and '1' output for invalid states (9 to 15), because if $R_{0\ (1)}$ and $R_{0\ (2)}$ both are 1, then only IC 7493 resets.

3) Design reset circuit, Draw K-map.



4) Circuit diagram:

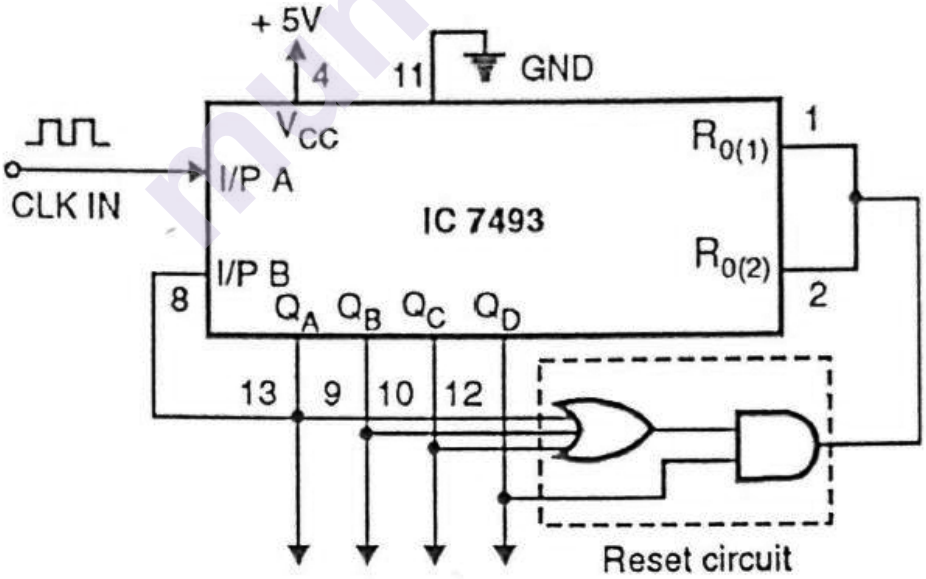


Figure 10.4.2: mod - 9 counter using IC 7493 - circuit diagram

10.5 SYNCHRONOUS COUNTERS

In asynchronous counter mainly two problems were present, glitch and delay in counter. To avoid this, synchronous counter came into picture. Synchronous counter is now most widely used. Generalized block diagram of synchronous counter shown in figure 10.5

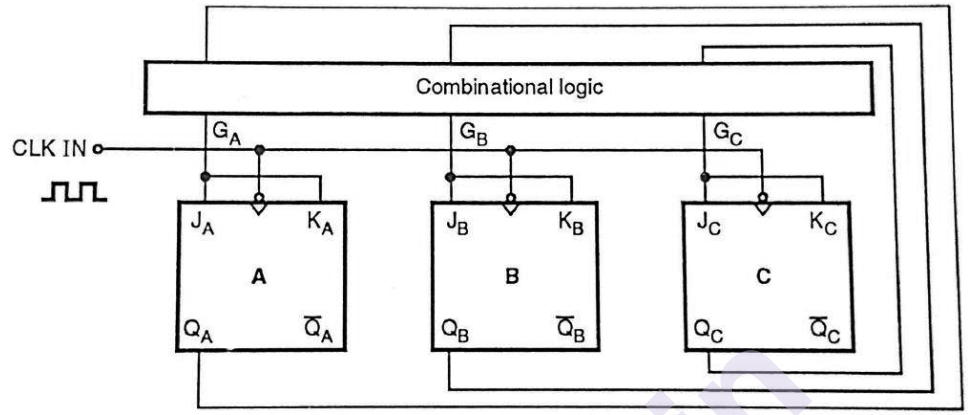


Figure 10.5: Generalized block diagram of synchronous counter

Observation from block diagram of Synchronous counter

1) CLK pin of all FFs are tied together, so that FF changes output in synchronism.

2) O/p Q given to combinational logic circuit. This circuit is designed in such a way that G_A G_B G_C generated from it, applies proper logic to input of FF, so that we get correct next state.

3) O/p Q depends upon previous input of FF, when CLK edge hits. Presently in block diagram, JK is tied together. But this will not be the case every time. JK can be controlled separately by combinational circuit.

10.6 BUSHING

If you observe examples solved up till now, in full sequence '0' is always present and once counter enters in valid state, it will continue the chain unless and until power supply is switched off. To enter in the chain we use power on reset circuit, therefore first state is normally '0'.

Problem:

1) In some of the applications we don't require '0' state at all.

2) Secondly due to power supply fluctuation (glitch of power supply), Electromagnetic inference and RFI (radio frequency interference), it may happen that counter will enter in invalid state.

If due to problem stated above it enters in to invalid condition then what will happen?

Then here your luck factor counts

1. Lock Out State (If Luck Is Bad):

It may happen that counter will lock itself in invalid states only. Let's say in above case counter enters into state 1; because of logic circuit next state happens to be 6. After 6 next state happens to be 1. Then counter will toggle between two states only, as shown in state diagram figure 10.6(a).

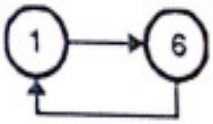
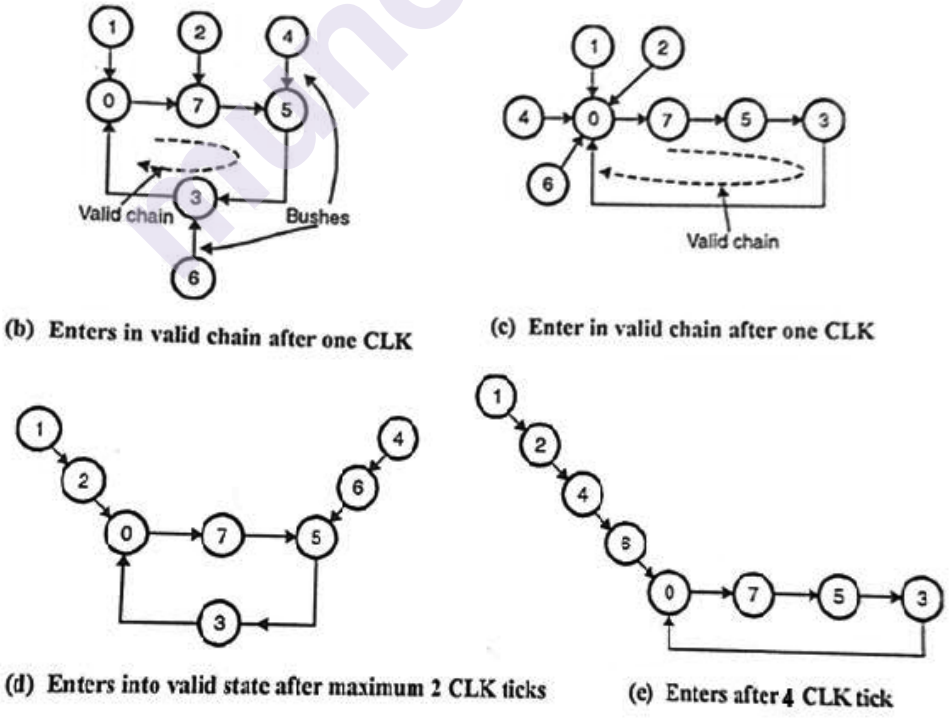


Figure 10.6(a): State diagram

(2) If you are lucky enough after entering into invalid state it may switch over to valid state. But don't try to check your luck. Take precaution. As a precautionary step we implement ***bushing*** to state diagram. Bushing means branches. We are going to branch invalid state in such a way that even though it enters into invalid state after one, two or three CLK ticks it will enter into valid state chain.

Following are some of the examples for implementing bushing in

Example 5



(b) Enters in valid chain after one CLK

(c) Enter in valid chain after one CLK

(d) Enters into valid state after maximum 2 CLK ticks

(e) Enters after 4 CLK tick

Figure 10.6 (b-e): Implementation of Bushing

Thus one can have 'n' number of combinations. Choice depends upon designer

10.7 TYPE T DESIGN

Let's start designing synchronous counter using **T FF**.
Example 4: Design mod – 4 regular sequential synchronous up counter by using T FF.

Solution:

1. No. of states are 4, for mod – 4 counter.

2. Regular means n = N.

3. **No. of FF** required will be $\mathbf{4 \leq 2^m}$ $\therefore$ **m = 2**

4. Sequential and up means **0, 1, 2, 3.**

5. $\therefore$ State design is Figure 10.7(a).

6. **FF A** and **B** are used. $\mathbf{G_A}$ and $\mathbf{G_B}$ are output from combinational circuit to provide input to **T FF**. $\therefore$ Circuit will be as shown in Figure 10.7(b).

7. Design of combinational circuit to generate $\mathbf{G_A}$, $\mathbf{G_B}$

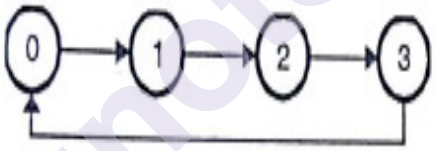


Figure 10.7(a): State diagram

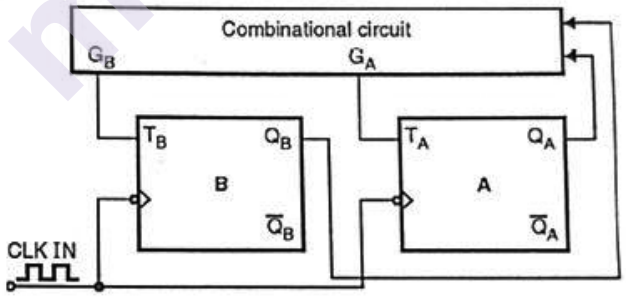


Figure 10.7(b): Circuit diagram

Step 1: For generation of truth table, depending upon number of bits (FF), write down sequentially all states.

In our case **2 bits (FF)** are used, $\therefore$ **state are 4.**

A	**0**	**0**	**1**	**1**	**;**	**A=MSB**
B	**0**	**1**	**0**	**1**	**;**	**B = LSB**
decimal	**0**	**1**	**2**	**3**		

Here $\mathbf{Q_A = A, Q_B = B}$ for simplicity

Step 2: Now below this you write down required output.

A B	0 0 (Prev)	0 1 (Next)	1 0	1 1
G_A G_B				

Step 3: Now start from state of counter. Treat this state as previous state. Take next state of counter from state diagram, and it will be obviously next state.

In our case **1st state = AB = 00 → Previous state.**
2nd state = AB = 01→ Next state.

Step 4: Now refer excitation table of **T FF**, treat change of state for **A** and **B** individually and write down $\mathbf{G_A}$ and $\mathbf{G_B}$ the step is explained as follows:

A	0	0	0	1	1
B	0	1	0	0	1
G_A	0				
G_B	1				

As **A** changes from **0 → 0** ∴ $\mathbf{G_A\ (T_A) = 0}$

B changes from **0 → 1** ∴ $\mathbf{G_B\ (T_B) = 1}$

Step 5: Now treat previously next as previous state. i.e. **AB = 01** which was next state previously, will be **Now** previous state. Refer state diagram and find next state. In our case it will be **AB = 10**. Again the procedure is same as **STEP 4**.

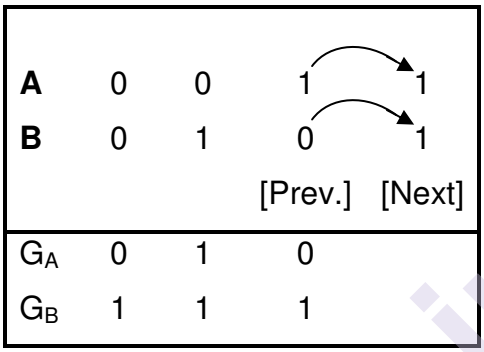
A	0	0	1	1
B	0	1	0	1
		[Prev.]	[Next]	

G_A	0	1
G_B	1	1

As **A** changes from **0 → 1,** ∴ **G_A (T_A) = 1**

B changes from **1 → 0,** ∴ **G_B (T_B) = 1**

Step 6: Now **previous state = AB = 10** and **Next state = AB = 11**, again follow **STEP4**

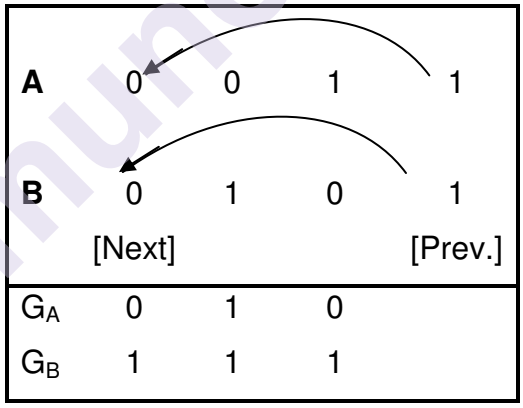


G_A	0	1	0
G_B	1	1	1

As **A** changes from **1 → 1,** ∴ **G_A (T_A) = 0**

B changes from **0 → 1,** ∴ **G_B (T_B) = 1**

Step 7: Finally **previous state = 11** and **next state = 00**. Next we are completing full chain of counter. Follow **step 4.**

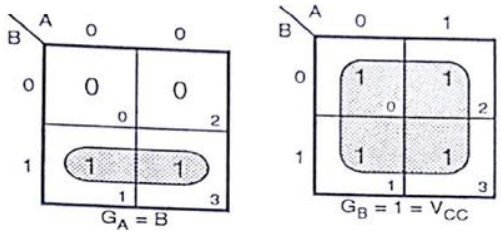


G_A	0	1	0
G_B	1	1	1

As **A** changes from **1→ 0,** ∴ **GA (T_A) = 1**

B changes from **1→ 0,** ∴ **GB (T_B) = 1**

Step 8: After filling truth table, next step is to draw K-Map



Step 9: Draw final circuit diagram.

Already we have drawn half circuit as shown in Figure 10.7(b). Only this is to replace combinational block by directly some connection.

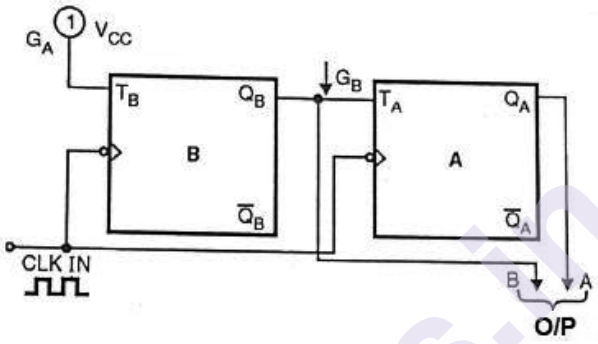


Figure 10.7(c): Final circuit diagram

Let us draw the waveforms and analyze the working of the circuit

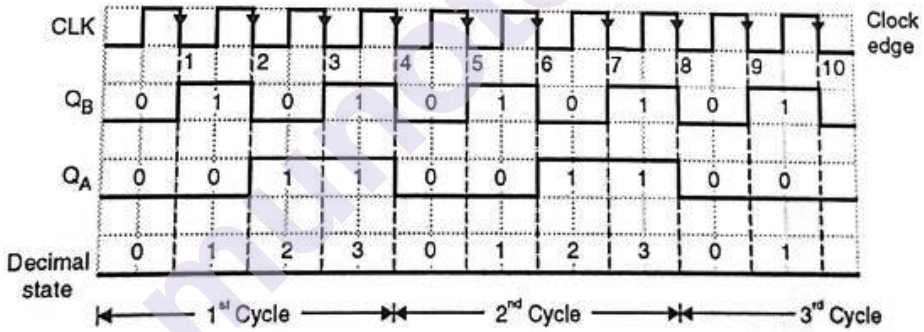


Figure 10.7(d): Waveforms

Working:

1. Initially counter is Reset, ∴ **AB = 00**

2. At 1st CLK edge, $T_A = 0$ as **B = 0**, $T_B = 1$.
∴ **A** remains constant as Q_B toggles from **0 → 1**. ∴ **AB = 01**

3. At 2nd CLK edge, as $T_A = 1$ (because **B = 1**) $T_B = 1$,
∴ **A** and **B** both will toggle, giving ∴ $A_B = 10$

4. At 3rd CLK edge, $T_A = 0$ as **B = 0** and $T_A = 1$,
∴ **A = unchanged**, **B = toggle** ∴ **AB = 11**

5. Finally at 4th CLK edge $T_A = 1$ and $B = 1$, $T_B = 1$
$\therefore$ **A** and **B** will toggle from **1 → 0** $\therefore$ **AB = 00**

Now the point 2, 3, 4 and 5 will continue till CLK is present, and counter generates required states.

