

NOTES ON DIGITAL ELECTRONICS

Er Neeranjam Behera

Sr. Lect

PKA IET, BARGARH

Number System

Niranjan Behera
S. E. I. D. P. K. A. I. E. T., B. G. H.

Analog Representation: The numerical representation in which a quantity is represented by a voltage, current or meter movement that is proportional to the value of that quantity is called Analog representation.

Ex: - Automobile Speedometer →

Deflection of the needle is proportional to the speed of the auto.

Microphone

Digital Representation: The numerical representation in which a quantity is represented by the symbols of digits is known as digital representation.

Ex → Digital meter

Analog signal - Continuous

Digital = Discrete (Step by step)

Advantage of Digital System

- 1) Digital system are easier to design.
- 2) Storage information is easy.
- 3) Greater accuracy and precision.
- 4) Operation can be controlled by program.
- 5) Digital system is less affected by noise.
- 6) More digital circuitry can be ~~affe~~ fabricated on IC devices.

Numbers System :

A number system consist of a set of symbols and rules for representation of any number.

Types of number systems.

1. Decimal number system
2. Binary number system
3. Octal number system
4. Hexa-decimal number system.

Decimal number systems !

The decimal number system is composed of 10 numerals or symbol. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Using these 10 symbols as digits of a ~~num~~ number we can express any quantity how big or small it could be. The decimal number system is also called the base - 10 system because it has 10 digits. The decimal number system has evolved naturally as a result of the fact that people have 10 fingers on their hands. The word 'digit' Latin word means finger.

Base or radix : → The base or radix of a number system is defined as the number of different digits which can occur in each position in the number system.

Three important characteristics of a number system having positional notation (place value)

- The base or radix is equal to the number of digits
 - The largest digit is one less than the base
 - Each digit is multiplied by the base or radix raised to an appropriate power depending upon the digit position.

$$m_{SP} \quad \downarrow \quad LSD$$

3	5	47	2	16
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
10^3	10^2	10^1	10^0	10^{-1}

$$(3 \times 10^3) + (5 \times 10^2) + (4 \times 10^1) + (7 \times 10^0) \\ + (2 \times 10^{-1}) + (1 \times 10^{-2}) \\ + (6 \times 10^{-3})$$

In general any decimal number is equal to the sum of products of each digit value and its positional value.

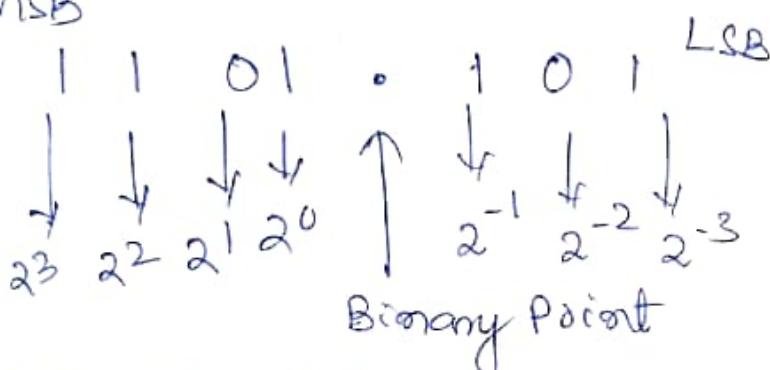
MSD \rightarrow most significant digit its carries most weight.

LSD - Least significant digit (or) least weight

Binary number system:

In digital systems and instruction, the number system used has a radix 2 and called as binary system. Binary means two. The binary number system uses only two digits 0 and 1. The two symbols '0' and '1' used in binary number are called bit.

MSB



The bit at the left-most position (i.e. the bit with the longest position value) is called most-significant bit (MSB).

The bit at the right-most position (i.e. the bit with the smallest position value) is called the least significant bit (LSB).

Largest decimal number = $2^n - 1$

$$n = \text{no. of bit}$$

Ex: Largest decimal number for a 5-bit number

$$n=5 \quad 2^5 - 1 = (32 - 1) = 31$$

Binary to decimal conversion

Following steps required to convert binary to decimal number or base 2 to base 10.

Step 1 - Write the binary number

Step 2 - Directly under the binary number write the position values or weights of each bit working from right to left

Step 3 → If a zero appears in a digit position cross-out the weight for that position

Step 4 → Adding the remaining weights to obtain the decimal equivalent.

Example $(101)_2 = (?)_{10}$

$$\begin{array}{r} 1 \ 0 \ 1 \\ \downarrow \ \downarrow \ \downarrow \\ 2^2 \ 2^1 \ 2^0 \end{array}$$

$$2^2 + 0 + 2^0 = 4 + 0 + 1 = (5)_{10}$$

Fractional binary-to-Decimal conversion

$$\begin{array}{r} 0 . \ 1 \ 0 \ 1 \ 1 \\ \downarrow \ \downarrow \ \downarrow \ \downarrow \\ 2^{-1} \ 2^{-2} \ 2^{-3} \end{array}$$

$$2^{-1} + 2^{-3} = 0.5 + 0.125 = (0.625)_{10}$$

Example

Convert the binary number 1101.0110
to its decimal number
equivalent.

$$\begin{array}{ccccccccc} 1 & 1 & 0 & 1 & . & 0 & 1 & 1 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow \\ 2^3 & 2^2 & 2^1 & 2^0 & . & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \end{array}$$

$$2^3 + 2^2 + 2^0 \cdot 2^{-2} + 2^{-3}$$

Alternatively $8 + 4 + 1 + 0.25 + 0.125 = 13.375$

$$(1101.0110)$$

$$= (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \cdot (0 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (0 \times 2^{-4})$$

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

$$= 8 + 4 + 0 + 1 \cdot (0 \times 0.5) + (1 \times 0.25) + (1 \times 0.125)$$

$$= (13.375)_{10} + (0 \times 0.0625)$$

Decimal - to - Binary conversion

① Sum-of-weight method

② Repeated division by 2-method

Eight bit binary weight (Right to left)

128, 64, 32, 16, 8, 4, 2, 1

Conversion of number 9 by Sum-of-Weight method.

$$9 = 8 + 1 \text{ or, } 9 = 2^3 + 2^0$$

By placing 1s in appropriate weight positions 2^3 and 2^0 , and 0s in the 2^2 and 2^1 position

$$\begin{array}{cccc}
 2^3 & 2^2 & 2^1 & 2^0 \\
 \downarrow & \downarrow & \downarrow & \downarrow \\
 1 & 0 & 0 & 1
 \end{array}$$

$$9_{10} = (1001)_2$$

Example $(93)_{10} = (?)_2$

$$93 = 64 + 16 + 8 + 4 + 1$$

$$\begin{array}{ccccccccc}
 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 1 & 0 & 1 & 1 & 1 & 0 & 1
 \end{array}$$

$$(93)_{10} = (1011101)_2$$

Fractional Decimal - To - Binary

List of four fractional binary

Weight \rightarrow

$$\begin{array}{cccccc} & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0.5 & 0.25 & 0.125 & 0.625 \end{array}$$

Conversion of the decimal fraction
0.625 to its equivalent binary.

$$\begin{aligned} 0.625 &= 0.5 + 0.125 \\ &= 2^{-1} + 2^{-3} \\ &= 0.101 \end{aligned}$$

1 is in the 2^{-1} position and 1 in the
 2^{-3} position

Example

$$\text{Fraction } (0.375)_{10} = (?)_2$$

$$0.375 = 0.25 + 0.125$$

$$= 2^{-2} + 2^{-3} = (0.011)_2$$

0 in the 2^1 position, 1 in the 2^{-2} and 2^{-3}
positions.

Repeated Division-by-2 method for
Integer Decimal - to - Binary

Step-1 - Begin by dividing decimal
numbers by 2.

Step-2 \rightarrow Divide each resulting quotient
by 2 until there is a 0 whole
number quotient.

Step-3 The remainders generated by each division from the binary number. The first remainder to be produced is the least significant bit (LSB) in the binary number and the last remainder to be produced is the most significant bit (MSB). Reading the remainders from bottom-to-top constitutes required binary number.

Example

$$\begin{array}{r}
 2 | 10 \\
 \hline
 2 | 5 — 0 \quad \text{LSB} \\
 \hline
 2 | 2 — 1 \\
 \hline
 1 — 0 \quad \text{MSB}
 \end{array}$$

$$(10)_{10} = (1010)_2$$

$$\begin{array}{r}
 2 | 45 \\
 \hline
 2 | 22 — 1 \quad \text{—LSB} \\
 \hline
 2 | 11 — 0 \\
 \hline
 2 | 5 — 1 \\
 \hline
 2 | 2 — 1 \\
 \hline
 2 | 1 — 0 \\
 \hline
 0 — 1 \quad \text{—MSB}
 \end{array}$$

$$(45)_{10} = (101101)_2$$

Repeated Multiplication by 2 for fractional to Binary conversion.

Decimal to binary (fractional) can be converted to binary by repeated multiplication - by - 2.

Step-1 → Begin by multiplying the given decimal fraction by 2 and then multiplying each resulting fraction part of the product by 2.

Step-2 → Repeat step 1 until the fractional product is zero or until the desired number of decimal places is reached.