If you compare the waveform in Figure 10.7(d) with waveforms of ripple counter. You will find that falling edge is shown only on CLK, not on Q_A or Q_B output, because CLK is synchronized. Counter changes state ONLY when CLK edge hits.

Example 5: Design synchronous, non-sequential counter
Solution:

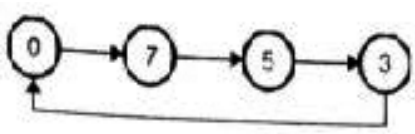


Figure 10.7.1: State diagram

1. Counter is non sequential as well as truncated.

2. While designing non sequential counter, instead of number of state you should consider maximum count of the state. In our case maximum count of the state is 7. To represent 7 in binary we require **3 bits (111)** and therefore **m = 3. As m = 3, n = 2^3 = 8**

3. While writing truth table as told in **Example – 4 step 1**, depending upon number of bits (FF), write down states sequentially. In our case **N = 8** as **m = 3.** So we are supposed to write 0 to 7 sequentially, refer Figure 10.7.1(a) Truth table

4. Truth table.

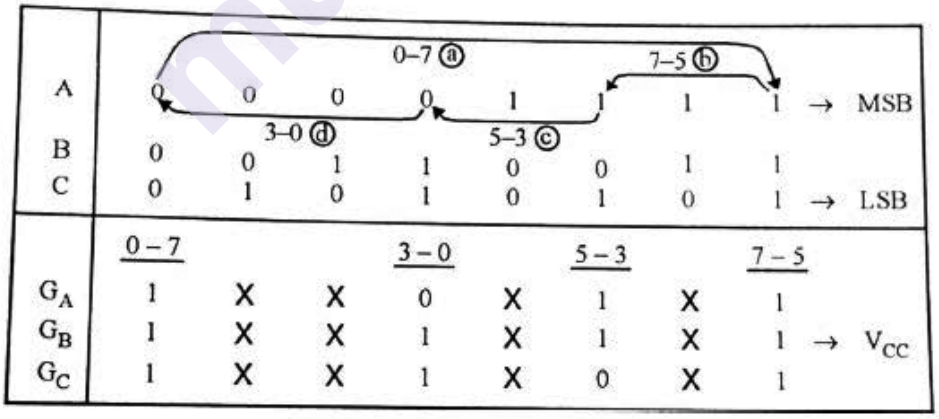
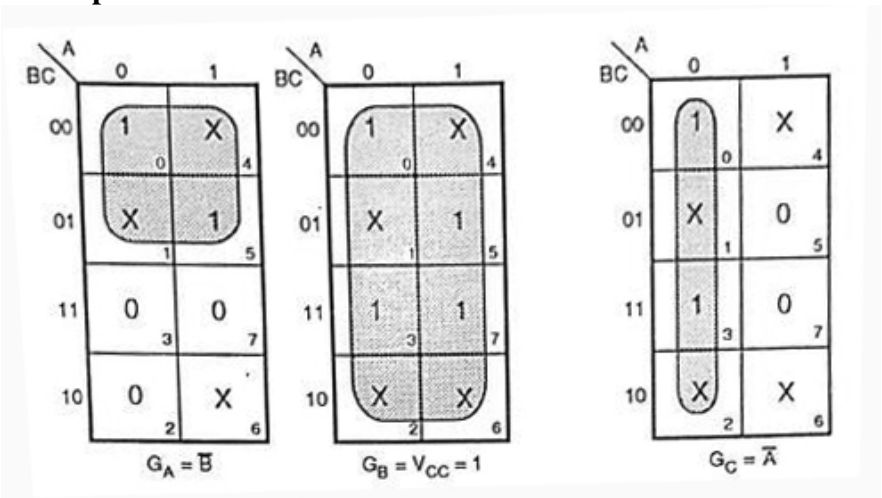


Figure 10.7.1(a): Truth table

5. K – map:



6. Circuit diagram:

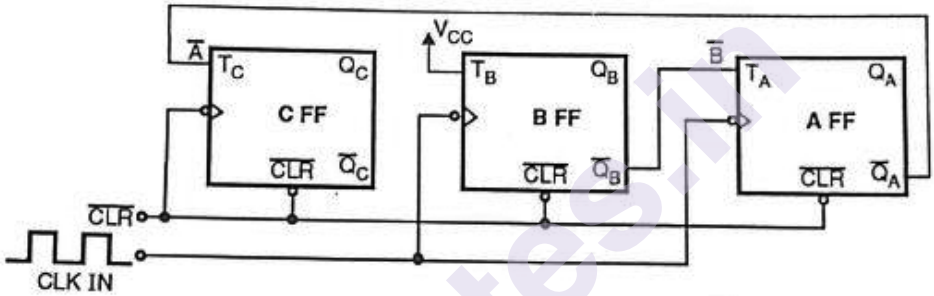


Figure 10.7.1(b): Circuit diagram

Step 1: Write standard table A, B, C and G_A, G_B, G_C.

Step 2: Write 0 – 7 states sequentially under A, B, C.

Step 3: Write down X (don't care) for invalid states. For our case invalid states are 1, 2, 4, 6, ∴ for 1, 2, 6, G_A, G_B, and G_C are X (don't care).

Step 4: Now the steps are standard. Start from 1st state of state diagram i.e. 0. Next state is 7. So the transition is shown in Figure 12.7.1(a) by (a). Previous state of ABC = 000, Next state of ABC = 111.

∴ G_A, G_B, and G_C = 111, written under column '0 – 7' (shown in Figure 10.7.1(a)).

Step 5: 2nd transition is from 7 – 5 (shown by arrow (b)).

ABC changes from 111 to 101 ∴ $G_A = 0$, $G_B = 1$ and $G_C = 0$. Written under column '7 – 5'.

Step 6: 3rd transition is from '5 – 3' (shown by arrow (c)) ABC changes from 101 to 011 ∴ $G_A = 1$, $G_B = 1$ and $G_C = 0$, written under column '5 – 3'.

Step 7: Final transition is from 3 to 0 (3 – 0) shown by arrow (d). ABC changes from 011 to 000 ∴ $G_A = 0$, $G_B = 1$ and $G_C = 1$, written under column '3 – 0'

Step 8: After going through one full cycle draw K – map.

Example 6: Design synchronous counter for state diagram shown in figure 10.7.2

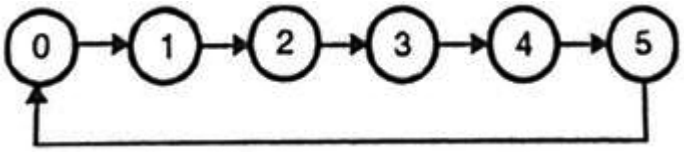


Figure 10.7.2: state diagram

Solution:

1. Counter is sequence but truncated.
2. Number of states **n = 6.** ∴ **$n = 6 \leq 2^m$,** ∴ **m = 3**
   As **$m = 3, 2^3 = 8 = N$** ∴ **n < N (truncated counter)**
3. Up counter (observe from given state diagram)
4. Truth table for designing combinational logic circuit.

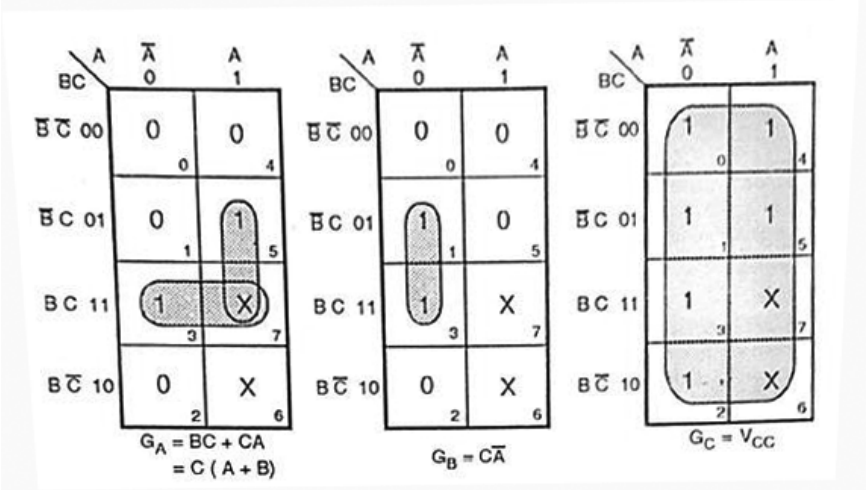
A	0 →	0 →	0 →	0 →	1 →	1	1	1
B	0 →	0 →	1 →	1 →	0 →	0	1	1
C	0 →	1 →	0 →	1 →	0 →	1	0	1
G_A	0	0	0	1	0	1	X	X
G_B	0	1	0	1	0	0	X	X
G_C	1	1	1	1	1	1	X	X

→ V_{cc}

Figure 10.7.2(a): Truth table

As shown in Figure 10.7.2(a) truth table, valid states are only 0, 1, 2, 3, 4 and 5. Therefore, for invalid states i.e. 6 and 7, G_A, G_B and G_C are takes don't care conditions because we know that these states are not going to occur.

5. K – Map for G_A, G_B and G_C:



6. Circuit diagram:

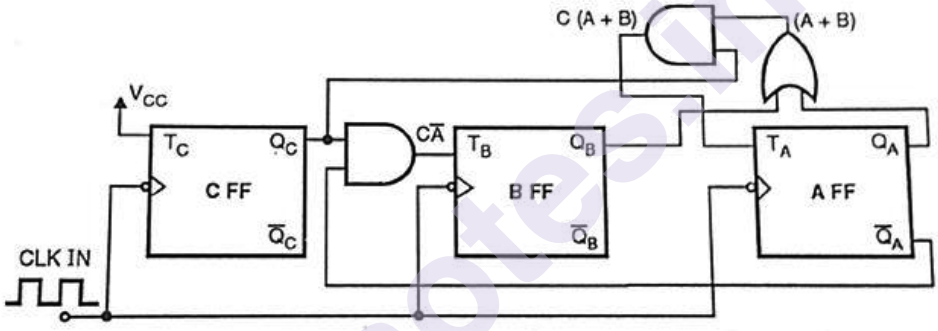


Figure 10.7.2(b): Circuit diagram

7. Waveforms:

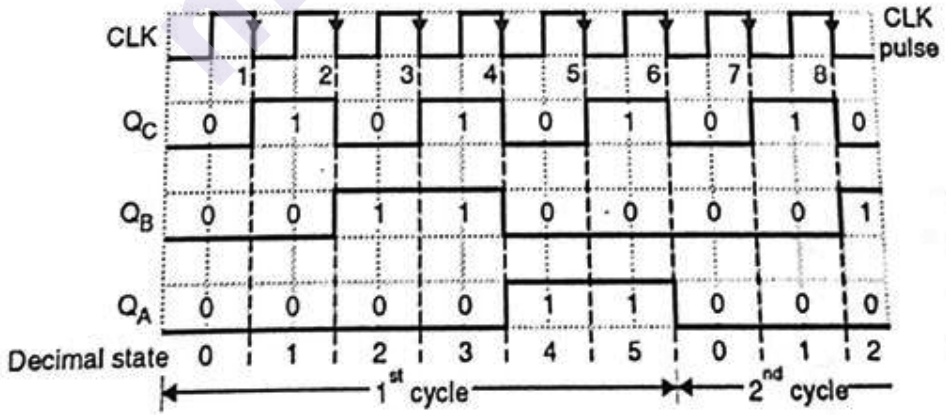


Figure 10.7.2(c): Waveforms.

10.8 TYPE JK DESIGN

Counter design using JK-FF is most widely used. Here J and K terminals are controlled separately. We have already generated excitation table for JK-FF.

As far as counter design goes, method is same as we followed up till now. Figure 10.8 shows generalized block diagram for counter using JK.

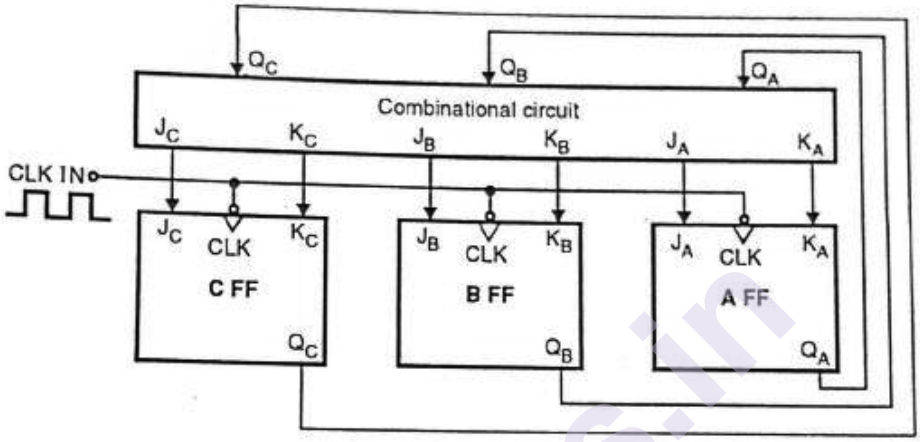


Figure 10.8: Generalized block diagram

As shown in Figure 10.8

1. CLK IN of all FFs is tied together. (synchronous counter)
2. Q_A, Q_B, Q_C output are fed to combinational circuit.
3. Depending upon next state combinational circuit generates proper J and K input for A, B and C FF.

Example 7: Design a modulo – 6 counter using JK FF. Explain its action by writing a truth table and draw waveforms for the outputs of the flip flops.