Step-3 → The carried bits or carries generated by the multiplication produce the binary number. The first carry produced is the most-significant bit (MSB) and the last carry is the least significant bit (LSB). Reading from top to bottom constitutes the required fractional binary.

$$(0.3125)_{10} = ()_2$$

$$0.3125 \times 2 = 0.625 \text{ with carry } 0$$

$$0.625 \times 2 = 1.25 \text{ with carry } 1$$

$$0.25 \times 2 = 0.50 \text{ with carry } 0$$

$$0.50 \times 2 = 1.00 \text{ with carry } 1$$

$$(0.3125)_{10} = (0.0101)_2$$

TOP
MSB

LSB
Bottom

Example:

Convert the decimal 0.8127 to its

binary equivalent

$$0.8127 \times 2 = 1.6254 \text{ Carry } - 1 \text{ MSB}$$

$$0.6254 \times 2 = 1.2508 \text{ Carry } - 1$$

$$0.2508 \times 2 = 0.5016 \text{ Carry } - 0$$

$$0.5016 \times 2 = 1.0032 \text{ Carry } - 1$$

$$0.0032 \times 2 = 0.0064 \text{ Carry } - 0 \text{ LSB}$$

Bottom

$$\underline{(0.8127)} \quad (0.8127)_{10} = (0.11010\cdots)_2$$

Binary Arithmetic:

Binary addition is carried out in a same manner as decimal addition. It is much simpler than the decimal. As the binary only two digits 0 and 1 are used.

case - 1 $\rightarrow 0+0=0$

case - 2 $\rightarrow 0+1=1$

case - 3 $\rightarrow 1+0=0$

case - 4 $\rightarrow 1+1=10$ (Carry one 1)

Example. Add $(1001)_2$ and $(1110)_2$

$$\begin{array}{r}
 1110 \rightarrow 14 & \text{1st column } 0+0=1 \\
 1001 - 09 & \text{2nd } " - 1+0=1 \\
 \hline
 10111 - 23 & \text{3rd } " - 1+0=1 \\
 & \text{4th } " - 1+1=10 \\
 & \text{Carry 1}
 \end{array}$$

Example

$$\begin{array}{r}
 100101 \\
 -110111 \\
 \hline
 10110100
 \end{array}$$

Binary Subtraction:

$0-0=0$
$1-0=1$
$1-1=0$
$\boxed{10-1=1}$

(one is borrowed
and 0 becomes 10)

Example
Subtract $(11)_2$ from $(1000)_2$

$$\begin{array}{r}
 1000 \rightarrow 8 \\
 -11 \quad -3 \\
 \hline
 101 \quad 5
 \end{array}$$

Subtract $(111001)_2$ from $(1110001)_2$

$$\begin{array}{r}
 1110001 \rightarrow 113_{10} \\
 111001 \rightarrow 57_{10} \\
 \hline
 111000 \quad 56_{10}
 \end{array}$$

Binary multiplication:

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Multiply $(10111)_2$ by $(101)_2$

$$\begin{array}{r} 10111 \\ \times 101 \\ \hline 10111 \\ 00000 \times \\ 10111 \\ \hline 1110011 \end{array}$$

$\begin{array}{r} 23_{10} \\ - 5_{10} \\ \hline (115)_{10} \end{array}$

Binary division :

$$\boxed{\begin{array}{l} 0 \div 1 = 0 \\ 1 \div 1 = 1 \end{array}}$$

Expt. Divide 11000_2 by 1000_2

$$\begin{array}{r} 11 \\ 1000 \overline{)11000} \\ 1000 \\ \hline 1000 \\ 1000 \\ \hline 0000 \\ 11110 \end{array}$$

$8 \overline{)24} \quad \begin{array}{r} 3 \\ 24 \\ 24 \\ \hline 0 \end{array}$

Expt.

$$\begin{array}{r} 1111000 \\ 100 \downarrow \quad | \\ 0111 \quad | \\ 100 \downarrow \quad | \\ 110 \quad | \\ 100 \quad | \\ 100 \\ 000 \\ 000 \end{array}$$

$4 \overline{)120} \quad \begin{array}{r} 308 \\ 120 \\ 120 \\ \hline 00 \end{array}$

Niranjan Behere
Sr. Lect PRAEST
BARGARH

Octal Number System

The octal number system provides a convenient way to express binary numbers and codes. This system is composed of eight digits 0, 1, 2, 3, 4, 5, 6, 7. To count beyond 7 we form two digit combinations in the same as decimal and binary number system. Thus beyond 7 we count as 10, 11, 12, 13, 14, 15, 16, 17, ~~18~~, 20, 21 and so on.

$$\begin{array}{ccccccccccccc}
 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & & -1 & -2 & -3 & -4 & -5 \\
 & \cancel{7} & \cancel{6} & \cancel{5} & \cancel{4} & \cancel{3} & \cancel{2} & \cancel{1} & . & 8 & 8 & 8 & 8 & 8 \\
 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & & 8 & 8 & 8 & 8 & 8
 \end{array}$$

↑
Octal Point

Octal - to Binary Conversion

Step-1 : Write the octal number

Step-2 : Directly under the octal number write the position weight of each digit working from right to left.

Step-3 : Multiply each octal digit by its position weight and take sum of the product.

Example : $(2374)_8 = (?)_{10}$

$$\begin{array}{ccccccc}
 & 2 & 3 & 7 & 4 & & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & & \\
 8^3 & 8^2 & 8^1 & 8^0 & & &
 \end{array}$$

$$= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0)$$

$$= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1)$$

$$= 1024 + 192 + 56 + 4$$

$$= (1276)_{10}$$

Example $(372.6)_8 = (\text{ })_{10}$

$$\begin{array}{cccc} 3 & 7 & 2 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 8^2 & 8^1 & 8^0 & 8^{-1} \end{array}$$

$$= (3 \times 64) + (7 \times 8) + (2 \times 1) + (6 \times \frac{1}{8})$$

$$= 192 + 56 + 2 + 0.75 = (250.75)_{10}$$

Decimal-to-octal conversion! -

Integers decimal to octal conversion

Step-1 - Begin by dividing decimal number by 8.

Step-2 - Divide each resulting quotient by 8 until there is zero whole number quotient.

Step-3 → The remainders are generated by each division from the octal number.

The first remainder digit (LSD) in the octal number and the last remainder to be produced is the most significant digit (MSD). Reading the remainders from bottom-to-top constitutes the octal number.

Example $(266)_{10} = (\text{ })_8$

8	266	2	- LSD
8	33		
8	4	1	
8			↓ MSD

$$(266)_{10} = (412)_8$$

Fractional decimal to octal conversion

- Begin by multiplying the given decimal fraction by 8 and
- Repeat step until the fractional product is zero with the desired number of decimal places reached.
- The carried digits or carries by the multiplication produce the octal digits.

Exp $(0.3125)_{10} = (?)_8$

$$0.3125 \times 8 = 2.5000 \text{ with a carry } 2$$

$$0.5 \times 8 = 4.00 \text{ with a carry } 4$$

$$(0.3125)_{10} = (\cancel{1}00)_{10} (24)_8$$

Octal-to-Binary conversion:

The conversion from octal-to-binary is performed by converting each digit to its 3-bit binary equivalent.

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Example $(13)_8 = (\quad)_2$

$$\begin{array}{r} 1 \quad 3 \\ \overbrace{001} \quad \overbrace{011} \end{array}$$

$$(13)_8 = (001011)_2 = (1011)_2$$

$$(526)_8 = (\quad)_2$$

$$\begin{array}{r} 5 \quad 2 \quad 6 \\ \downarrow \quad \downarrow \quad \downarrow \\ 101 \quad 010 \quad 110 \end{array}$$

$$(526)_8 = (101010110)_2$$

Binary-to-octal conversion

Conversion of a binary integer number to an octal integer number is reverse of the octal-to-binary conversion.

Example $(100111010)_2 = (\quad)_8$

$$\begin{array}{r} 100 \quad 111 \quad 010 \\ \downarrow \quad \downarrow \quad \downarrow \\ 4 \quad 7 \quad 2 \end{array} \quad (472)_8$$

$$(011010110)_2 = (\quad)_8$$

$$\begin{array}{r} 011 \quad 010 \quad 110 \\ \downarrow \quad \downarrow \quad \downarrow \\ 3 \quad 2 \quad 6 \end{array} \quad (326)_8$$

Convert $(1011100)_2 = (?)_8$

adding a leading zero.

$$\begin{array}{r} \overset{2}{\cancel{1}} \overset{1}{\cancel{0}} \overset{1}{\cancel{1}} \overset{1}{\cancel{1}} \overset{0}{\cancel{0}} \\ + \quad + \quad + \\ \hline 2 \quad 7 \quad 1 \end{array} \quad (271)_8$$

Usefulness of Octal System:

In computers, binary numbers may represent:

1. actual numerical data
2. number corresponding to a location (called Address) in a memory.
3. an instruction code
4. a code representing alphabetic and other non-numerical characters
5. group of bits representing the status of devices internal or external to the computer.

Hexadecimal numbers system:

The hexadecimal system is composed of 16 digits which includes numerals and alphabetic characters.

The 16 digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F and F.

most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal numbers very convenient because each hexadecimal digit represents 4 bit binary number

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

In order to count in hexadecimal beyond F, we form two digit combinations by starting with the second digit followed by the first digit.

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C,
 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28
 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31 - - -

Hexadecimal - to - Decimal conversion

- Write the hexadecimal number
- Directly under the hexadecimal number, write the position weight of each digits working from right to left.
- multiply the decimal value of each hexadecimal digit by its position weight and take sum of the product.

$$(E5)_{16} = (?)_{10}$$

$$\begin{array}{r} E \quad \quad 5 \\ \downarrow \quad \quad \downarrow \\ 16^1 \quad \quad 16^0 \end{array}$$

$$\begin{aligned} &= (E \times 16^1) + (5 \times 16^0) \\ &= (14 \times 16) + (5 \times 1) \\ &= (224 + 5) = (229)_{10}. \end{aligned}$$

$$(0.12)_{16} = (?)_{10}$$

$$\begin{array}{r} 1 \quad \quad 2 \\ \downarrow \quad \quad \downarrow \\ 16^{-1} \quad 16^{-2} \end{array}$$

$$\begin{aligned} &= (1 \times 16^{-1}) + (2 \times 16^{-2}) \\ &= (0.0625) + (0.0078) \\ &= (0.0703)_{10} \end{aligned}$$

Decimal-to-Hexadecimal Conversion

- Begin by dividing the given decimal number by 16.
- Divide each resulting quotient by 16 until there is zero whole number quotient.
- The remainder generated by each division form the hexadecimal number. The first remainder to be produced is the least-significant digit (LSD) in the hexadecimal number and the last remainder to be produced is the most significant digit (MSD). Reading the remainders from bottom-to-top constitutes the hexadecimal number.