Solution:

1. State diagram, modulo 6 mean 0 to 5.

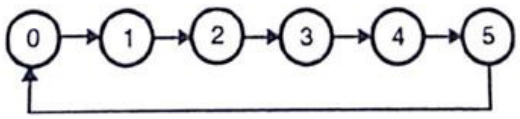


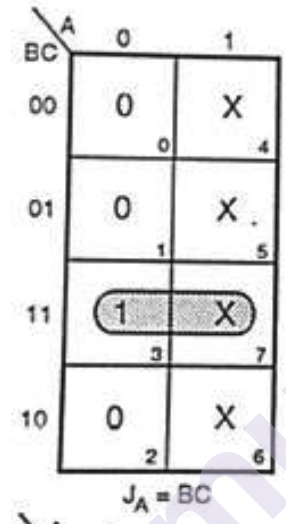
Figure 10.8.1(a): State diagram

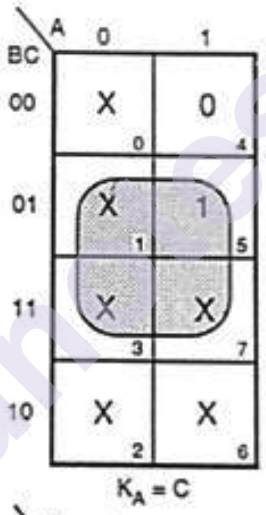
2. Number of FFs required will be 3. $\therefore$ **$N=2^m = 2^3 = 8$.**

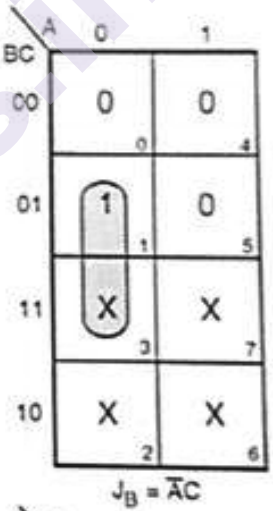
3. Truth table

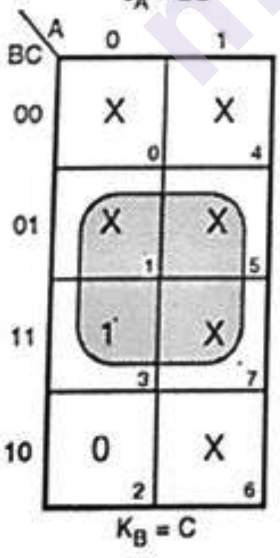
	Invalid states							
A	0 → 0 → 0 → 0 → 1 → 1						1	1
B	0	0	1	1	0	0	1	1
C	0	1	0	1	0	1	0	1
J_A	0	0	0	1	X	X	X	X
K_A	X	X	X	X	0	1	X	X
J_B	0	1	X	X	0	0	X	X
K_B	X	X	0	1	X	X	X	X
J_C	1	X	1	X	1	X	X	X → Vcc
K_C	X	1	X	1	X	1	X	X → Vcc

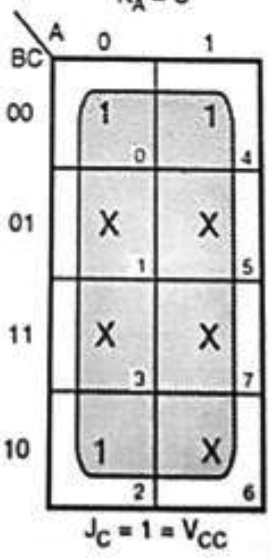
4. **K-Map:**

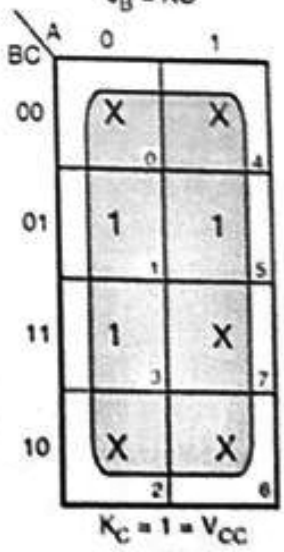












5. Circuit Diagram:

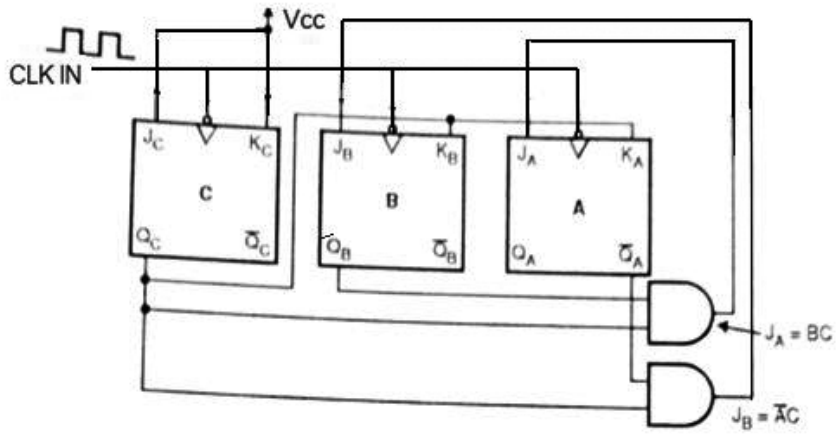


Figure 10.8.1(b): Circuit diagram

6. Waveforms:

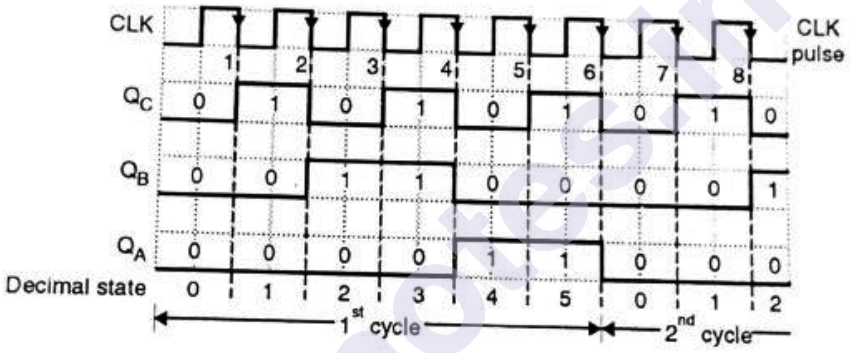


Figure 10.8.1(c): Waveforms

Example 8: Implement synchronous counter using JK FF. for the state diagram shown in Figure 10.8.2

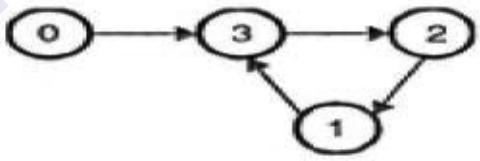


Figure 10.8.2: State diagram

Solution:

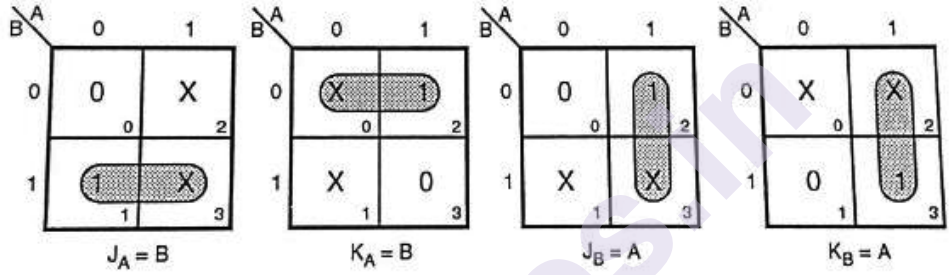
1. Counter is with bushing

2. Number of FFs required will be 2.

3. Truth table:

A	0	0	1	1
B	0	1	0	1
J_A	0	1	X	X
K_A	X	X	1	0
J_B	0	X	1	X
K_B	X	0	X	1

4. K – Map:



5. Circuit diagram:

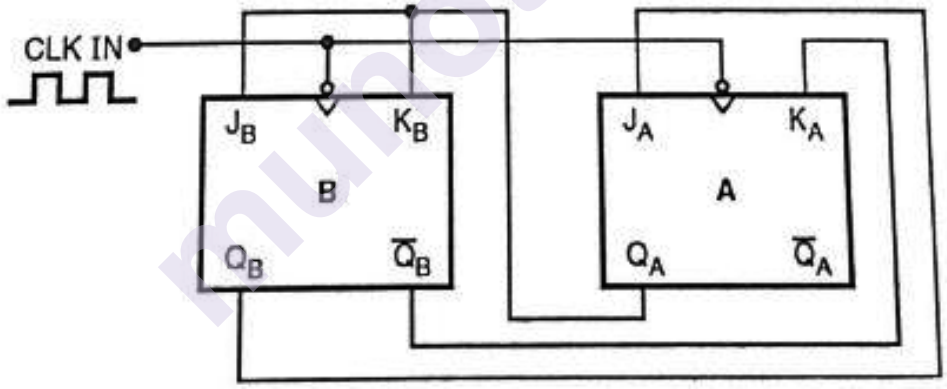


Figure 10.8.2(a): Circuit diagram

10.9 PRESETTABLE COUNTER

As we have discussed previously that sometime counter will not start from '0' state. It will start from some predefined state. Let's say we want to down count from decimal 10. So preset counter to start 10, apply CLK pulses, so it will count down to 9, 8, 7, ... to 0. After that it will again start from 10.

Counters can be preset to any desired starting count either synchronously or asynchronously. This can be done using **PRESET** and **CLEAR** terminals of FF

Figure 10.9 shows **3 bit parallel** presettable up counter.

1. Counter is synchronous.
2. Asynchronous inputs, preset and clear, are controlled through **load** signal.
3. Steps for **'Preset',**
   (a) Apply required count (step) to P_2, P_1, P_0. Let's say P_2, P_1, P_0 = **101**
   (b) Apply **'LOW'** going signal to **'LOAD'** pin
   (c) ∴ Output **of NAND1, NAND3 = 0** and **NAND2 = 1** at the same time **NAND4**,
   NAND6 = 1 and **NAND5 = 0.**
   ∴ **FF B** will be reset and **AC** will be preset
   ∴ **ABC = 101.**
   d) CLK is not going to affect above action.
4. After presenting applies CLK pulse so that counter can start next count.

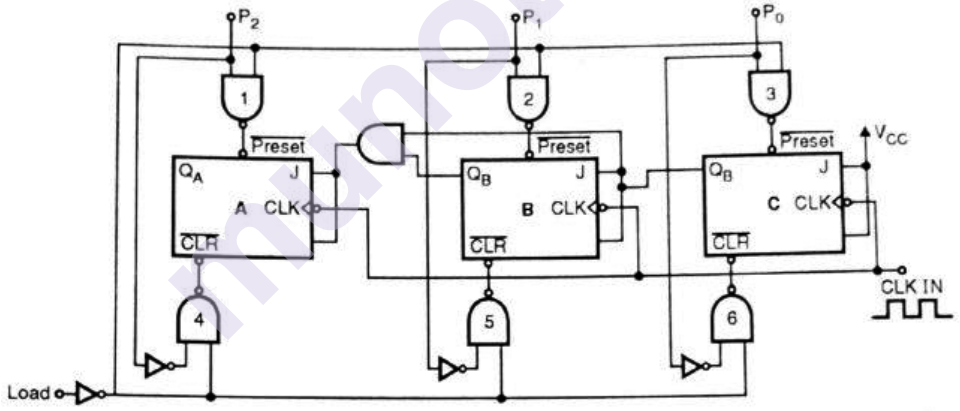


Figure 10.9:3 bit parallel presettable up counter

10.10 IC 7490

IC 7490 is TT1 MSI decade counter. It contains four M/S FF and additional gating to provide divide by 2 counter (Mod-2) and three stage binary counter which provides divide by 5 counter (Mod-5). Figure 10.10 shows block diagram of IC 7490.

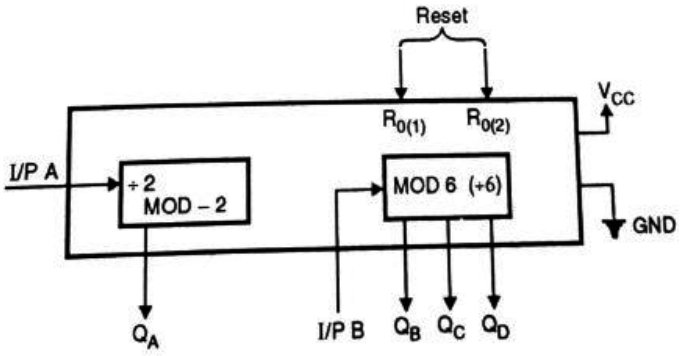


Figure 10.10: Block diagram of IC 7490

7490 have gated zero reset ($\mathbf{R_{0(1)}}$, $\mathbf{R_{0(2)}}$) and gated set to nine inputs $\mathbf{R_{9(1)}}$, $\mathbf{R_{9(2)}}$ for use in BCD nine's complement applications

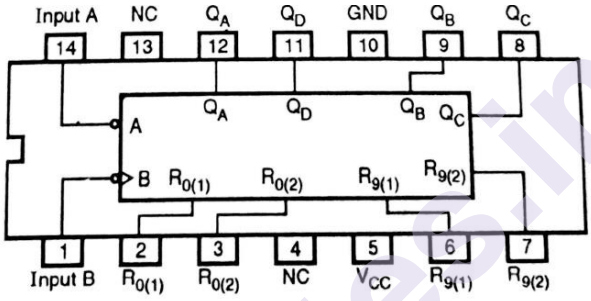


Figure 10.10(a): Pin Configuration - IC 7490

Reset Inputs				Output			
$R_{0(1)}$	$R_{0(2)}$	$R_{9(1)}$	$R_{9(2)}$	Q_D	Q_C	Q_B	Q_A
1	1	0	X	0	0	0	0
1	1	X	0	0	0	0	0
X	X	1	1	1	0	0	1
X	0	X	0	COUNTER			
0	X	0	X	COUNTER			
0	X	X	0	COUNTER			
X	0	0	X	COUNTER			

→ Reset to 0000

→ Set to 1001

If CLK applied it will count or step from previous to next step

Figure 10.10(b): Truth table - IC 7490

As shown in truth table when $\mathbf{R_{0\,(1)} = R_{0\,(2)} = 1}$, **IC 7490 Reset.**
When $\mathbf{R_{9\,(1)} = R_{9\,(2)}}$ **= IC 7490 sets to 1001** (9 decimal).

So one has to provide logic 1 to these input temporarily, set / reset it, and lower the input terminals so IC 7490 can start counting as CLK pulses applied to

Example 9: Implement sequential decade counter using IC 7490 and write counter states

Solution: In IC 7490 mod-2 counter and mod-5 counter is available ∴ 2 x 5 = 10 ∴ gives decade counter.

As mentioned in above example, requirement is sequential decade counter. ∴ Apply CLK IN to INPUT A pin. Output QA is connected to INPUT B i.e. input of mod-5 counter. Output terminals are **Q_D Qc Q_B Q_A**, where **Q_D = MSB**, **Q_B = LSB** connections are shown in Figure 10.10.1(a).

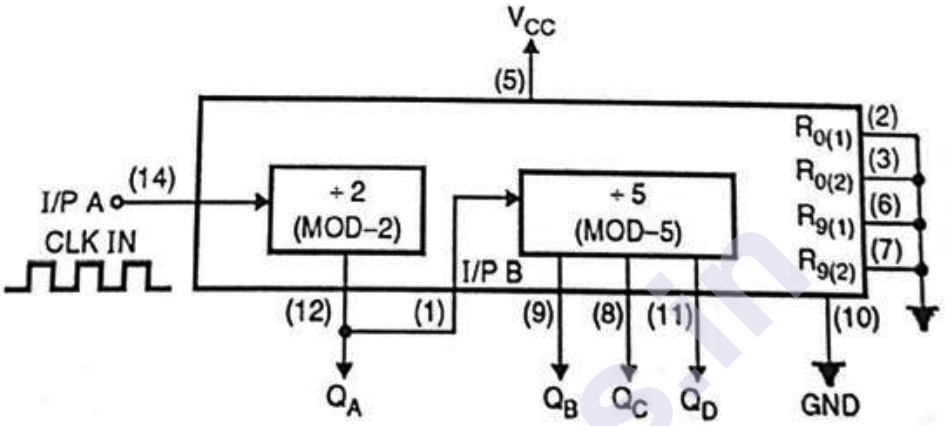


Figure 10.10.1(a): Sequential decade counter

Important point to be remembered is, now at each falling edge of **Q_A** O/P, mod 5 counter will increment. And **Q_A** change at falling edge of CLK input.

The count sequence can be written as,

Count	Q_D	Q_C	Q_B	Q_A	Falling edge of Q_A
0	0	0	0	0	0
1	Change 0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1

Figure 10.10.1(b): Count Sequence of Sequential decade counter

If you observe Figure 10.10.1(b), concentrate on Q_D, Q_B, Q_c and Q_A separately initially Q_D, Q_c, Q_B = 000. When **QA** changed from **1** → Q_D, Q_c, Q_B changed to 001. For 2nd CLK period state of Q_D, Q_c, Q_B remains unchanged.

10.11 IC 7492

IC 7492 is TTL, MSI divide by 12 counter. IC 7492 is having 2 counter inside it. mod 2 (÷ 2) and mod 6 (÷ 6) counter.
∴ Combination of both gives 2 x 6 = 12 ∴ ÷ 12 counter.