Exp-1
$$\begin{array}{r} 16 \longdiv{650} \\ 16 \longdiv{40} \\ 32 \end{array} \quad \begin{array}{l} \text{LSD} \\ 10 = A \\ 8 \\ \text{MSD} \end{array} \quad (28A)_{16}$$

Exp-2 $(14)_{10} = (E)_{16}$

Exp-3
$$\begin{array}{r} 16 \longdiv{80} \\ 5 = 0 \end{array} \quad (80)_{10} = (50)_{16}$$

Exp-4
$$\begin{array}{r} 16 \longdiv{3000} \\ 16 \longdiv{187} \\ 11 \end{array} \quad \begin{array}{l} 8 \\ 11 - (B) \\ B \end{array} \quad (3000)_{10} = (BB8)_{16}$$

Hexadecimal - to - Binary Conversion

The procedure for converting hexadecimal number to its binary equivalent is as follows! -

- Write the hexadecimal number
- Write the 4-bit binary equivalent for ~~the~~ each hexadecimal digit.
- If there are any zeros on the leftmost position of the binary numbers obtained, drop off the zeros and write the answer as a binary number.

conversion of $(2D6)_{16}$ to its equivalent binary.

2	D	6
\downarrow	\downarrow	\downarrow
0010	1101	0110

0010 1101 0110
 $\underbrace{\downarrow}_{\text{Drop off the leading zeros}}$

Drop off the leading zeros

$$(2D6)_{16} = (1011010110)_2$$

Example - 1 $(9F2)_{16} = (?)_2$

9	F	2
1001	1111	1010

$$(9F2)_{16} = (100111111010)_2$$

Binary-to-Hexadecimal Conversion

Conversion from Binary to Hexadecimal numbers is reverse of the process of ~~Bi~~ Hexadecimal to Binary number.

→ Write the binary number

→ Starting from the right-most position group the binary numbers into groups of four bits. If necessary, we can add zeros at the left-most position

→ Convert each 4-bit binary to its equivalent hexadecimal digit

Conversion of binary 10111 to its equivalent hexadecimal

$$\begin{array}{r} 10111 \\ \text{Three zeros are added to form a four-bit group} \\ \swarrow \quad \downarrow \quad \downarrow \\ 0001 \quad 0111 \end{array}$$

$(10111)_2 = (17)_{16}$

Example: $(111110000)_2 = (\text{?})_{16}$

$$\begin{array}{r} 00111110000 \\ \text{Grouped as } 00 \ 1111 \ 10000 \\ \text{3 F 0...} = (3F0)_{16} \end{array}$$

Hexadecimal-to-octal conversion

- Write the hexadecimal number
- Replace each hexadecimal digit by its 4-bit binary equivalent. This will give us the binary equivalent of the given hexadecimal number.
- Form 3-bit combinations by starting from the most rightmost position. Replace each 3-bit combination by its octal equivalent for the given hexadecimal number.

Example Convert $(5C2)_{16} = (?)_8$

5	C	2
↓	↓	↓
0101	1100	0010

010111000010
.....
↓ ↓ ↓
2 7 0 2

 $= (2702)_8$

Convert $(B9F.AE)_{16}$ to octal

B	9	F	:	A	E
↓	↓	↓		↓	↓
1011	1001	1111	.	1010	1110

10111001111	.	101011100
.....	
↓ ↓ ↓ ↓		↓ ↓ ↓
5 6 3 F		5 3 4

$(5637.534)_8$

Octal-to-Hexadecimal conversion →

- Write the octal number.
- Replace each octal digit by its 3-bit binary equivalent. This will give us the binary equivalent of the given octal number.
- Form the 4-bit combinations by starting from the right-most-position. Replace each 4-bit combination by its hexadecimal equivalent. This will give us the hexadecimal equivalent for the given octal number.

Example Convert $(321)_8 = (?)_{16}$

$$\begin{array}{ccc}
 3 & 2 & 1 \\
 \downarrow & \downarrow & \downarrow \\
 \widetilde{011} & \widetilde{010} & \widetilde{001} \\
 \begin{array}{c}
 \overset{\circ}{0} \overset{\circ}{0} \overset{\circ}{0} \overset{\circ}{1} \overset{\circ}{1} \overset{\circ}{0} \overset{\circ}{1} \overset{\circ}{0} \overset{\circ}{0} \overset{\circ}{1} \\
 \downarrow \quad \downarrow \quad \downarrow \\
 D \quad 1
 \end{array} & & = (D1)_{16}
 \end{array}$$

Convert the octal number $(1024)_8$ to its hexadecimal equivalent

$$\begin{array}{cccc}
 1 & 0 & 2 & 4 \\
 \downarrow & \downarrow & \downarrow & \downarrow \\
 001 & 000 & 0010 & 100 \\
 \begin{array}{c}
 \overset{\circ}{0} \overset{\circ}{0} \overset{\circ}{1} \overset{\circ}{0} \overset{\circ}{0} \overset{\circ}{0} \overset{\circ}{1} \overset{\circ}{0} \overset{\circ}{1} \overset{\circ}{0} \\
 \downarrow \quad \downarrow \quad \downarrow \\
 2 \quad 1 \quad 4
 \end{array} & & & = (214)_{16}
 \end{array}$$

SIGNED NUMBER

In Sign(-) magnitude form, additional bit called the sign bit is placed in front of the number. If the sign bit is 0, the number is positive. If it is a 1, the number is negative.

Example

$$0 \uparrow 101001 = +41$$

Sign bit

$$1 \uparrow 101001 = -41$$

Sign bit

Subtraction using complement method

1's complement :

In 1's complement subtraction, add the 1's complement of subtrahend to the minuend. If there is a carry out then the carry is added to LSB. This is called end-around carry. If the msb is 0, the result is positive. If the msb is 1, the result is negative and is in its 1's complement form. Then take its 1's complement to get the magnitude in binary.

Example

Subtract $(10000)_2$ from $(11010)_2$
using 1's complement

$$\begin{array}{r}
 11010 \text{ f.26} \rightarrow 11010 \\
 - 10000 \text{ (16)} \\
 \hline
 +10 \quad 01111 \text{ (1's complement)}
 \end{array}$$

Carry $\rightarrow 101001$

$$\begin{array}{r}
 \hline
 & 1 \\
 \hline
 01010 = +10
 \end{array}$$

Subtract $(100011)_2$ from $(010010)_2$

$$\begin{array}{r}
 010010 \text{ (18)} \quad 010010 \\
 - 100011 \text{ (35)} \quad + 011100 \text{ 1's complement} \\
 \hline
 -(17) \quad \hline
 101110
 \end{array}$$

As there is no carry the minuend is smaller than the subtrahend. Thus it is required to complement the sum and attach - sign to get the result.

$$\text{Result} = -(010001)_2$$

Using 2's complement

In 2's complement subtraction, add the 2's complement of subtrahend to the minuend. If there is a carry out ignore it. If the MSB is 0, the result is positive. If the MSB is 1, the result is negative and is in its 2's complement form. Then take its 2's complement to get the magnitude in binary.

Example i) Subtract $(11010101)_2 - (10011010)_2$

$$\begin{array}{r} \text{1's complement of Subtrahend} - 01100101 \\ + 1 \\ \hline \end{array}$$

$$\begin{array}{r} \text{2's complement of Subtrahend} \quad 01100110 \\ 11010101 - \text{Minuend} \\ + 01100110 \quad \begin{array}{l} \text{2's complement of} \\ \text{Subtrahend} \end{array} \\ \hline 100111011 \end{array}$$

$$\text{Result } (00111011)$$

Carry is ignored

$$\text{ii) } (01111)_2 - (01000)_2$$

$$01111 \rightarrow \text{Minuend}$$

$$\begin{array}{r} 10111 \rightarrow \text{1's complement of} \\ \text{Subtrahend} \\ + 1 \\ \hline 11000 \rightarrow \text{2's complement of} \end{array}$$

$$\begin{array}{r} 01111 \\ + 11000 \\ \hline \end{array}$$

100111 Neglect the carry

$$\text{Result } (00111)_2 \quad \begin{array}{l} \text{2's complement of Result} \\ (10001)_2 \end{array}$$

Digital codes

In practise the digital electronic requires to handle data which may be numeric, alphabets and special characters. This requires the conversion of the incoming data into binary format before it can be processed. ~~These~~ There are various possible ways of doing this and this process is called ~~coding~~ encoding. To achieve the reverse of it we use decoders.

Weighted and Non-weighted codes

There are two types of binary codes.

1) Weighted binary codes

2) Non-weighted binary codes.

In weighted codes for each position

(or bit) there is specific weight attached. For example in binary numbers each bit is assigned particular weight 2^n where n is bit number for $n = 0, 1, 2, 3, 4, 5, \dots$ the weights are 1, 2, 4, 8

Non-weighted codes are codes which are not assigned with any weight to each digit position i.e. each digit position within numbers is not assigned fixed value.

Example - Excess-3 (X_{S-3}) codes and Gray codes

Binary codes Decimal (BCD codes)

BCD is a weighted code. In weighted codes, each successive digit from right to left represents weight equal to some specified value and to get the equivalent decimal number ~~we~~ add the products of the weights by the corresponding binary digit. 8421 is the most common because 8421 BCD is the most natural amongst the other possible codes.

In 8421 BCD codes each decimal digit is expressed by its 4 bit binary equivalent. for example 2735 can be expressed in its binary equivalent as follows

2	7	3	5
0010	0111	0011	0101

5	6	7
0101	0110	0111

BCD Addition:

Addition of BCD (8421) is performed by adding two digits of binary, starting from least significant digit. In case if the result is an illegal code (greater than 9) or if there is a carry out of one then add 0110 (6) and the resulting carry to the next most significant.

Example: Add 679.6 and 536.8 using BCD addition

$$\begin{array}{r} \text{679.6 - BCD - 0110 0111 1001 - 0110} \\ \text{536.8 BCD + 0101 0011 0110 - 1000} \\ \hline \text{1216.4} \end{array}$$

All are illegal
Codes
Add 0110 to each

$$\begin{array}{r} 0001 0010 0001 0110 .0100 \\ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ 1 \quad 2 \quad 1 \quad 6 - 4 \end{array}$$

Corrected Sum = 1216.4

BCD SUBTRACTION! -

The BCD subtraction is performed by subtracting the digits of each 4-bit group of the subtrahend from corresponding 4-bit group of the minuend in the binary starting from the LSD. If there is no borrow from the next higher group then no correction is required. If there is a borrow from the next group, the $6_{10}(0110)_2$ is subtracted from the difference term of this group.

Example

Subtract 147.8 from 206.7 using
8421 BCD code.

$$\begin{array}{r}
 206.7 \\
 -147.8 \\
 \hline
 58.9
 \end{array}
 \quad
 \begin{array}{l}
 0010\ 0000\ 0110.\ 0111 \text{ (206.7 in} \\
 \text{BCD)} \\
 0010\ 0100\ 0110.\ 1000 \text{ (147.8 in} \\
 \text{BCD)} \\
 \hline
 0000\ 1011\ 1110.\ 1111 \\
 -0110\ 0110.\ 0110 \\
 \hline
 0101\ 1000.\ 1001 \\
 5\ 8\ .\ 9 \text{ (correct} \\
 \text{difference 58.9)}
 \end{array}$$

Excess Three (XS-3) code:

The Excess-3 code is a non-weighted BCD code. This derives it name from the fact that each binary code word is the corresponding 8421 code word plus 0011(3). It is a sequential code, it is a self complementing code.

Example

XS-3 code $(11011)_2$

$$\begin{array}{r}
 & 1 & 1 & 0 & 1 & 1 \\
 + & & & & & 1 & 1 \\
 \hline
 \text{XS-3 code} & 1 & 1 & 1 & 1 & 0
 \end{array}$$

XS-3 Addition:

In XS-addition, add the XS-3 numbers by adding 4-bit groups in each column starting from the LSD. If there is no carry out from the addition to any of the 4-bit groups subtract 0011 from the sum term of those groups. If there is a carry out add 0011 to the sum term.

of those groups.

Example

Add 37 and 28 using XS-3 code

$$\begin{array}{r} 37 \\ + 28 \\ \hline 65 \end{array}$$

$$\begin{array}{r} 0110 \quad 1010 \quad (37 \text{ in XS-3}) \\ + 0101 \quad 1011 \quad (28 \text{ in XS-3}) \\ \hline 1011 \quad 10101 \\ \quad \quad \quad +11 \\ \hline 1100 \quad 0101 \\ - 0011 \quad + 0011 \\ \hline 1001 \quad 1000 \end{array}$$

carry is generated
propagate carry
(Add 0110 to
correct 0101 and
Subtract 0011 to
correct 1100)
correct sum in XS-3
 $= (65)_{10}$

X-3 SUBTRACTION

To subtract in XS-3 numbers by
subtracting each 4-bit group of the subtractor
from the corresponding 4-bit group of the
minuend starting from LSD. If there is no
borrow from the next 4-bit group add
0011 to the difference term of such groups.
If there is borrow, subtract 0011 from the
difference term.

Example → Subtract 175 from 267 using
XS-3 code

$$\begin{array}{r} 267 \\ - 175 \\ \hline 092 \end{array}$$

$$\begin{array}{r} 0101 \quad 1010 \quad (267 \text{ in XS-3}) \\ - 0100 \quad 1010 \quad 1000 \quad (175 \text{ in XS-3}) \\ \hline 0000 \quad 1111 \quad 0010 \quad (\text{correct } 0010 \text{ and } 0000 \text{ by adding } 0011 \text{ and correct } 0011) \\ + 0011 \quad - 0011 \quad + 0011 \quad 1111 \text{ by subtracting } 0011 \\ \hline 0011 \quad 1100 \quad 0101 \quad 1111 \end{array}$$

Corrected difference
in XS-3 = 92₁₀

ASCII CODE

The American Standard Code for Information Interchange (ASCII) pronounced as 'ASKEE' is widely used alphanumeric code. This is basically 7 bit code. The number of different bit patterns that can be created with 7 bits is $2^7 = 128$, the ASCII can be used to encode both the uppercase and lowercase characters of the alphabets (52 symbols) and some special symbols in addition to the 10 decimal digits. It is used extensively for pointers and terminals that interface with small computer systems.

EBCDIC CODE :-

The Extended Binary Coded Decimal Interchange code (EBCDIC) pronounced as eb-si-dik is an 8 bit alphanumeric code. Single $2^8 = 256$ bit patterns can be formed with 8 bits. It is used by most large computers to communicate in alphanumeric data.

GRAY CODE :

The gray code is a non-weighted code. It is not a BCD code. It is cyclic code because successive word in this differ in one bit position only i.e. it is a unit distance code.

Gray code is used in instrumentation and data acquisition systems where linear or angular displacement is measured. They are also used in shaft encoders, I/O devices, A/D converters and other peripheral equipment.

Binary to Gray Conversion:

- Step - 1 Starting from the most-significant bit (msb), the first digit of the two codes is the same.
- Step - 2 : Add the first and second Binary bits, to obtain the second Gray bit, discard the carry.
- Step - 3 : Add the second and third Binary bits, to obtain the third Gray bit, discard the carry.
- Step - 4 : Follow the same procedure, till the conversion is complete.

Example: convert the binary 1001 to the Gray code.

$$\begin{array}{r} \oplus \\ | \rightarrow 0-\oplus-0-\oplus-1 \\ | \quad | \quad | \quad | \\ | \quad 0 \quad | \end{array}$$

Result - 1101

Convert the binary number 101101 to Gray code.

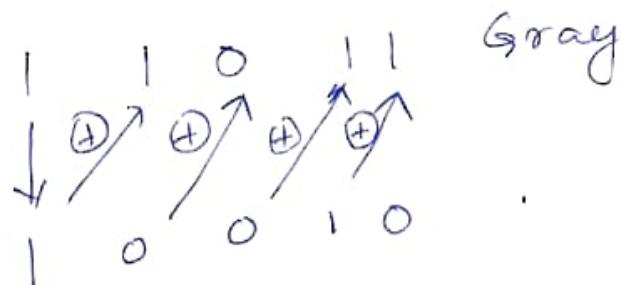
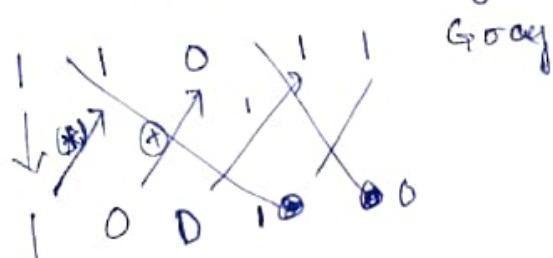
$$\begin{array}{r} \oplus \\ | \rightarrow 0 \oplus | \oplus | \rightarrow 0 \rightarrow 1 \\ | \quad | \quad | \quad | \quad | \quad | \\ | \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \end{array}$$

Result - (111011)

Gray to Binary conversion

- Step-1: The most-significant Bit (msB), but for the Binary is the same as that of Gray code.
- Step-2: To obtain the second Binary bit, add the msB to second Gray bit, as counted from msB. Ignore any carry that is generated in the process.
- Step-3: Repeat the process, till the conversion is complete.

Example: Convert the Gray code to binary
11011 to binary.



Result is 10010

Convert the Gray code 1101 to the binary



Result is 1001

Parity Bit!

A Parity Bit is an extra bit that is attached to a code group that is being transmitted from one location to another. The Parity Bit is made either 0 or 1 depending on the number of 1s that are contained in the code group.

Even Parity method!

In this method, the value of the parity bit is chosen so that the total number of 1s in the code group (including Parity Bit) is an even ~~odd~~ number. For example, suppose that the code group is 1000101. This is the ASCII character 'E'. The code group has three 1s. Therefore we will add a parity bit of 1 make the total number of 1s an even number. The new code group including the parity bit thus becomes 11000101.

ODD Parity method!

In this method, the value of the parity bit is ~~is~~ chosen so that the total number of 1s in the code group (including the parity bit) is an odd number. For example, suppose the code group is 1000101. The code group has three 1s. Therefore we will add a parity bit of 0 to keep the total number of 1s an odd number. The new code group including the parity becomes 01000101.

Niranjan Bher

Sr. Lect PKAET, BGPT

LOGIC GATES

Niranjan Boire
Sr. Lect.

- * Logic gates are the basic building block of digital systems.
- * There are 3 basic types of gate AND, OR and NOT.
- * Logic gates are electronic circuits because they are made up of a number of electronic devices and components
- * Inputs and outputs of logic gates can occur only 2 levels. These two levels are termed HIGH and LOW or TRUE and FALSE or ON and OFF or simply 1 and 0
- * The table which lists all the possible combinations of input variables and the corresponding output is called a truth table

EVEL LOGIC!

A logic in which the voltage levels represents logic 1 and logic 0. Level logic may be positive or negative ~~level~~ logic.

POSITIVE LOGIC : — A Positive logic system

is the one in which the higher of the two voltage levels represents the logic 1 and the lower of the two voltages level represents the logic 0.

NEGATIVE LOGIC : — A negative logic system

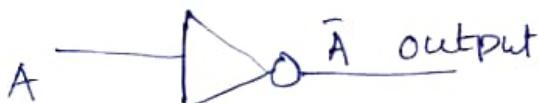
is the one in which the lower of the two voltage levels represents the logic 1 and the higher of the two voltages level represents the logic 0.