Block diagram of IC 7492:

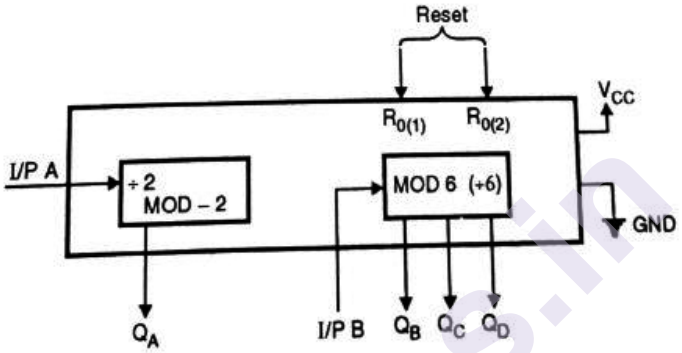


Figure 10.11: Block diagram of IC 7492

Figure 10.11 shows block diagram of IC 7492. For mod 6 counter IC 7492 counts from 0, I, 2, 4, 5, 6, 0. (Observe sequence carefully, state 3 is absent).

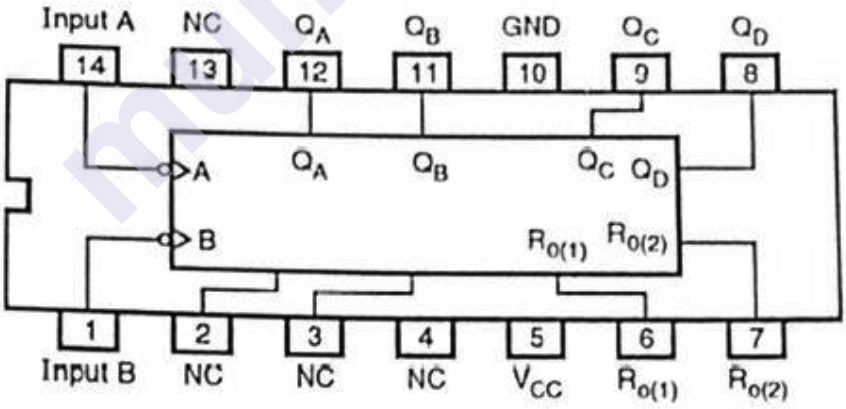


Figure 10.11(a): Pin diagram of IC 7492

Example 10:

(a) Write down truth table for Mod 6 counter in IC 7492.

(b) Write down truth table for Mod 12 counter in IC 7492, if CLK IN is given to Input A and Input **B** Q_A is shorted.

Solution:

(a) Truth table of IC 7492 for mod 6 counter. As we know that internally IC has mod 2 and mod 6 counter. So use only mod 6 counter. CLK IN gives to Input **B** and outputs are Q_D Q_C Q_B only. States are as shown in Figure 10.10.1(a).

CLK edge	Q_D	Q_C	Q_B	Decimal equivalent
1st	0	0	0	0
2nd	0	0	1	1
3rd	0	1	0	2
4th	1	0	0	4
5th	1	0	1	5
6th	1	1	0	6
7th	0	0	0	0

Figure 10.11.1(a): Truth table – IC 7492 mod 6 counter

(b) In this part,

(i) Give CLK IN to Input **A**

(ii) Conned Input **B** to Q_A.

∴ Circuit will be as shown in Figure 10.11.1(b)

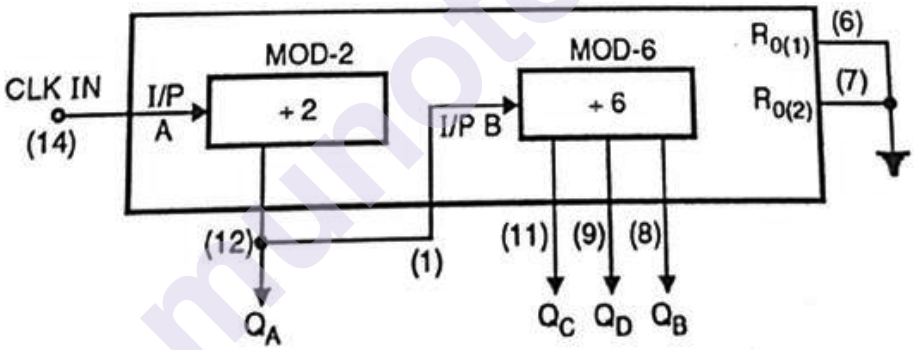


Figure 10.10.1(b): Circuit diagram - IC 7492 mod 6 counter

Point to be remembered.

(1) Q_A toggles at each falling edge of CLK IN.

(2) Q_B Q_C Q_D will change state at each falling edge of Q_A.

Working: Refer Figure 10.11.1(a)

(1) Initially Q_D Q_C Q_B Q_A **(DCBA) = 0000**

(2) At first CLK edge, a changes from **0 → 1** (rising edge.)
∴ **= 000** ∴ **DCBA = 0001.**

(3) At 2nd CLK edge, A. toggles from 1 → 0 (falling edge.)
∴ **DCB** will change from **000 to 001**. (From Figure 10.10.1(a)) **DCBA = 0010.**

(4) 3rd CLK edge, **A** changes from **0 → 1** (rising edge), ∴ **DCBA = 0010.**

(5) 4 th CLK edge, **A** changes from **1 → 0** (falling edge).
∴ **DCB** changes from **001** to **010** ∴ **DCBA = 0100.**

(6) At 5 th CLK edge, **A** changes from 0 → 1 (rising edge),
∴ **DCB = unchanged** ∴. **DCBA = 0101.**

(7) At 6 th CLK edge, **A** changes from **1 → 0** (falling edge),
∴ **DCB** changes from **010** to **100** (Decimal count 011 is not present in **mod 6** refer Figure 10.11.1(a) ∴ **DCBA = 1000**

(8) At 7 th CLK edge, A changes from **0 → 1**,
∴ **DCB = unchanged.** ∴**DCBA = 1001.**

(9) At 8th CLK edge, **A** changes from **1 → 0**, ∴ **DCB** changes from **100 to 101**. ∴ **DCBA = 1010**

(10) At 9 th CLK edge, A changes from **0 → 1**, ∴ **DCB unchanged**, ∴**DCBA = 1011.**

(11) At 10th CLK edge, **A** changes from **1 → 0**, ∴ **DCB** changes from **101** to **110** ∴ **DCBA = 110**

(12) Finally at 11 th CLK edge, A changes from **0 → 1**,
∴ **DCB unchanged**, **DCBA = 1101.**

Truth table is written as follows

CLK edge	Q_D	Q_C	Q_D	Q_A	**DECIMAL EQUIVALENT**
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9

10	1	0	1	0	10
11	1	0	1	1	11
12	1	1	0	0	12
13	1	1	0	1	13
0	0	0	0	0	0

Figure 10.11.1(c): Truth table - IC 7492 mod 12 counter

10.12 SYNCHRONOUS COUNTER ICS

Let's study some synchronous counter ICs

IC 74190:

IC 74190 is UP/DOWN decade counter with preset and ripple clock. It is a reversible BCD (8421) decade counter featuring synchronous counting and asynchronous presetting. Reversible BCD means it can count 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 i.e. up and 9, 8, 7, 6, 5, 4, 3, 2, 1, 0. 9 i.e. down. Asynchronous presetting means parallel input data can be loaded at any instant, it is not dependent upon CLK input.

Pin configuration and logic symbol is as shown in Figure 10.12 (a) & (b)

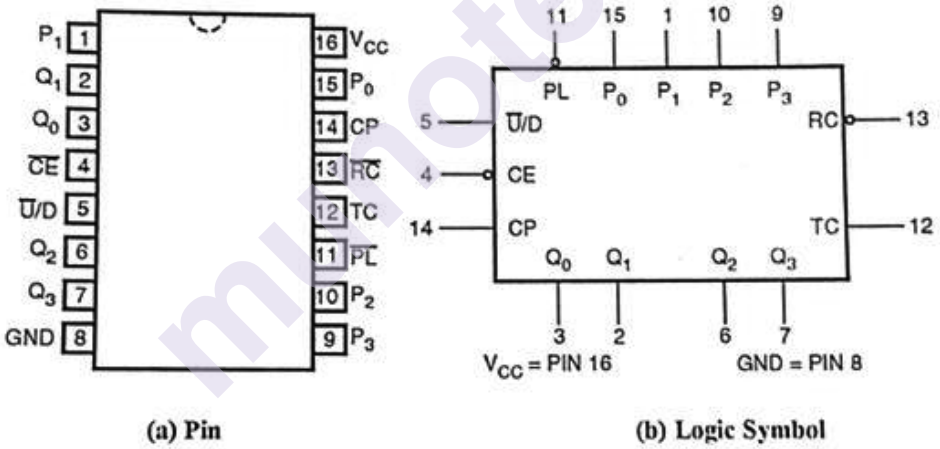


Figure 10.12: Pin configuration and logic symbol – IC 74190

Let us see the Pin names and functions

NAMES	DESCRIPTION
$\overline{CE}$	Count enable input (Active low)
CP	Clock pulse input (Active rising edge)
$P_0 - P_3$	Parallel data input
$\overline{PL}$	Asynchronous parallel load input (Active low)

$\bar{U}/D$	Up / Down count control input 0 – up count 1 – down count
$Q_0 - Q_3$	Flip Flop outputs
$\overline{RC}$	Ripple clock output (Active low)
TC	Terminal count output (Active HIGH)

Basically 74190 is decade counter. Internally four JK FFs are used, output is **4 bit $Q_0 - Q_3$.** It may happen because of supply fluctuation or noise it may enter in invalid state i.e. **10 – 15,** then how 74190 behaves? This point can be explained by using state diagram.

Refer figure 10.12(c)

1) When counter counts up (Shown by dark line

Main loop is → …. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 …

Bushing branches are

(1) 14, 15, 2
(2) 12, 13, 4
(3) 10, 11, 6

Where 2, 4 and 6 are valid states. Worst case counter will take maximum three CLK Cycle to enter in main loop.

2) When counter counts down (shown by dotted line):

Main loop is → …. 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 9 …

Bushing branch is → 15, 14, 13, 12, 11, 9.

Here we have only one branch. In worst case situation counter will take maximum 7

CLK cycles to enter valid states.

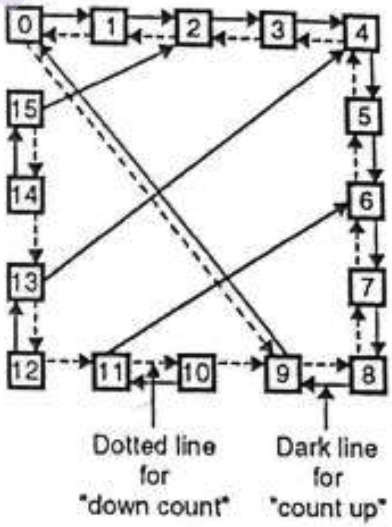


Figure 10.12(c): State diagram

Different Modes of 74190:

74190 has different modes of operation. Modes are selected by $\overline{PL}, \overline{CE}, \overline{U} / D$

Model Select Table

Inputs				Mode
$\overline{PL}$	$\overline{CE}$	$\overline{U}/D$	C/P	
1	0	0	⌐	Count Up
1	0	1	⌐	Count Down
0	X	X	X	Preset(Async.)
1	1	X	X	NC (Hold)

We will study all these modes step by step.

COUNT UP:

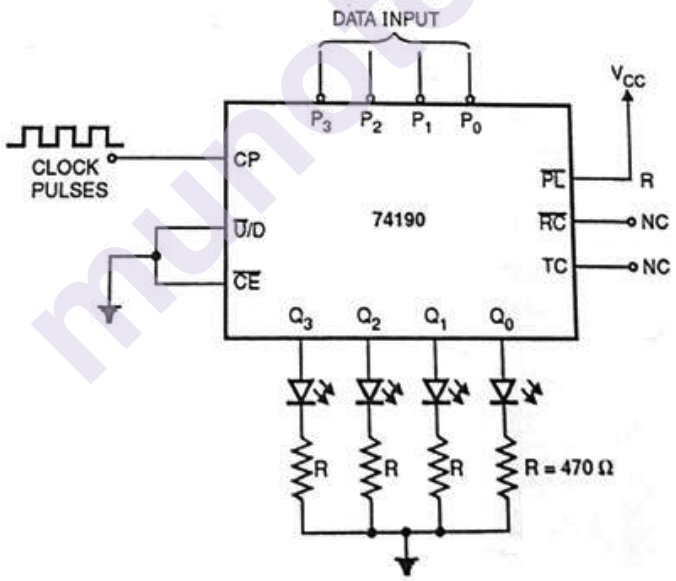


Figure 10.12(d): UP Counter using IC 74190

For counter counting up one has to tie $\overline{PL} = 1$ and $CE = \overline{U}/D = 0$. Apply clock pulse and counter counts up. The circuit is as shown in figure 10.12(d)

COUNT DOWN:

For counter counting down tie $\overline{PL} = 1 = V_{cc}$ and $\overline{U}/D = 1 = V_{cc}, \overline{CE} = 0$, apply clock pulses and counter will start counting down.

The circuit is as shown in figure 10.12(e).

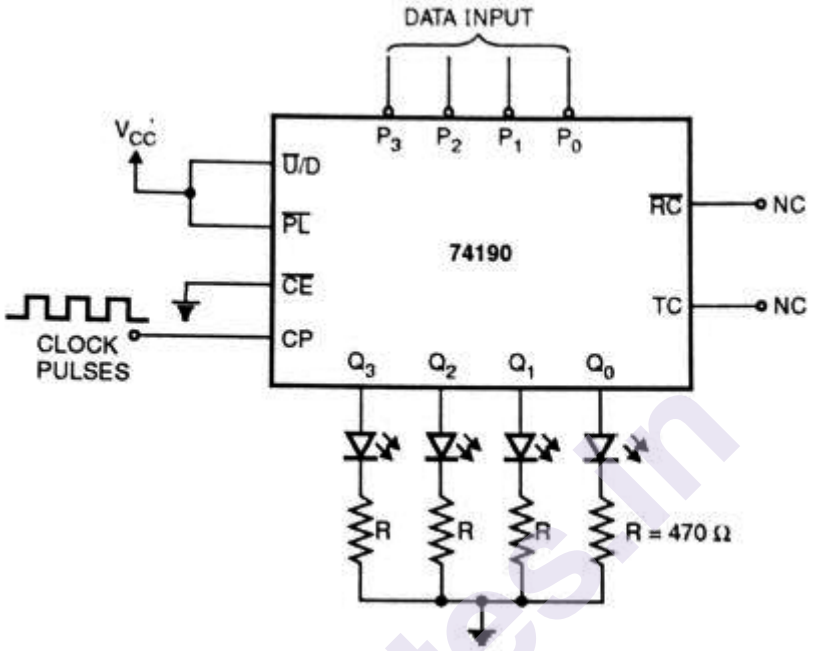


Figure 10.12(e): Down Counter using IC 74190

PRESET MODE:

When you need programmable divider one should PRESET output **Q_0–Q_3** to the required count and start counting.

The figure 10.12(f) shows experimental setup for studying presettable Up/down counter.

Case 1: Presettable Up counter.

Step 1: Close switch **S_2**

Step 2: Apply the count to which you want to PRESET the counter to **P_0 – P_3** input.

Step 3: Close switch **S_1**. Moment switch is closed **P_0 – P_3** contents will appear at **Q_3 – Q_0.** You can see it as LED glows.

Step 4: Apply clock pulses. (The pulses should be of low frequency so that you can see output LED changing)

Step 5: Release (open) **S_1** switch.

Step 6: Now counter will start counting from presetted count. Let's say **$Q_3 - Q_0 = 0101 = (5)_{10}$.**

Then it will count 5, 6, 7, 8, 9 and back to '0'. It won't again preset to zero unless and until **S_1** is closed again.

If we want that after 9, counter again preset to '5' (or presetted count) then we have to add combinational circuit.