The Inverters (not), AND and OR gates are considered to be the basic logic gates. The NAND, NOR, Exclusive OR and Exclusive NOR gates are constructed using these basic logic gates.

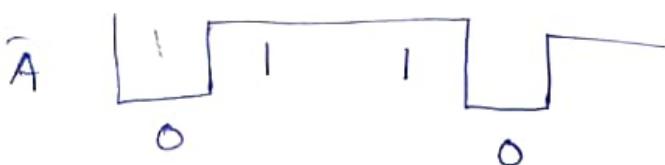
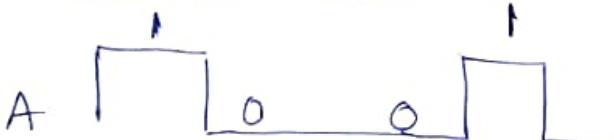
NOT GATE (INVERTER) :

- * A NOT GATE also called inverter, has only one input and one output.
- * It is a device whose output is always the complement of its input.
- * The output of a NOT gate is the logic 1 state when its input is in logic 0 state and the logic 0 state when its input is in logic 1 state.

IC 40 - 7404



Timing Diagram



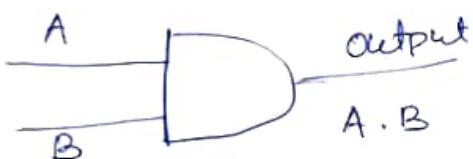
Truth table

INPUT A	OUTPUT \bar{A}
0	1
1	0

AND GATE

- * An AND gate has two or more inputs but only one output.
- * The output is logic 1 state only when each one of its inputs is a logic 1 state.
- * The output is logic 0 state even if one of its inputs is at logic 0 state.

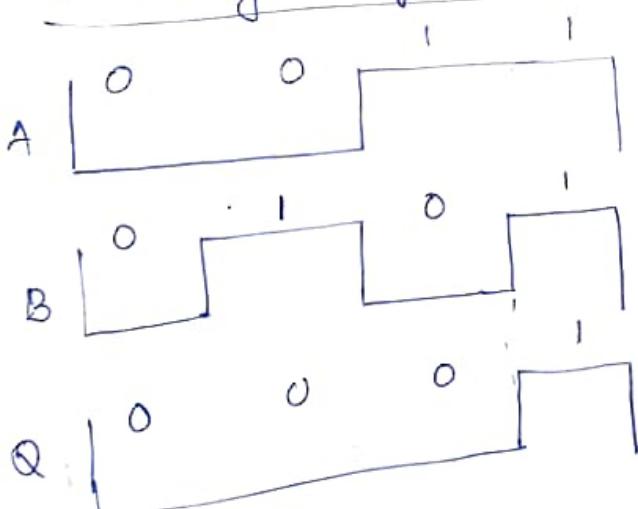
IC NO - 7408



Truth Table

INPUT		OUTPUT
A	B	$Q = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

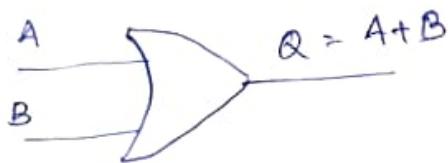
Timing Diagram



OR GATE

- * An OR gate may have two or more inputs but only one output.
- * The output is logic 1 state, even if one of its inputs is in logic 1 state.
- * The output is logic 0 state, only when each one of inputs is in logic 0 state.

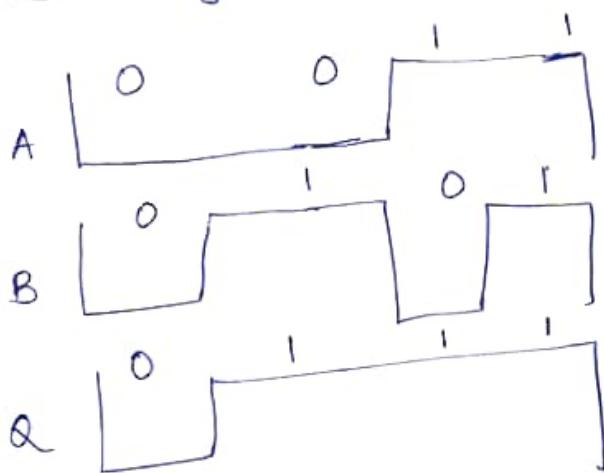
IC NO : - 7432



Truth Table

INPUT		OUTPUT
A	B	$Q = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Timing Diagram



NAND GATE !

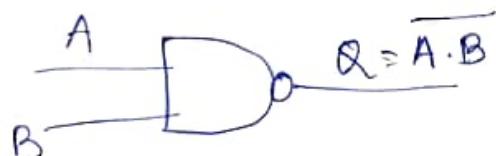
- NAND gate is a combination of an AND gate and NOT gate .
- The output is logic 0 when each of the input is logic 1 and for any other combination of input the output is logic 1.

IC NO - 7400 two input NAND gate

7410 three input NAND gate

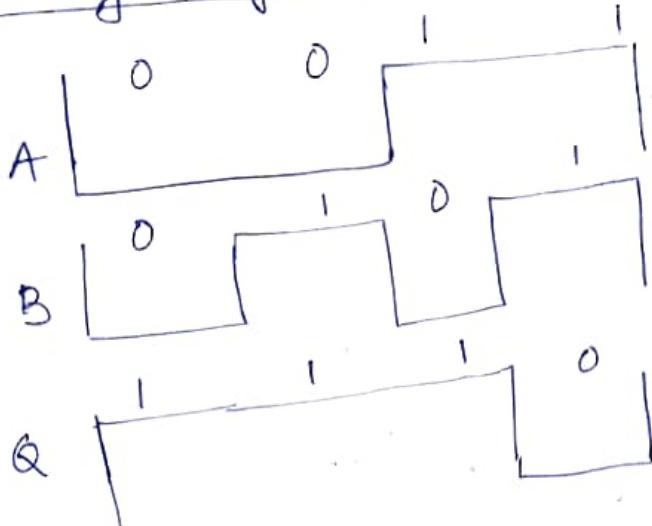
7420 four input NAND gate

7430 eight input NAND gate .



INPUT		output
A	B	$Q = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Timing Diagram



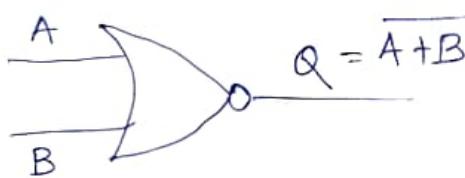
NOR GATE

- * NOR gate is a combination of an OR gate and a NOT gate.
- * The output is logic 1, only when each one of the input is logic 0 and for any other combination of inputs, the output is a logic 0 level.

IC NO - 7402 two input NOR Gate

7427 three input NOR gate

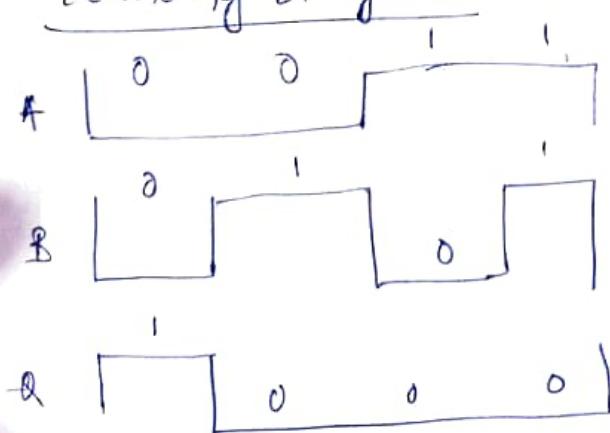
7425 four input NOR gate



Truth Table

INPUT		OUTPUT $Q = \overline{A+B}$
A	B	
0	0	1
0	1	0
1	0	0
1	1	0

Timing Diagram

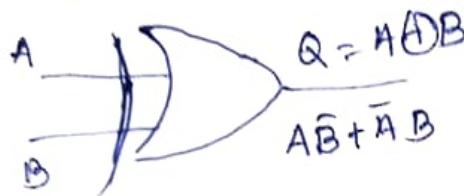


Exclusive-OR (X-OR) Gate

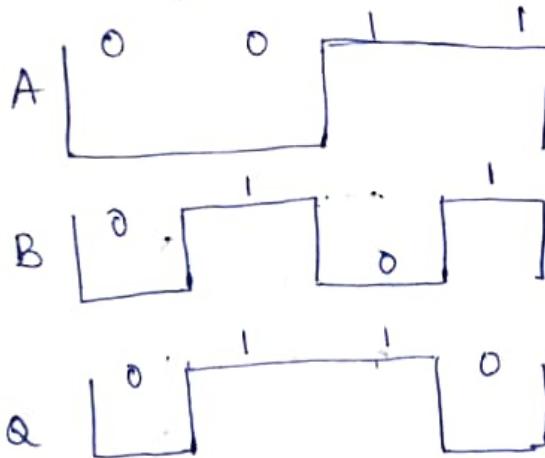
- An X-OR gate is a two input, one output logic gate.
- The output is logic 1 when one and only one of its two inputs is logic 1. When both the inputs is logic 0 or when both input is logic 1, the output is logic 0.

IC 7486

Logic Symbol



Timing Diagram:



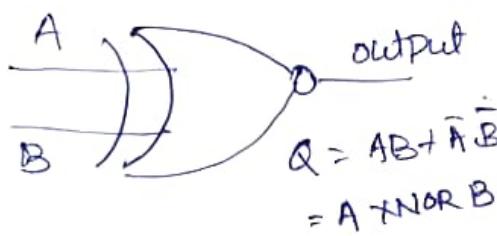
Truth Table

INPUT		OUTPUT
A	B	$Q = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive - NOR (X-NOR) Gate

- * An X-NOR gate is the combination of an X-OR gate and a NOT gate.
- * An X-NOR gate is a two input, one output logic gate circuit.
- * The output is logic 1 only when both the inputs are logic 0 or when both the input is 1.
- * The output is logic 0 when one of the inputs is logic 0 and other is 1.

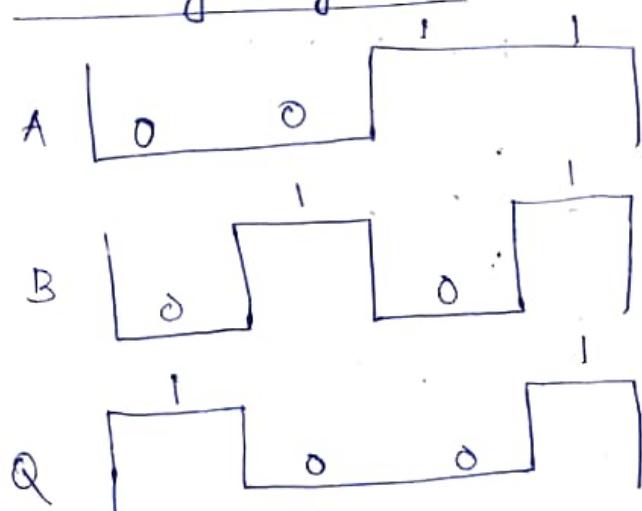
IC NO - 74266



Truth table

INPUT		OUTPUT
A	B	$Q = AB + \bar{A}\bar{B}$
0	0	1
0	1	0
1	0	0
1	1	1

Timing Diagram:



Universal gate :

- * There are three basic gates OR, AND and NOT
- * Two universal gate NAND and NOR.
- * All the basic gates are ~~not~~ realize using the universal gate single.
- * Both NAND and NOR gates can perform all logic functions i.e AND, OR, NOT, EXOR and EXNOR.

NAND Gate

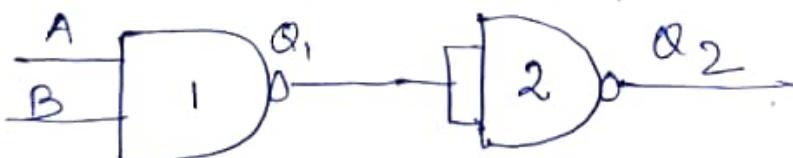
a) Invertor or NOT Gate using NAND gate.



$$\text{INPUT} = A$$

$$\text{Output} = Q = \bar{A}$$

b) AND gate from NAND gate.



$$\text{INPUT} = A \text{ and } B$$

OUTPUT of 1st NAND Gate

$$Q_1 = \overline{A \cdot B}$$

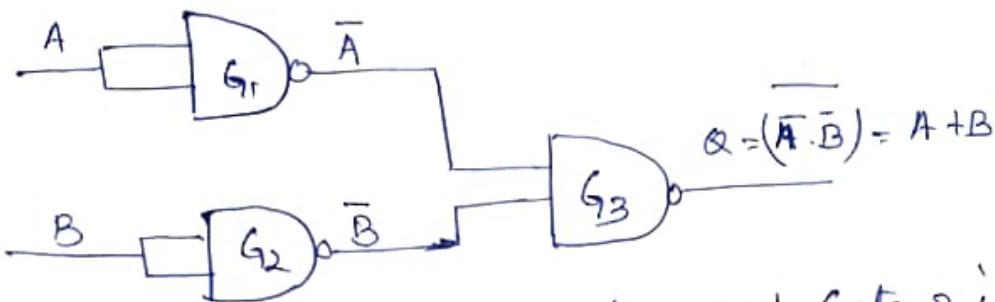
* INPUT to 2nd NAND gate is

$$\bullet \quad \overline{A \cdot B}$$

OUTPUT of 2nd NAND gate is

$$Q_2 = (\overline{A \cdot B}) = AB$$

① OR gate from NAND gate



INPUT of the NAND Gate 1 and Gate 2 is A and B respectively.

OUTPUT of the Gate (G_1) and (G_2) is \overline{A} and \overline{B}

INPUT of the NAND gate (G_3) is \overline{A} and \overline{B}

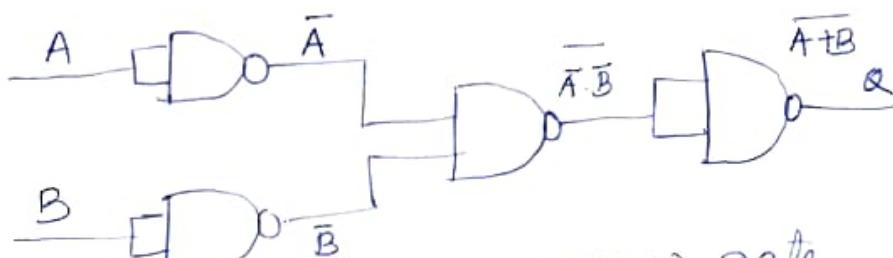
OUTPUT of the NAND gate G_3 is

$$Q = \overline{(\overline{A} \cdot \overline{B})} = (\overline{A}) + (\overline{B}) = A + B$$

(d) NOR gate from NAND gate

Input are A and B

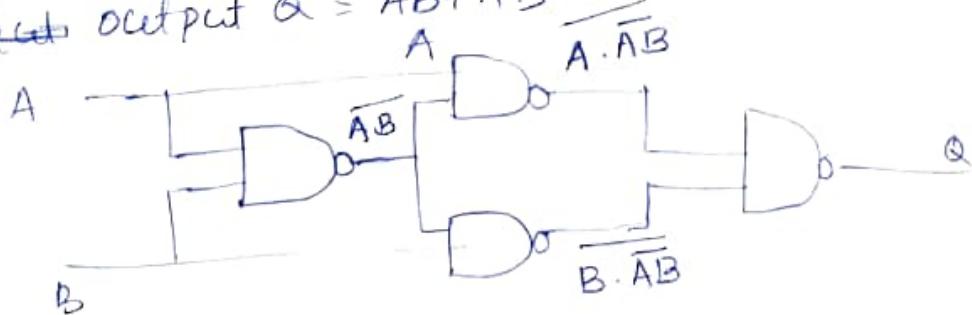
$$\text{output } Q = \overline{A+B}$$



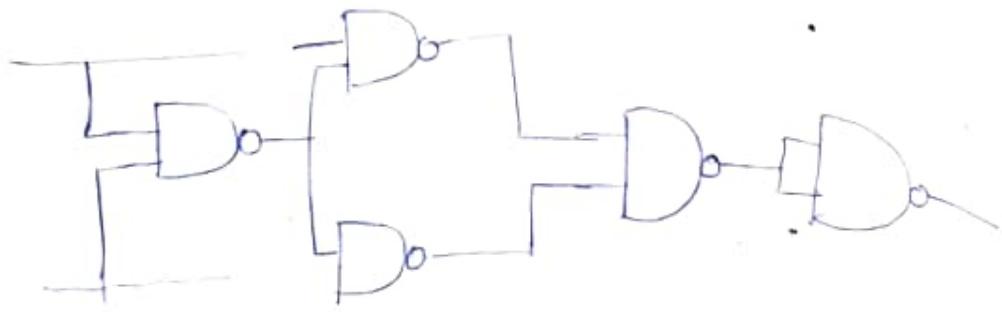
② Ex-OR gate from NAND gate

Input are A and B

~~Output~~ Output $Q = A\bar{B} + \bar{A}B$



U ⑤ Ex-NOR gate from NAND gate

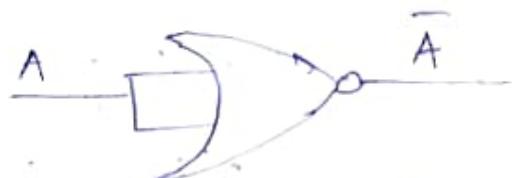


NOR GATE :-

- a) Inverters from NOR gate :

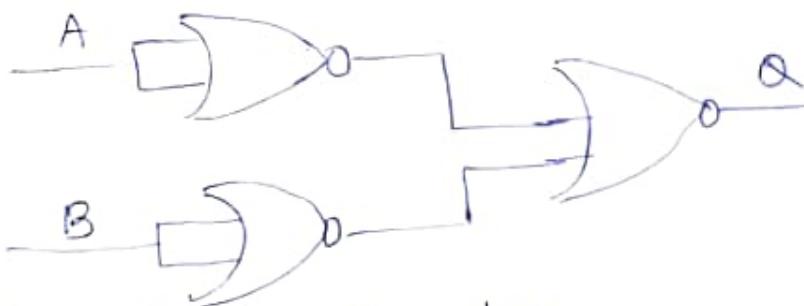
Input = A

Output Q = \bar{A}



N1

- b) AND gate from NOR gate



Input are A and B

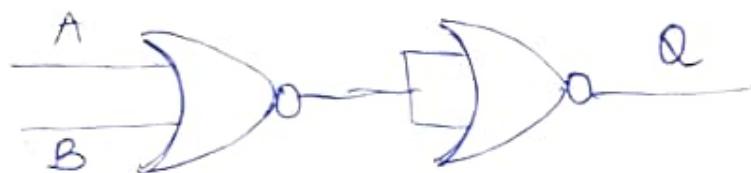
⑥

Output Q = A · B

- c) OR gate from NOR gate

Input are A and B

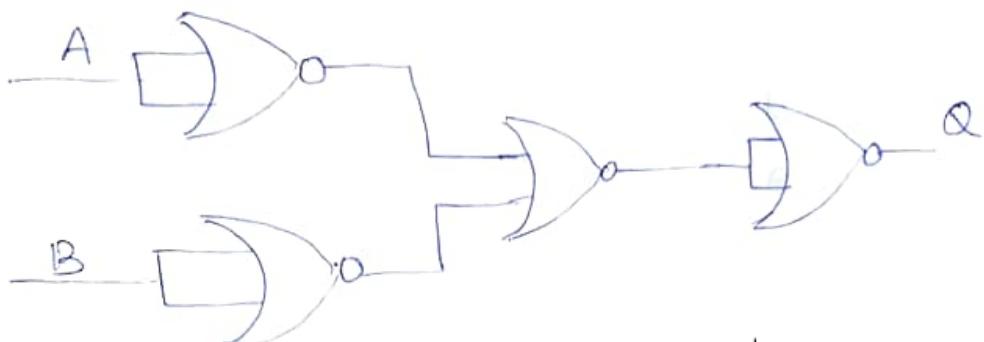
Output Q = A + B



④ NAND gate bmm NOR gate :