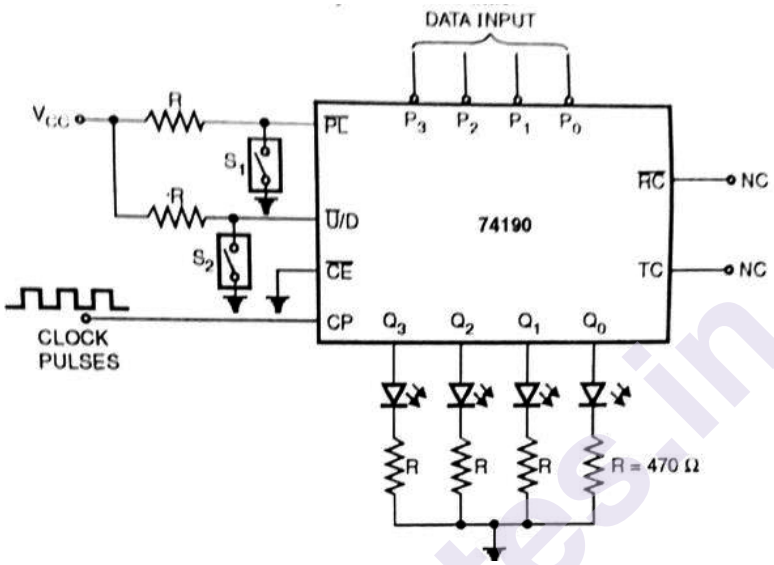


Figure 10.12(f): Presettable up / down counter.

Case 2: Presettable down counter.

Step 1: Open switch **S_2.**

Step 2: Steps written in case 1 from (2) to (5) is same here also.

Step 3: Now counter will, start counting from presetted count. Let's say **$Q_3 - Q_0 = 0101 = (5)_{10}$.**

Then counter will count 5, 4, 3, 2, 1, 0 and back to 9. It won't preset again to 5 in this circuit unless and until **S_1** is closed again.

HOLD:

To HOLD or LATCH the count in the counter make $\overline{PL} = \overline{CE} = 1\overline{U}/D$ 1 /D and CP, both pins are treated as don't care

IC 74191:

IC 74191 is binary up/down counter with preset and ripple clock. If you compare it with 74190, IC 74190 is decade counter, whereas 74191 counts 0 to 15. This IC can count up or down synchronously and has a

synchronous preset. Preset feature allows the chip to be used in dividers. The pin configuration and logic symbol is as shown in figure 10.12.1

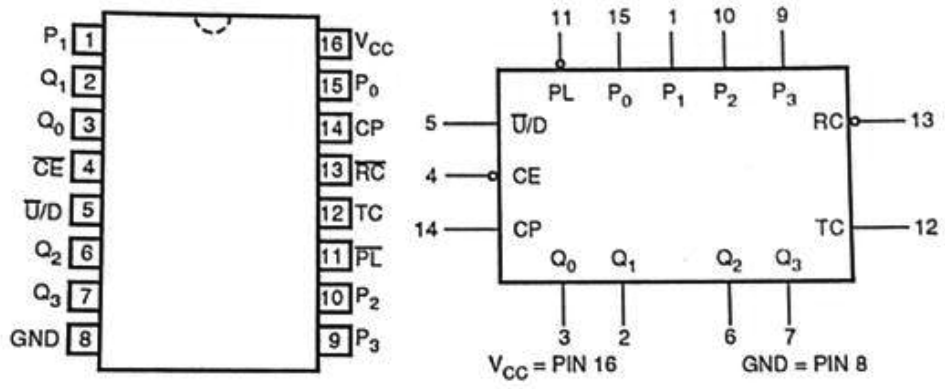


Figure 10.12.1: Pinout and Logic Symbol – IC 74191

Pin names are as follows

Pin Names	Description
$\overline{CE}$	Count enable input (Active low)
CP	Clock pulse input (Active rising edge)
P_0-P_3	Parallel data inputs
$\overline{PL}$	Asynchronous parallel load input (Active low)
$\overline{U}/D$	Up/Down count control input
Q_0-Q_3	Flip Flop outputs
$\overline{RC}$	Ripple clock output (Active low)
TC	Terminal count output (Active HIGH)

State diagram for chip is as shown in Figure 10.12.1(a)

74191 counts up from 0 to 15 (shown by dark line) and also counts down from 15 – 0 (shown by dotted line).

74191 can be operated in different modes

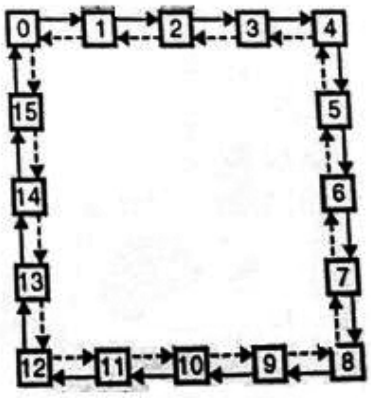


Figure 10.12.1(a): State diagram – IC 74191

Following is the Mode select table for IC 74191

INPUTS				MODE
$\overline{PL}$	$\overline{CE}$	$\overline{U}/D$	CP	Count down
1	0	0	⌐	Count Up
1	0	1	⌐	Preset (Async)
0	X	X	X	No Change (HOLD)
1	1	X	X	

Following is the RC Truth table

Inputs		Output	
$\overline{CE}$	TC	CP	$\overline{RC}$
0	1		
1	X	X	1
X	0	X	1

The modes are same as 74190, therefore the explanation given in 74190 holds for 74191 also.

For cascading or multistage counter, the configuration explained in 74190 holds for 74191 also. RC truth table is also similar.

10.13 ANALYSIS OF COUNTER CIRCUITS

Up till now we have solved some example based on T, D and JK FF design. This part was designing the counter circuit or synthesis of counter, where we were given state diagram and we have to find circuit for the same.

Now it is analysis, i.e. circuit will be given to you and we have to find state diagram, and some time unused state will be given to us and we have to find how counter is going to behave.

The method is very simple. Explained it point by point

(1) Firstly present state will be given to you e.g. say presently counter is in **CBA = 010**, now next states.

By chance if the state is not given, you assume some state. The thumb rule is to start from **000**.

(2) This present state is now previous state for us.

(3) You have to now find out, I/ps to all the FFs, because of this previous state.

(4) After finding out inputs of all FFs. assume that CLK edge hits. Now you already know the input, you know the truth table of each FF. So find out what will be Q output of FF. This 'new' state will be *next state* of the FF.

(5) Now von assume the next state as previous and start from point 3. This chain should be combined unless and until you get *next state,* which already exists previously. For example, We started with slate 4. From 4 next state is 6. From 6 next is 8. From 8 next is 9 and from 9 next state is 4. Means '4' has occurred twice so chain is complete. If you want you can draw state diagram as shown in Figure 10.13(a).

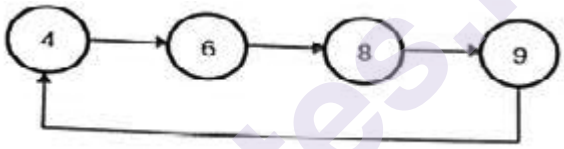


Figure 10.13(a): State diagram

In above example it may happen that you get 6, after 9.6 is already present. But now state diagram will be different. It is as shown in Figure 10.13(b). ∴ 4 is bushing.

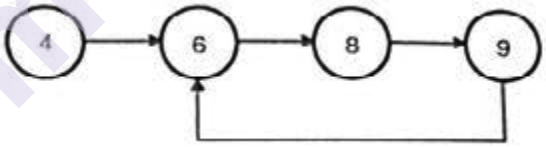


Figure 10.13(b): State diagram

Thus you go on trying different combination and find *next states*. To make analysis simple, all 5 points can be combined in 'one' table. You keep the table standard now onwards.

Previous State				FF inputs				∴ Next state			Decimal Equivalent
A	$\overline{A}$	B	$\overline{B}$	J_A =	K_A =	J_B =	K_B =	A	B	C	

10.14 SUMMARY

In this chapter we have studied in detail about counters which includes asynchronous, synchronous counters, terms related to counters, IC 7493 (4-bit binary counter), Bushing, Type T Design, Type JK Design, Presettable counter , IC 7490, IC 7492, and Synchronous counter ICs with examples.

10.15 REFERENCE FOR FURTHER READING

For further detailed study following are books prescribed by the University

1. Digital Electronics and Logic Design by N. G. Palan
2. Modern Digital Electronics by R. P. Jain
3. Digital Principles and Applications by Malvino and Leach

*****

11

REGISTERS

Unit Structure

11.0 OBJECTIVES

This chapter would make you understand the following concepts

What is register?

- Different types of registers – Parallel and Shift registers.
- Serial Shifting,
- Ring counter, Johnson Counter.
- Applications of Shift Register.
- Pseudo-random binary sequence generator
- IC7495, Analysis of Shift Counter

11.1 INTRODUCTION

Till this point, we have studied different flip-flops and conversion from one to another. Now let's concentrate on application part of flip-flops. The most common use of FF is a simple *Register* (OR a n bit memory storage device). In this section, we will start with basic definition of register and slowly move on to shift register.

Register is a group of memory element that works together as a unit. Register simply stores a binary word. When register accepts parallel data and outputs parallel data, the same is referred as *Parallel register* or *buffer register*. About shift register, it is nothing but memory element with facility of shifting left and right, bit by bit. Before going deep into shift register, let's see structure of buffer register.

11.2 PARALLEL AND SHIFT REGISTERS

11.2.1 Buffer Register:

Buffer register is simplest kind of register and stores digital word. The structure of buffer register, built with the edge triggered D type FF is shown in Figure 11.2

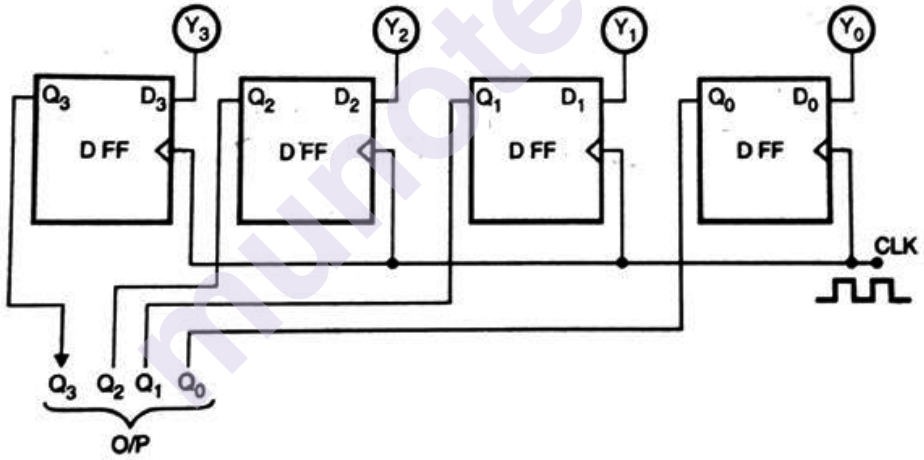


Figure 11.2: Buffer Register

Working:

(1) Initially, consider $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 0$

(2) Apply data to be stored at input terminals $Y_3 - Y_0$. Let's say $Y_3\ Y_2\ Y_1\ Y_0 = 1010$.

(3) Apply clock pulse.

(4) When first clock edge arrives, $Y_3\ Y_2\ Y_1\ Y_0 = Q_3\ Q_2\ Q_1\ Q_0 = 1010$

(5) Even though if now $\mathbf{Y_3\ Y_2\ Y_1\ Y_0 = 0\ 0\ 0\ 0}$, $\mathbf{Q_3\ Q_2\ Q_1\ Q_0 = 1010}$ i.e. latched

Conclusion:

From Fig. 11.2 and WORKING we conclude that,

(1) There must be one FF for each bit in binary number. ∴ For 4 bit we require 4 FFs.

(2) Simultaneously $\mathbf{Y_3 - Y_0}$ was applied and loaded to $\mathbf{Q_3 - Q_0}$ by single clock pulse. i.e. at a time $\mathbf{Q_3}$ **to** $\mathbf{Q_0}$ was loaded.

(3) Therefore, the way we applied input and taken output, is called *parallel in - parallel out* (PIPO). The block schematic representation is shown in Figure 11.2.1.

This is also called as *Parallel Shifting*.

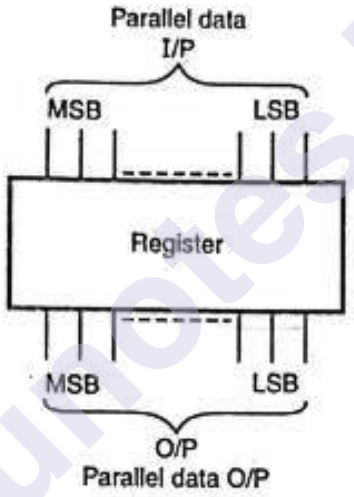


Figure 11.2.1: Parallel in – Parallel out (PIPO)

11.3 SERIAL SHIFTING

One more method of loading or shifting the data is called *Serial Shifting*. What do you mean by serial?

Serial means bit by bit data flow, serially, on single line. Serial shifting has only single bit data line, not 4 or 8 as in parallel loading. The serial shifting can be represented by block schematically, as shown in Figure 11.3.

Single bit data is entered in register and serially single data bit is taken out, through register. A group of flip-flops connected to provide serial out, when serial input is given, is called as *Shift register*

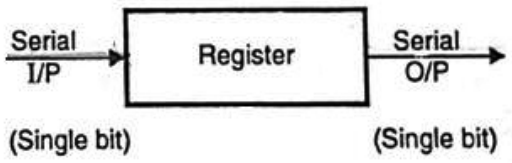


Figure 11.3: Serial Shifting

Basically if you see, loading parallel data is much faster than the serial. Secondly, normally we work with 8 bit, 16 bit, and 32 bit parallel data bits.

So *one can ask, why you think of serial data*? The answer for the question is very simple. If you see, up till now we are looking of loading the registers, but now think of *loading or transmitting data* at distant end. Fig. 11.3.1 shows parallel and serial data transmission.

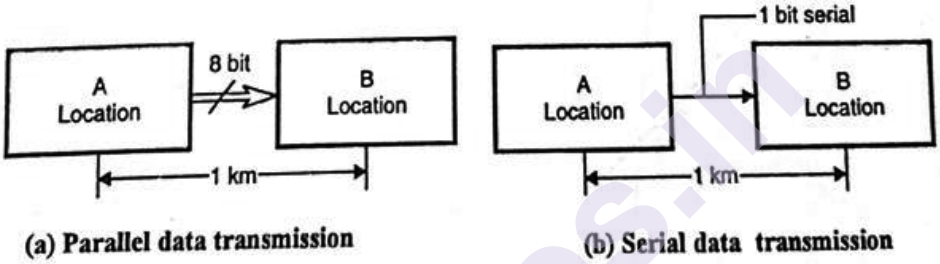


Figure 11.3.1: Parallel and Serial data transmission

As shown in Figure 11.3.1 parallel transmission requires 8 lines, serial requires single line. Now think of distance between A and B location is in meters and kilometers. At such a large distance just increasing length of the wire is not the solutions because now wire is Transmission line, so reflection, impedance matching, power required etc. parameter, we have to consider. Increasing power will increase total power consumption of the system. In parallel, power for 8 lines has to be increased; ∴ power consumed is much, in comparison with single line of serial. ∴ Normally serial transmission is preferred over parallel. At this point you must be eager to know how serial data looks like. It is shown in figure 11.3.2

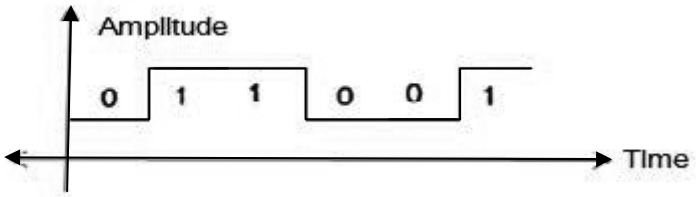


Figure 11.3.2: Format of Serial data

As shown in Figure 11.3.2, data is represented on time axis is bit by bit, not full 8 bit or 16 bit as in parallel. Time 't' for each bit is same. Data stream shown is, ..0, 1, 1, 0, 0, 1,…

Therefore serial communication has become common and most widely used media for communication. But our systems are normally dealing with parallel (8 bit, 16 bit and so on), ∴ one has to convert parallel to serial. This function is also called as **parallel input and serial output (PISO)**.