Inputs are A and B

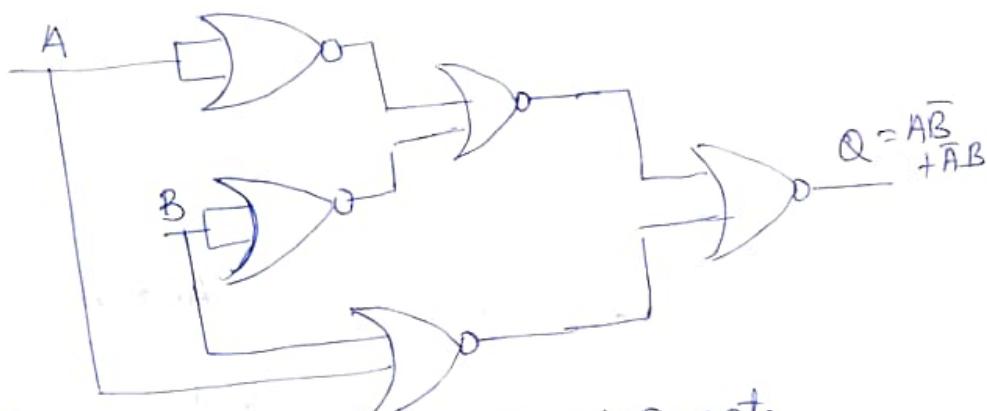
$$\text{output } Q = \overline{A \cdot B}$$



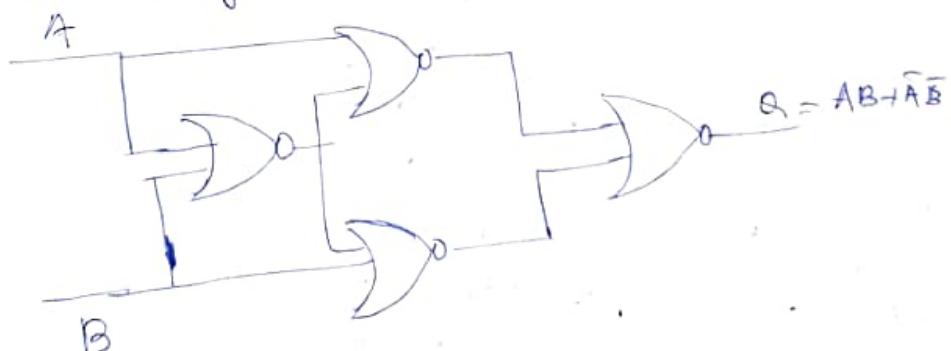
⑤ Ex-OR gate bmm NOR gate

Inputs are A and B

$$\text{output } Q = A\bar{B} + \bar{A}B$$



⑥ Ex-NOR gate bmm NOR gate



Inputs are A and B

$$\text{output } Q = AB + \bar{A}\bar{B}$$

BOOLEAN ALGEBRA

Niranjan
Sr. Lect. B.E.
PKAIE/B.T.

- * Switching Circuits are also called logic circuits, gate circuit and digital circuits.
- * Switching algebra is also called Boolean Algebra.
- * Boolean Algebra is a system of mathematics logic. It is an algebraic system consisting of the set of elements {0,1}, two binary operators called OR and AND and NOT ~~and~~.
- * It is the basic mathematical tool in the analysis and synthesis of switching circuits.
- * It is a way to express logic functions algebraically.
- * Any complex logic can be expressed by one Boolean function.
- * The Boolean algebra is governed by certain well developed rules and laws.

Axioms and Laws of Boolean Algebra:

Axioms or Postulates of Boolean algebra are set of logical expression that are accepted without proofs and upon which we can build a set of useful theorems. Actually, axioms are nothing more than the definitions of the three basic logic operations AND, OR and NOT (Inverter). Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

AND operation

OR operation

NOT operation

- | | | |
|---------------------------|-----------------------|--------------------------|
| Axiom 1 : $0 \cdot 0 = 0$ | Axiom 5 : $0 + 0 = 0$ | Axiom 9 : $\bar{1} = 0$ |
| Axiom 2 : $0 \cdot 1 = 0$ | Axiom 6 : $0 + 1 = 1$ | Axiom 10 : $\bar{0} = 1$ |
| Axiom 3 : $1 \cdot 0 = 0$ | Axiom 7 : $1 + 0 = 1$ | |
| Axiom 4 : $1 \cdot 1 = 1$ | Axiom 8 : $1 + 1 = 1$ | |

1. Complementation Laws :-

The term complement simply means to invert i.e. to changes 0 to 1 and 1 to 0. The five laws of complementations are as follows:

$$\text{Law 1} : \bar{0} = 1$$

$$\text{Law 2} : \bar{1} = 0$$

$$\text{Law 3} : \text{if } A = 0 \text{ then } \bar{A} = 1$$

$$\text{Law 4} : \text{if } A = 1 \text{ then } \bar{A} = 0$$

$$\text{Law 5} : \bar{\bar{A}} = A \text{ (double complementation law)}$$

2. OR Laws :-

The four OR laws are as follows

$$\text{Law 1} : A + 0 = A \text{ (Null law)}$$

$$\text{Law 2} : A + 1 = 1 \text{ (Identity law)}$$

$$\text{Law 3} : A + A = A$$

$$\text{Law 4} : A + \bar{A} = 1$$

3. AND Laws!

The four AND laws are as follows:-

$$\text{Law 1} : A \cdot 0 = 0 \text{ (Null law)}$$

$$\text{Law 2} : A \cdot 1 = A \text{ (Identity law)}$$

$$\text{Law 3} : A \cdot A = A$$

$$\text{Law 4} : A \cdot \bar{A} = 0$$

4. Commutative Laws:

Commutative laws allow change in position of AND or OR variables. There are two commutative laws.

$$\text{Law 1: } A + B = B + A$$

Proof

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

$$\text{Law 2: } A \cdot B = B \cdot A$$

Proof

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$B \cdot A$
0	0	0
0	1	0
1	0	0
1	1	1

This law can be extended to any number of variables. For example

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

5. Associative Laws

The associative laws allow grouping of variables. There are 2 associative laws.

$$\text{Law 1: } (A + B) + C = A + (B + C)$$

Proof

A	B	C	$A \cdot B$	$(A + B) + C$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

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

Law 2: $(A \cdot B)C = A(B \cdot C)$

Proof

A	B	C	AB	$(A \cdot B) \cdot C$
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	1	0
1	1	1	1	1

A	B	C	BC	$A(B \cdot C)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

This law can be extended to any number of variables. For example
 $A(BCD) = (ABC)D = (AB)(CD)$

6. Distributive Laws: -

The distributive laws allow factoring or multiplying out of expressions. There are two distributive laws.

$$\text{Law 1: } A(B+C) = AB + AC$$

Proof

A	B	C	$B+C$	$AC(B+C)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	AB	AC	$AB+AC$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

Law 2: $A + BC = (A+B)(A+C)$

Proof RHS $(A+B)(A+C)$

$$\begin{aligned} &= A \cdot A + A \cdot C + B \cdot A + B \cdot C \\ &= A + AC + AB + BC \\ &= AC(1+C+B) + BC \\ &= A \cdot 1 + BC \quad (1+C+B = 1+B=1) \\ &= A + BC = \text{LHS} \end{aligned}$$

7. Redundant Literal Rule (RLR)

Law 1: $A + \bar{A}B = A + B$

Proof $A + \bar{A}B = (A+\bar{A})(A+\bar{B})$

$$\begin{aligned} &= 1 \cdot (A+B) = A+B \end{aligned}$$

Law 2: $A(\bar{A}+B) = AB$

$$\begin{aligned} A(\bar{A}+B) &= A \cdot \bar{A} + A \cdot B \\ &= 0 + AB = AB \end{aligned}$$

8. Idempotence Laws:

Law 1: $A \cdot A = A$

Proof If $A = 0$ then $A \cdot A = 0 \cdot 0 = 0 = A$

If $A = 1$ then $A \cdot A = 1 \cdot 1 = 1 = A$

This law states that AND of a variable with itself is equal to that variable only.

Law 2: $A + A = A$

Proof If $A = 0$ then $A + A = 0 + 0 = 0 = A$

If $A = 1$ then $A + A = 1 + 1 = 1 = A$

This law states that OR of a variable with itself is equal to that variable only.