Serial data is now transrnitted to location B. On receiving side (B side), one has to convert serial data back to parallel. This function of conversion is called **serial in and parallel out (SIPO).** Finally one may require **serial in and serial out (SISO)**. Figure 11.3.3 shows generalized block schematic of four functions.

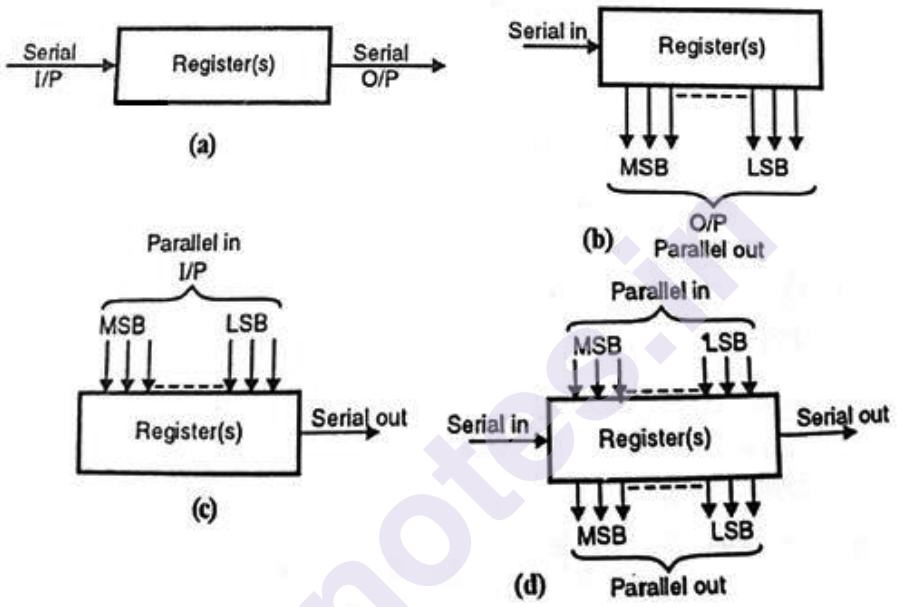


Figure 11.3.3: Generalized block schematic of four functions.

11.4 SERIAL IN – SERIAL OUT (SISO)

In SISO, two main operations are performed, serial left shifting and serial right shifting

Serial Left Shift

Figure 11.4 shows serial operation, built using D Flip-Flop. To know data shifts left we temporarily take output of $\mathbf{Q_2}$ $\mathbf{Q_1}$ and $\mathbf{Q_0}$ also

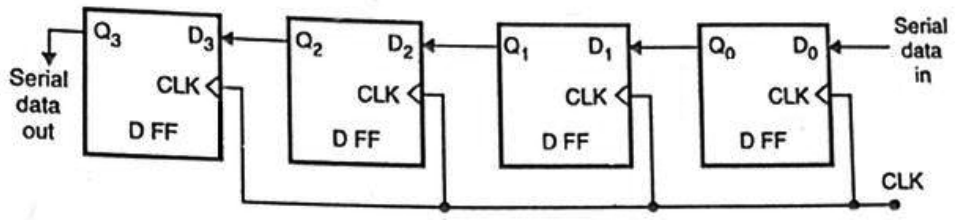


Figure 11.4: Serial operation

Working:

(1) Initially all **FFs** are reset ∴ $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 0$

(2) Make Data **IN = 1.**

(a) At 1st CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 1$ (Refer truth table of D FF).

(b) At 2nd CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 1\ 1$

(c) At 3rd CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 1\ 1\ 1$

(d) At 4th CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 1\ 1\ 1\ 1$

(3) Now the Data **IN = 0.**

(e) At 5th CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 1\ 1\ 1\ 0$

(f) At 6th CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 1\ 1\ 0\ 0$

(g) At 7th CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 1\ 0\ 0\ 0$

(h) At 8th CLK edge, output will be $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 0$

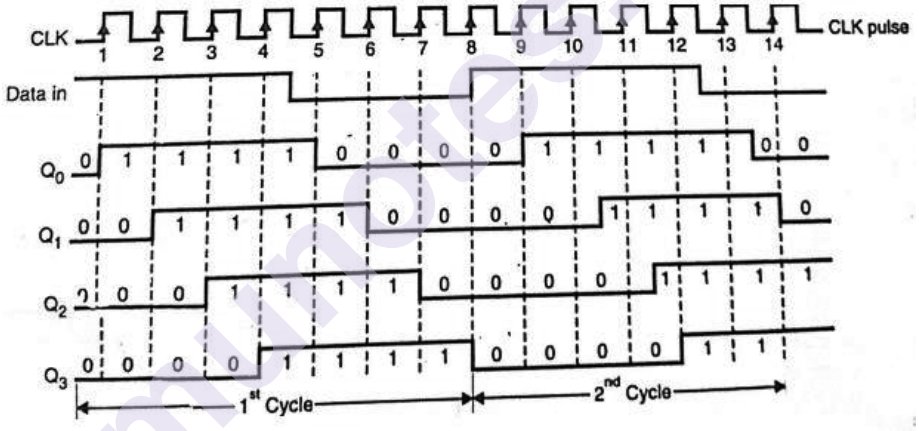


Figure 11.4.1: Waveforms

If you observe, **1 or 0** travels from right hand to left hand, ∴ called *left shifting*. Let's observe waveforms for the same. Waveforms are self-explanatory. Refer Figure 11.4.1.

Shift Right:

Figure 11.4.2 shows serial right shift circuit, built using D FF.

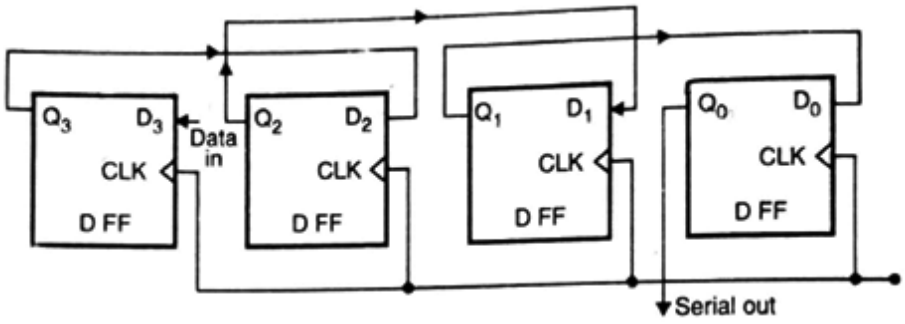


Figure 11.4.2: Serial right shift circuit

Working:

(1) Initially all **FFs** are cleared ∴ **Q_3 Q_2 Q_1 Q_0 = 0 0 0 0**

(2) Make Data **In = 1**

(a) 1st CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 1 0 0 0**

(b) 2nd CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 1 1 0 0**

(c) 3rd CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 1 1 1 0**

(d) 4th CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 1 1 1 1**

(3) Now apply Data **In = 0.**

(e) 5th CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 0 1 1 1**

(f) 6th CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 0 0 1 1**

(g) 7th CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 0 0 0 1**

(h) 8th CLK edge, output will be, **Q_3 Q_2 Q_1 Q_0 = 0 0 0 0**

Thus **1** and **0**, travel from left to right, ∴ right shifting.

11.5 Serial in - Parallel out

In this function, data is entered serially and taken out in parallel. To take data parallel out, it is necessary to have all data bits available as output, at the same time and therefore output of each FF is taken.

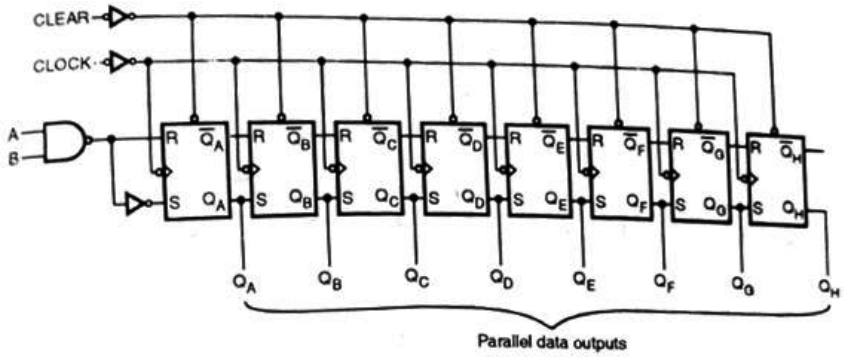


Figure 11.5: Circuit diagram Serial in – Parallel out

If you observe the diagram carefully you will find that,

(1) Clock of all FFs are tied together, therefore all FF changes state at same time.

(2) Asynchronous input, CLEAR, is given to all FF. So that when **CLEAR = 0**, all FFs are resetted, ∴ output **Q = 0000 0000.**

(3) **A** and **B** are data inputs, either of the terminal can be used as control line. When **A = 0**, irrespective of **B**, input to FF will be **0**. At each CLOCK edge **0** travels from left to right and after 8 CLOCK pulses. **Q output = 0000 0000**. It means if **B = Data In**, data will get inhibited.
When **A = 1**, NAND gates enabled and data **Input B** passes through NAND gate inverted. The input data shifted serially in the register.

(4) Output of all FFs are taken as final output. O/P is **Q,** not. Output appears at a time, ∴ It is parallel output.

(5) Data inputs may be changed while CLOCK is either low or high.

11.6 PARALLEL IN – PARALLEL OUT

This type we have already studied in **section 11.2.1, buffer register.** In that Y_3 - Y_0 was inputted parallel and Q_3 - Q_0 taken out parallel.

11.7 RING COUNTER

Refer **figure 11.4** in that circuit if Q_3 is connected to Serial Data In. then circuit is going to circulate injected pulse. The way circuit behaves, it is named as '**Ring Counter'.**

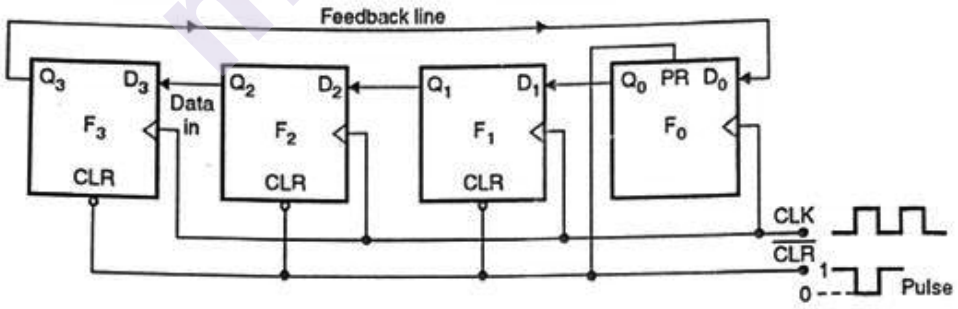


Figure 11.7: Ring counter

Working:

Basically ring counter resembles shift register because the bits are shifted left one position per positive clock edge. Only change is *feedback line*. i.e. output of Q_3 given to Q_0.

(1) Initially CLOCK goes low then back to high. F_3, F_2, F_1 will be reset and F_0 is preset giving ∴ output $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 1$.

(2) At 1st positive **CLOCK edge, MSB** moves to **LSB**, $Q_3 \rightarrow Q_0$, $Q_2 \rightarrow Q_3$, $Q_1 \rightarrow Q_2$ and $Q_0 \rightarrow Q_1$ ∴ output $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 1\ 0$

(3) At 2nd positive **CLOCK edge**, with respect to second point, bits will shift left. ∴ Output $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 1\ 0\ 0$.

(4) At 3rd positive **CLOCK edge**, with respect to 3rd point, bits will shift left. ∴ Output $Q_3\ Q_2\ Q_1\ Q_0 = 1\ 0\ 0\ 0$

(5) At 4th positive **CLOCK edge**, with respect to 4th point, bits will shift left. ∴ Output $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 1$

Thus presetted '1' follows circular path or makes ring. ∴ It is called as ring counter. Observe waveform for the same. Refer Fig. 11.7 (a)

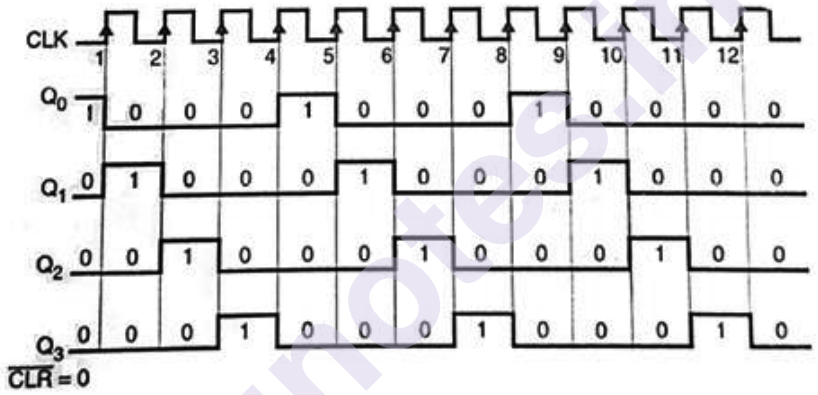


Figure 11.7 (a): Waveform – Ring counter

Conclusion:

(1) Number of **FFs** used are **4.**

(2) Number of states for ring counter is also **4**. i.e. **0001, 0010, 0100, 1000** and chain repeats.

(3) Number of 1's s is single.

Let's write output **Q** for **8 bit**. So for 8 bit, 8 FFs will be used.

Initially Q = 0000 0010

1st clock edges, **Q = 0000 0010**

2nd clock edges, **Q = 0000 0100**

3rd clock edges, **Q = 0000 1000**

4th clock edges, **Q = 0001 0000**

5th clock edges, **Q = 0010 0000**

6th clock edges, **Q = 0100 0000**

7th clock edges, **Q = 1000 0000**

8th clock edges, **Q = 0000 0001** → Repeat,

If you require you can also take output from $\overline{Q}$, ∴ For 4 bit Refer Figure 11.7

Initially when $\overline{CLOCK}$ = 0, Q = 0001 ∴ $\overline{Q}$ = 1110

1st CLOCK edge $\overline{Q}$ = **1101**

2nd CLOCK edge $\overline{Q}$ = **1011**

3rd CLOCK edge $\overline{Q}$ = **0111**

4th CLOCK edge $\overline{Q}$ = **1110**

Application:

Ring counters are invaluable when it is necessary to control sequence of operation. Mainly ring counters are used in microprocessor.

Relationship of number of states and FFs:

In *ring counter number of FFs* and *number of states* are equal. Ring counter using JK FF is shown in Figure 11.7.1.

The working of circuit is same as that of, when D FF is used

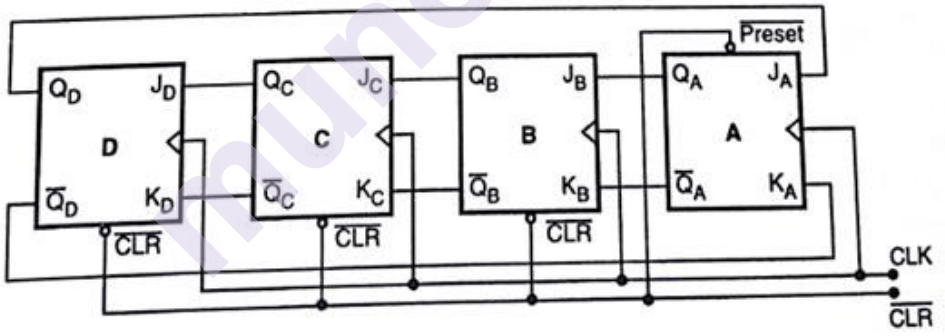


Figure 11.7.1: Ring counter using JK FF

11.8 JOHNSON COUNTER (TWISTED RING COUNTER)

In ring counter output shift register was fed back to input of first FF, that technique was referred as ***direct feedback***. But if outputs of last FF are crossed and then connected to inputs of first FF. the technique is

called *inverse feedback.* The counter we get from this technique is called *Johnson counter* or *twisted ring counter*

Circuit:

Let's draw circuit of Johnson counter using positive edge triggered **JK FF**.

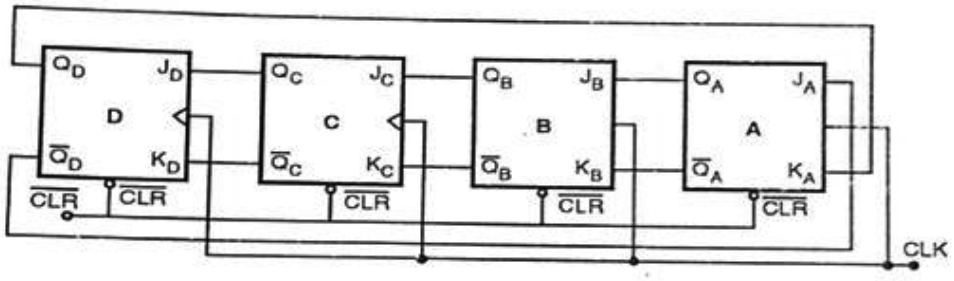


Figure 11.8: 4 bit Johnson counter

CLOCK	Q_D	Q_C	Q_B	Q_A	State	Decimal Equivalent
Initially	0	0	0	0	1	0
1	0	0	0	1	1	1
2	0	0	1	1	3	3
3	0	1	1	1	4	7
4	1	1	1	1	5	15
5	1	1	1	0	6	14
6	1	1	0	0	7	12
7	1	0	0	0	8	8
8	0	0	0	0	1	0

Figure 11.8(a): Truth table

Working:

Refer Figures 11.8 and 11.8(a).

(1) Initially $\overline{CLOCK}$ is made low and then high after some time.
∴ all FFs will be cleared and output $Q_D\ Q_C\ Q_B\ Q_A$ = **DCBA = 0000.**

(2) Now Inputs $J_D = J_C = J_B = 0$, $K_D = K_C = K_B = 1$ and $J_A = 1$, $K_A = 0$. So CLOCK edge hits. A will change from **0** ∴ **1 BCD** are unchanged ∴ **DCBA = 0001.**

(3) Inputs $J_D = J_C = 0$, $K_D = K_C = 1$. But $J_A = J_B = 1$, $K_A = K_B = 0$. ∴ the moment CLOCK edge hits **D** and **C** are unchanged. **A** is also unchanged. But **B** changes from **0** ∴ **1**. ∴ **DCBA = 0011.**

(4) Inputs $J_D = 0$ $K_D = 0$, $J_A = J_B = J_C = 1$ and $K_A = K_B = K_C = 0$ and ∴ the CLOCK edge will change output **C** from **0** ∴ **1**, **A, B** and **D** are unchanged. ∴ **DCBA = 0111**.

(5) Inputs $J_D = J_C = J_B = J_A$ and $K_D = K_C = K_B = K_A = 0$ ∴ next CLOCK edge changes **D** from **0** ∴ **1**, ∴ **DCBA = 1111**

(6) Because of change of state of **D** from **0** ∴ **1**. Now I/P to $J_A = 0$ and $K_A = 1$, $J_D = J_C = J_B = 1$ and $K_D = K_C = K_B = 0$. ∴ next CLOCK edge changes **A** from **1** ∴ **0** ∴ DCBA = 1110.

(7) Now input to $J_A = J_B = 0$ and $K_A = K_B = 1$ and $J_D = J_C = 1$ and $K_D = K_C = 1$ ∴ next CLOCK edge changes **B** from **1** ∴ **0**. ∴ **DCBA = 1100** means **A, D, C** are unchanged.

(8) At this stage input to $J_A = J_B = J_C = 0$ and $K_A = K_B = K_C = 1$, $J_D = 1$ and $K_D = 0$. ∴ next CLOCK edge keeps **A**, **B** and **D** unchanged, **C** changes from **1 0**. ∴ **DCBA = 1000.**

(9) Finally input to $J_A = J_B = J_C = J_D = 0$ and $K_A = K_B = K_C = K_D = 1$ ∴ at next CLOCK edge **D** changes from **1** ∴ **0.** ∴ **DCBA = 0000**, Now cycle will repeat from **point 2** onwards.

As we have seen in working of Johnson counter and observing truth table, we conclude that number of states of Johnson counter are double of number of FFs. Therefore for 4 FFs states will be 8. 5 FFs states will be 10 and for 8 FFs states will be 16.

Johnson counter using D FF:

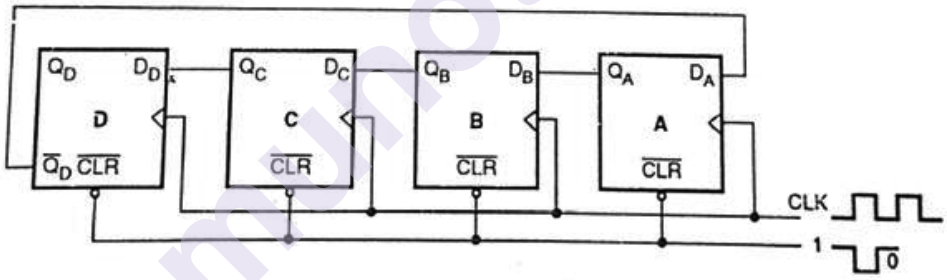


Figure 11.8(b): Johnson counter using D FF

The working of the circuit is same as what we have seen

11.9 APPLICATIONS OF SHIFT REGISTERS

Shift registers can be used for:

(1) Generating sequence (pulse generator)

(2) Counters

(3) Random bit generator.

(1) Sequence Generator (Pulse Generator):

Block schematic structure is shown in Figure 11.9

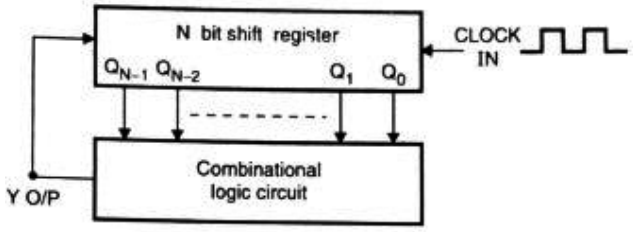


Figure 11.9: Block diagram – Sequence generator

Basically shift register sequence generator contains basic two blocks:

(1) 'N' bit shift register (SIPO type).

(2) Combination logic circuit.

Combinational logic circuit has input $\mathbf{Q_{N-1}}$ to $\mathbf{Q_0}$. From this it will decide, what should be the next serial input for shift register block, should be generated, so that new state or sequence will appear. Now we will take an example to design sequence generator.

Example 1:

The following pulse train is to be generated by using a shift register. Explain the steps in designing and draw logic diagram. pulse train ... 1011010110 ...

Solution:

(1) Observe the pulse train carefully and you will find that main pulse train, which is repeating, is 10110.

i.e ...$\underbrace{10110}_{Maim}\underbrace{10110}_{Re peating}$...

(2) Calculate number of distinct timing intervals. In short you calculate number of bits of pulse train. In our case it is

5 (1 0 1 1 0)

1 2 3 4 5…..

(3) Number of FFs can be given by

$s \leq 2^N - 1$

Where N = Number of FFs.

S = Number of distinct timing intervals.

$\therefore$ $S \leq 2^N - 1$

$\therefore$ $6 \leq 2^N$ $\therefore$ $N = 3$

$\therefore$ 3 FFs are required.

(4) Preparing state table for sequence generator:

Make table, containing number of states, decimal equivalent and 3 output of FF (N = 3)

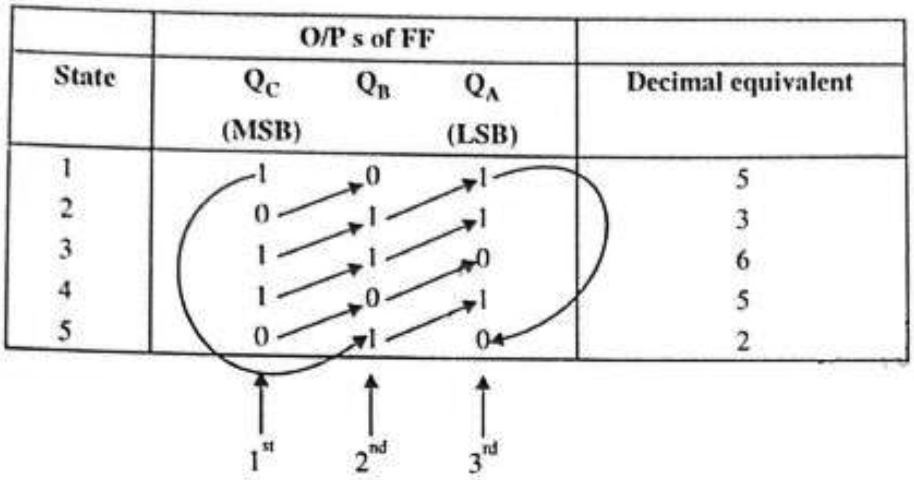


Figure 11.9(a): State table

Procedure for preparing State table:

(a) Write the format of table as shown in Figure 11.9(a). i.e. states, output of 'N' number of FF and decimal equivalent.

(b) Under column 1st (Normally MSB is selected) write down sequence 'vertically'.

(c) Now just imagine that sequence is having ring structure and shift the full sequence by y one bit. ∴ If sequence is 10110, rotate it left by one bit so we will get 01101. Write down shifted version under column 2.

(d) Again shift the sequence by one bit. Therefore with respect to original it will be shifted twice. Sequence under column 2 is 01101. Shift left by one bit. ∴ Sequence is 11010. Write it under column 3.

(e) Continue shifting of sequence unless you reach to LSB column. After shifting (Rolling) sequence now find out decimal equivalent of binary sequence. For our case we got 5, 3, 6, 5, 2.

In this decimal sequence, decimal number 5 is repeated twice, ∴ states are not distinct, ∴ we have to increase number of FFs to get distinct states. Therefore make state table again with four FFs.

State	O/P of FF				
	Q_D	Q_C	Q_B	Q_A	Decimal Equivalent
1	1	0	1	1	11
2	0	1	1	0	6
3	1	1	0	1	13
4	1	0	1	0	10
5	0	1	0	1	5

Figure 11.9(b): State table

Procedure for making truth table is same as we discussed just now. If we observe the decimal equivalent, it is coming out to be 11, 6, 13, 10, 5. All states are distinct.

(5) Design of Combinational circuit:

Here we have to find out O/P Y of combinational circuit, given to shift register, so that state changes from previous to next.

For example let's say **DCBA = 1011** (1 decimal), what should be input to shift register, so that **DCBA** will be **0110**. Means we have to apply '0' to input of shift register or in short **Y** should be **0**. Same way you find **Y** for each state. The format of table is shown in Figure 11.9(c) State table

State	Q_D	Q_C	Q_1
1	1	0	1
2	0	1	1
3	1	1	0
4	1	0	1
5	0	1	0

MSB Column

Figure 11.9(c): State table

As shown in Figure 11.9 (c). Output **Y** of combinational circuit is again nothing but **1** bit shifted version of LSB column.

(6) K-Map:

Valid states are 11, 6, 13, 10, 5, remaining states are don't care. States reached to 13, four variable K-Map should be used.

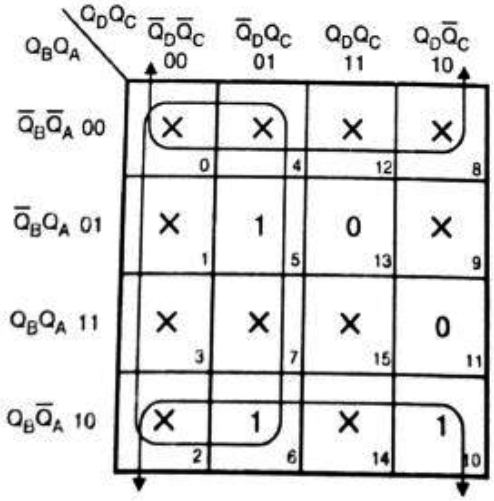


Figure 11.9(d): K – Map

(7) **Logic diagram:**

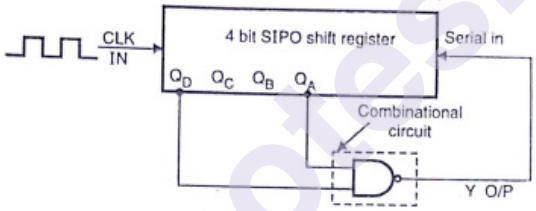


Figure 11.9(e): Logic diagram

11.10 PSEUDO – RANDOM BINARY SEQUENCE (PRBS) GENERATOR

One important application of shift register is pseudo-random generator. It is basically used to generate random sequence. Due to random sequence generation the same can be used for generating noise, as well as used for data encryption. By generating random noise, one can test *immunity given by his / her design to the noise*. By encrypting data, it becomes difficult job for a person who hacks the data in between.

Basically, the sequence generated is not truly random because it cycles through all possible combinations once every 2^n — 1 clock cycles (n= number of FFs).

Generation of pseudo random sequence is based on feedback given to shift register through *Combinational logic circuit*. Figure 11.10 depicts noise generator based on pseudo-random sequence. The combinational circuit used is simple EX-OR gate.

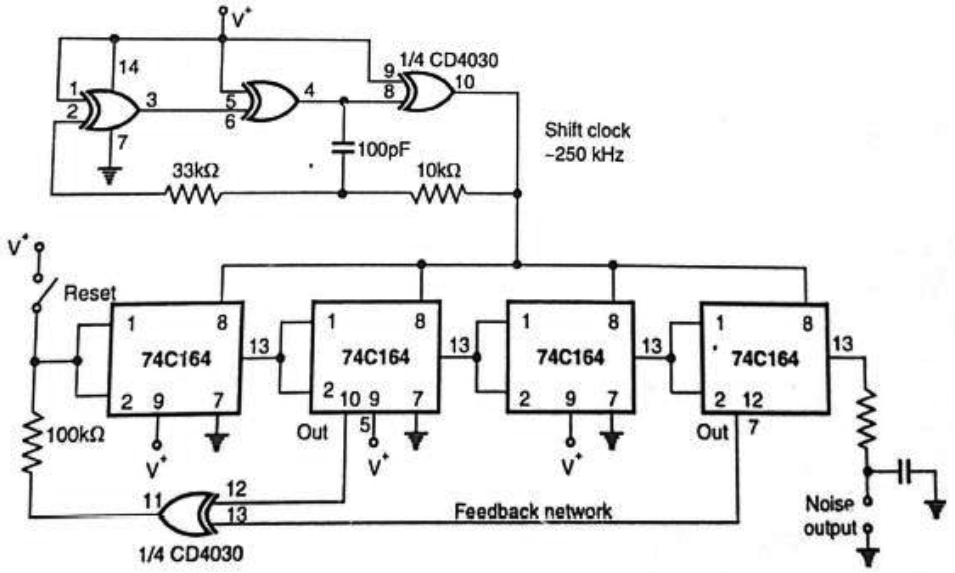


Figure 11.10: Noise generator based on pseudo-random sequence

It uses, 31 stage shift register provided with linear feedback to produce maximum length pseudo-random bit sequence. One of the significant features of pseudo-random sequence is that the noise produced from there is repeatable. The spectral density of the noise output of this circuit is uniform to within ±1 dB over the frequency range 20Hz to 20 kHZ.

The maximum length of sequence (MLS) generated will be $\mathbf{2^n - 1}$
$\therefore \mathbf{2^n - 1 = 2^{31} - 1}$, sufficient length to encrypt the data.

11.11 IC7495

We are going to study now TTL MSI Chip 7495. 4 bit shift register with serial and parallel synchronous operating mode.

Features of the chip are:

(1) Synchronous shift left capability.

(2) Synchronous parallel load.

(3) Separate shift and load clock inputs.

(4) Synchronous expandable shift right. Let's see internal block of IC 7495 and Pin configuration.

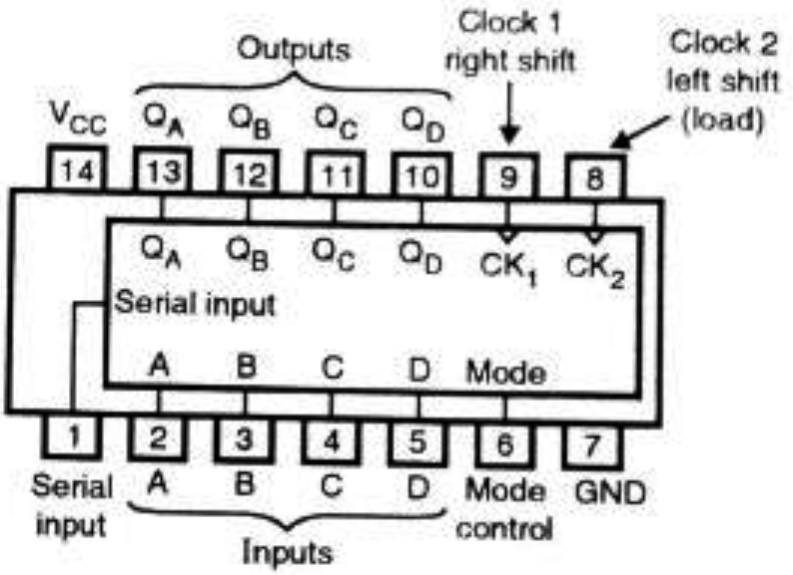


Figure 11.11(a): Pin Configuration – IC 7495

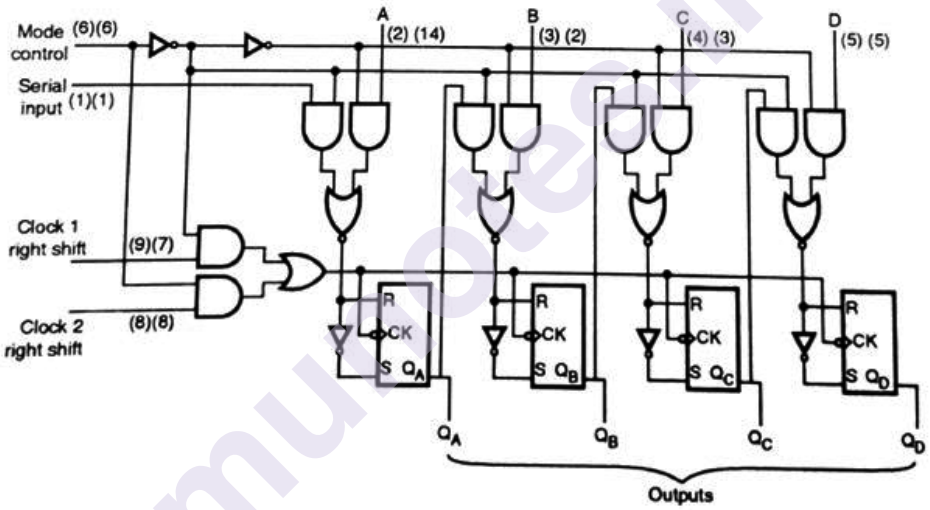


Figure 11.11(b): Logic diagram (Universal Shift Register)

We are going to perform basically three functions using 7495:

(1) Parallel load

(2) Shift left

(3) Shift right

1. Parallel Load:

a) Make mode control = 1, therefore AND gate 2, 4, 6, 8 will get enabled.

(b) Apply input ABCD.

(c) In this case clock 2 input (pin number 8) is enabled. Therefore a HIGH to LOW transition on clock and input will transfer parallel data ABCD inputs to $Q_0 - Q_3$.

(d) Here clock 1 input is don't case. DS is also don't care.

2. Shift Right:

(a) Make mode control = 0, therefore AND gate 1, 3, 5, 7 will be enabled and AND gate 2, 4, 6, 8 will get disabled.

(b) The data input to FF QA is now at serial input (DS).

(c) Clock and input is don't care. ABCD inputs are don't care.

(d) Apply CLOCK input to clock 1.

(e) A HIGH to LOW transition on enabled clock 1 input transfers data serially from from D_S to Q_A, Q_A to Q_B, Q_B to Q_C and Q_C to Q_D respectively (right shift).

3. Shift Left:

a) For shift left operation external connection are to be performed. Connect Q_D to C, Q_C to B and Q_B to A. as shown in figure 11.11.1

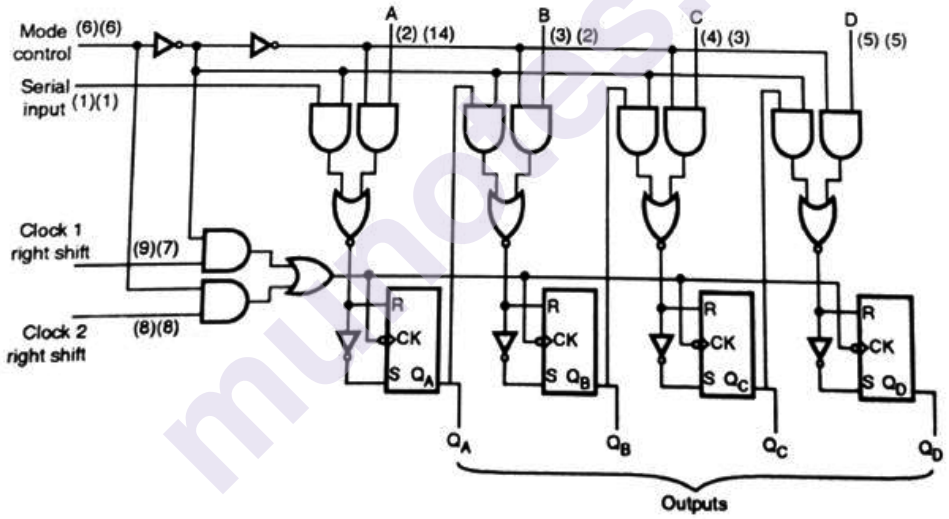


Figure 11.11.1: Three functions using 7495

11.12 SEVEN SEGMENT DISPLAYS

7 Segment display is one of the oldest methods of displaying values in electronic devices. The combination of 7 LEDs makes the whole display. Every time a single pin gets the power of a specific range it starts glowing. The pattern and drawing of LED make the decimal digit 8. Then turning on/off the specific pins make the 7-segment to show the other decimal numbers. The LED has a total of 10 input pins. It also comes in two types; one is the cathode and the other one is the anode. In both types,

one gives the common anode and the other has a common cathode. The 7-segment is useable directly by any low voltage device.

As 7-segment displays are very common components of digital devices, it is good to be familiar with the 'driving' circuits behind them, and the 4511 is a good example of a typical driver IC.

Its operating principle is to input a four-bit BCD (Binary-Coded Decimal) value and energize the proper output lines to form the corresponding decimal digit on the 7-segment LED display. The BCD inputs are designated A, B, C, and D in order from least-significant to most-significant. Outputs are labeled a, b, c, d, e, f, and g, each letter corresponding to a standardized segment designation for 7-segment displays. Of course, since each LED segment requires its own dropping resistor, we must use seven 470 Ω resistors placed in series between the 4511's output terminals and the corresponding terminals of the display unit.

Most 7-segment displays also provide for a decimal point (sometimes two), a separate LED and terminal designated for its operation. All LEDs inside the display unit are made common to each other on one side, either cathode or anode. The 4511 display driver IC requires a common-cathode 7-segment display unit, and so that is what is used here.

After building the circuit and applying power, operate the four switches in a binary counting sequence (0000 to 1111), noting the 7-segment display. A 0000 input should result in a decimal '0' display, a 0001 input should result in a decimal '1' display, and so on through 1001 (decimal '9'). What happens for the binary numbers 1010 (10) through 1111 (15)? Read the data sheet on the 4511 IC and see what the manufacturer specifies for operation above an input value of 9.

In the BCD code, there is no real meaning for 1010, 1011, 1100, 1101, 1110, or 1111. These are binary values beyond the range of a single decimal digit, and so have no function in a BCD system. The 4511 IC is built to recognize this, and output (or not output accordingly.

Three inputs on the 4511 chip have been permanently connected to either Vdd or ground: the 'Lamp Test,' 'Blanking Input,' and "Latch Enable." To learn what these inputs do, remove the short jumpers connecting them to either power supply rail (one at a time!), and replace the short jumper with a longer one that can reach the other power supply rail.

For example, remove the short jumper connecting the 'Latch Enable' input (pin #5) to ground, and replace it with a long jumper wire

that can reach all the way to the V_{dd} power supply rail. Experiment with making this input 'high' and 'low,' observing the results on the 7-segment display as you alter the BCD code with the four input switches.

After you have learned what the input's function is, connect it to the power supply rail enabling normal operation, and proceed to experiment with the next input (either 'Lamp Test' or 'Blanking Input').

Once again, the manufacturer's datasheet will be informative as to the purpose of each of these three inputs. Note that the 'Lamp Test' (LT) and 'Blanking Input' (BI) input labels are written with Boolean complementation bars over the abbreviations.

Bar symbols designate these inputs as active-low, meaning that you must make each one 'low' in order to invoke its particular function. Making an active-low input 'high' places that particular input into a "passive'' state where its function will not be invoked. Conversely, the "Latch Enable" (LE) input has no complementation bar written over its abbreviation, and correspondingly it is shown connected to ground ("low") in the schematic so as to not invoke that function.

The "Latch Enable" input is an active-high input, which means it must be made "high" (connected to V_{dd}) in order to invoke its function

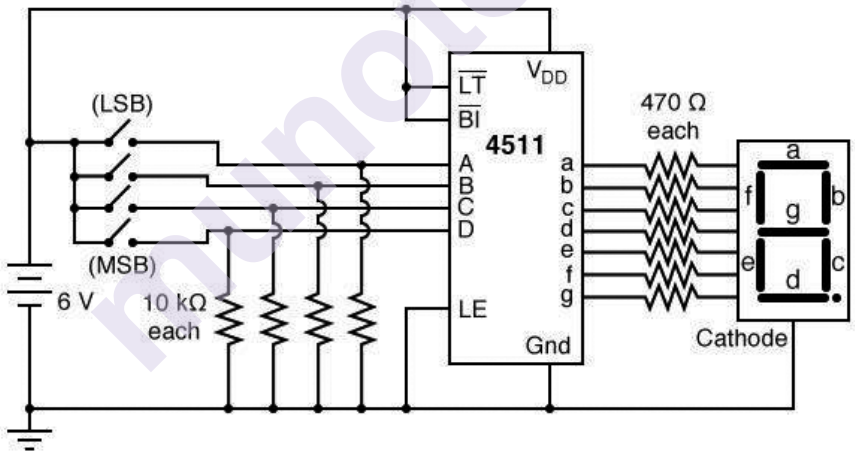


Figure 11.12 Schematic diagram of Seven Segment display with display driver

11.13 ANALYSIS OF SHIFT COUNTERS

In this circuit will be given and one has to find states through which shift register passes. To understand this let's solve example.

Example 2: A shift register with associated combinational logic circuit is shown in Figure 11.13. Explain its action by drawing waveforms for the

outputs of shift register (**ABCD**). Assume that initially shift register is in state **ABCD = 1111.**

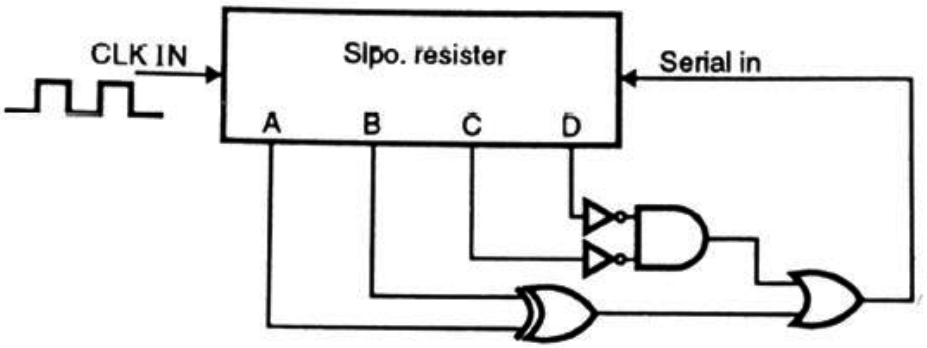


Figure 11.13: circuit diagram

Solution:

(1) Write equation for output Y.

(2) Basically output Y is serial in for shift register, shift is left shift, ∴ serial in placed in LSB i.e. D. So make table.

CLOCK edge	**Serial In**	**A**	**B**	**C**	**D**	$O/P\ Y=(A\oplus B)+\overline{C}\overline{D}$
-	-	1	1	1	1	0
1st	0	1	1	1	0	0
2nd	0	1	1	0	0	1
3rd	1	1	0	0	1	1
4th	1	0	0	1	1	0
5th	0	0	1	1	0	1
6th	1	1	1	0	1	0
7th	0	1	0	1	0	1
8th	1	0	1	0	1	1
9th	1	1	0	1	1	1
10th	1	0	1	1	1	1
11th	1	1	1	1	1	0

Figure 11.13(a): truth table

Explanation:

(Refer Figure 11.13(b))

Initially **ABCD = 1111**, ∴ **Y = (1 ⊕ 1) + = 0 + 0 = 0**
∴ **Y = serial in = 0**, ∴ when 1st CLOCK edge hits, **ABCD = 1110** (Legit shift)

Now **ABCD = 1110**, ∴ **Y = (1 ⊕ 1) + = 0 + 0 = 0**

$\therefore$ **Y = serial in = 0**, $\therefore$ when 2nd CLOCK edge hits, **ABCD = 1100**

Thus you proceed till you get initial condition **ABCD = 1111**.
From table (Figure 11.13(a)) we found that counter, counts through 15, 14, 12, 9, 3, 6, 13, 10, 5, 11, 7 and 15.

Thus generates sequence → 111100110101 Waveforms (Waveforms are as shown in Figure 11.13(b)).

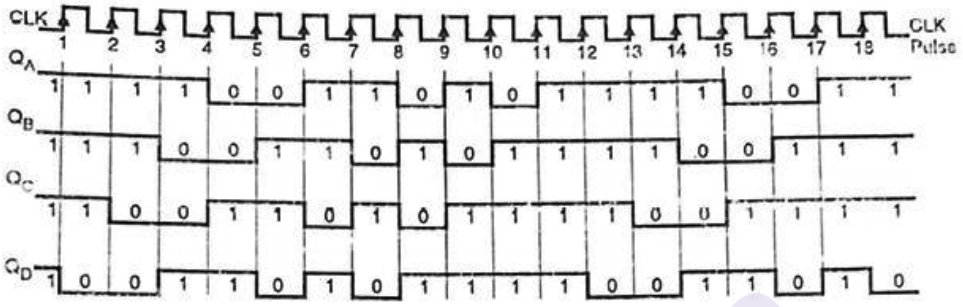


Figure 11.13(b): waveform

11.14 SUMMARY

In this chapter we have studied in detail about registers which includes parallel and shift registers, serial shifting, serial-in serial-out, serial-in parallel-out, parallel-in parallel-out, Ring counter, Johnson counter, Application of shift registers, Pseudo-random binary sequence generator, IC7495, Seven Segment displays and analysis of shift counters with examples.

11.15 REFERENCE FOR FURTHER READING

For further detailed study following are books prescribed by the University

1. Digital Electronics and Logic Design by N. G. Palan
2. Modern Digital Electronics by R. P. Jain
3. Digital Principles and Applications by Malvino and Leach

*****