9. Absorption Laws! -

Law 1 $A + A \cdot B = A$

Proof $A + A \cdot B = A(1+B) = A \cdot 1 = A$

Law 2 $A(A+B) = A$

Proof $A(A+B) = A \cdot A + A \cdot B$

$$= A + A \cancel{\cdot} A \cdot B$$

$$= A(1+B) = A \cdot 1 = A$$

10. Consensus theorem (Included Factor theorem)

Theorem 1: $AB + \bar{A}C + BC = AB + \bar{A}\bar{C}$

LHS $AB + \bar{A}C + BC$

$$= AB + \bar{A}C + BC(A + \bar{A})$$

$$= AB + \bar{A}C + BCA + BC\bar{A}$$

$$= AB + BCA + \bar{A}C + BC\bar{A}$$

$$= ABC(1+C) + \bar{A}C(1+B)$$

$$= AB(1) + \bar{A}C(1)$$

$$= AB + \bar{A}C = RHS$$

Theorem 2: $(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$

LHS $= (A+B)(\bar{A}+C)(B+C)$

$$= (\bar{A}A + AC + B\bar{A} + BC)(B+C)$$

$$= (AC + BC + B\bar{A})(B+C)$$

$$= ACB + BC + \bar{A}B + AC + BC + \bar{A}BC$$

$$= ABC + \bar{A}BC + BC + BC + \bar{A}B + AC$$

$$= BC + BC + BC + \bar{A}B + AC$$

$$= A(C + BC + \bar{A}B)$$

$$RHS = (A+B)(\bar{A}+C)$$

$$= A\bar{A} + AC + B\bar{A} + BC$$

$$\therefore AC + B\bar{A} + BC = AC + BC + \bar{A}B = LHS$$

11. Transposition Theorem:

$$\text{Theorem: } AB + \bar{A}C = (A+C)(\bar{A}+B)$$

$$RHS = (A+C)(\bar{A}+B)$$

$$= A \cdot \bar{A} + AB + C\bar{A} + CB$$

$$= 0 + \bar{A}C + AB + BC$$

$$= \bar{A}C + AB + BC(A+\bar{A})$$

$$= \bar{A}C + AB + ABC + \bar{A}BC$$

$$= AB + ABC + \bar{A}C + \bar{A}BC$$

$$= AB(1+C) + \bar{A}C(1+B)$$

$$= AB + \bar{A}C = LHS$$

12. De Morgan's Theorem:

De Morgan's theorem represents two law in Boolean Algebra.

$$\text{Law 1: } \overline{A+B} = \bar{A} \cdot \bar{B}$$

Proof:

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

A	B	\bar{A}	\bar{B}	$\bar{A}\bar{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

This Law states that the complement of a sum of variables is equal to the product of their individual complements.

Law-2 $\overline{A \cdot B} = \overline{A} + \overline{B}$

Products		$A \cdot B$		$\overline{A} + \overline{B}$				
A	B	$A \cdot B$	$\overline{A \cdot B}$	A	B	\overline{A}	\overline{B}	$\overline{A} + \overline{B}$
0	0	0	1	0	0	1	1	1
0	1	0	1	0	1	1	0	1
1	0	0	1	1	0	0	1	1
1	1	1	0	1	1	0	0	0

This law states that the complement of a product of variables is equal to the sum of their individual complements.

Duality:

The implications of the duality law left is that once a theorem or statement is proved, the dual also thus stand proved. This is called the principle of duality.

Sum-of Product form (SOP)

This is also called disjunctive canonical form (DCF) or Expanded sum of products form or Canonical sum of product form.

In this form, the function either is the sum of a number of products term where each product term contains all variables of the function either in complemented or uncomplemented value.

This can also be derived from the truth table by finding the sum of all the terms that corresponds to those combinations for which 'f' assumes the value 1.

$$\text{example } f(A, B, C) = \bar{A}B + \bar{B}C$$

$$= \bar{A}B(C + \bar{C}) + \bar{B}C(A + \bar{A})$$

$$= \bar{A}BC + \bar{A}\bar{B}C + A\bar{B}C + \bar{A}\bar{B}\bar{C}$$

→ The product term which contains all the variables of the functions either in complemented or uncomplemented form is called a minterm.

→ The minterm is denoted as $m_0, m_1, m_2 \dots$

→ An n-variable function can have 2^n minterms.

→ Another way of representing the function in canonical SOP form is the showing the sum of minterms for which the function equals to 1.

$$f(A, B, C) = m_1 + m_2 + m_3 + m_5$$

or

$$f(A, B, C) = \sum m(1, 2, 3, 5)$$

$\sum m$ = Sum of all minterms.

Product-of-sums form: (POS)

→ This form is also called as conjunctive canonical form (CCF) or Expanded Product of Sums form or Canonical Product of Sums form.

→ This is by considering the combination for which $f = 0$

$$\rightarrow \text{This function } f(A, B, C) = (\bar{A} + \bar{B} + C\bar{C})(A + B + C\bar{C}) \\ = (\bar{A} + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})(A + B + C)(A + B + \bar{C})$$

- The sum term which contains each of the n variables in either complemented or uncomplemented form is called maxterm.
- Maxterm is represented as m_0, m_1, m_2

$$f(A, B, C) = m_0 \cdot m_1 \cdot m_2 \cdot m_7$$

or
 $f(A, B, C) = \prod(0, 1, 6, 7)$

Conversion between canonical form:-

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.

Expt $f(A, B, C) = \sum m(0, 2, 4, 6, 7)$

Complement
 $f(\overline{A}, \overline{B}, \overline{C}) = \sum m(1, 3, 5)$

$$= m_1 + m_3 + m_5$$

If we complement by De-morgan's theorem we obtain f' in a form.

$$\begin{aligned} f' &= (\overline{m_1} + \overline{m_2} + \overline{m_5}) = \overline{m_1} \cdot \overline{m_2} \cdot \overline{m_5} \\ &= m_1 m_2 m_5 = \prod M(1, 3, 5) \end{aligned}$$

Expt Expand $A(\bar{A}+B)(\bar{A}+B+\bar{C})$ to maxterms and minterms.

In POS form

$$A(\bar{A}+B)(\bar{A}+B+\bar{C})$$

$$A = A + B\bar{B} + C\bar{C}$$

$$= (\underline{A+B})(A+\bar{B}) + C\bar{C}$$

$$= (A+B+C\bar{C})(A+\bar{B}+C\bar{C})$$

$$= (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C})$$

$$\begin{aligned} (\bar{A}+B) &= (\bar{A}+B+C\bar{C}) \\ &= (\bar{A}+B+C)(\bar{A}+B+\bar{C}) \end{aligned}$$

Therefore

$$\begin{aligned} & A(\bar{A}+B)(\bar{A}+B+\bar{C}) \\ &= (\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C}) \\ &\quad (\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+B+\bar{C}) \\ &= (\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C}) \\ &\quad (\bar{A}+B+C)(\bar{A}+B+\bar{C}) \\ &= (000)(001)(010)(011)(100)(101) \end{aligned}$$

$$= m_0 m_1 m_2 m_3 m_4 m_5$$

$$= \cancel{\prod} m(0, 1, 2, 3, 4, 5)$$

The maxterms m_6 and m_7 are missing in

the POS Form
so the SOP form will contain the minterms 6 and 7.

Karnaugh map OR K-map :

→ The K-map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.

→ The K-map is systematic method of simplifying the Boolean expression

Two variable K-map! -

A two variable expression can have $2^2 = 4$ possible combinations of input variables A and B.

Mapping of SOP Expression! —

- * The 2 variable K-map has $2^2 = 4$ squares.

These squares are called cells.

- * A '1' is placed in any square indicates that corresponding minterm is included in the output expression, and 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

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

Ex: Map expression $f = \bar{A}B + A\bar{B}$

SOL? The expression minterm is

$$f = m_1 + m_2 = m(1, 2)$$

	B	0	1
A	0	0 0	1 1
	1	1 2 0	3

Minimization of SOP expression! -

To minimize a Boolean expression given in the SOP form by using K-map, the adjacent squares having 1's that is minterms adjacent to each other are combined to form larger squares to eliminate some variables. The possible minterm grouping in two variables K-map are shown below.

	B	\bar{B}	B
A	1	1	
\bar{A}			
A	0	0	

$$f_1 = \bar{A}$$

	B	\bar{B}	B
A	1	0	
\bar{A}			
A	1	0	

$$f_2 = \bar{B}$$

	B	\bar{B}	B
A	0	1	
\bar{A}			
A	0	1	

$$f_3 = B$$

	B	\bar{B}	B
A	0	0	
\bar{A}			
A	1	1	

$$f_4 = A$$

	B	\bar{B}	B
A	1	1	
\bar{A}			
A	1	1	

$$f_5 = 1$$

- * Two minterms, which are adjacent to each other, can be combined to form a bigger square called 2-square or a pair. This square eliminates one variable that is not common to both the minterms.

- * Two 2-squares adjacent to each other can be combined to form a 4-square. A 4-square eliminates 2 variables. A 4-square is called a quad.

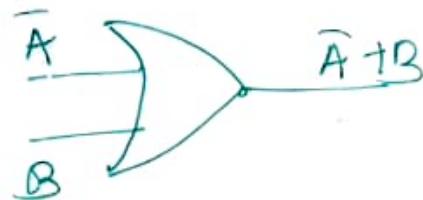
Exap Reduce the expression $f = \bar{A}\bar{B} + \bar{A}B + A\bar{B}$ using mapping.

Solⁿ! - Expressed in terms of minterms, the given expression is

$$f = m_0 + m_1 + m_3 = \Sigma m (0, 1, 3)$$

	B	\bar{B}	B
A	1	1	1
\bar{A}	0	1	1

$$f = \bar{A} + B$$



Mapping of POS Expression!

A function in two variable (A, B) has 4 possible minterms, $A+B$, $A+\bar{B}$, $\bar{A}+B$ and $\bar{A}+\bar{B}$.

	B	0	1
A	0	$A+B$ M ₀	$A+\bar{B}$ M ₁
\bar{A}	1	$\bar{A}+B$ M ₂	$\bar{A}+\bar{B}$ M ₃

Example Plot the expression $f = (A+B)(\bar{A}+B)(\bar{A}+\bar{B})$

Expression in term of maxterm

$$f = \prod M (0, 2, 3)$$

	B	0	1
A	0	0	1
\bar{A}	1	0	0

Minimization of POS Expression:

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

$$f_1 = A$$

$$f_2 = \bar{B}$$

$$f_3 = B$$

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

$$f_4 = \bar{A}$$

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

$$f_5 = 0$$

Three variable K-map!

A function in three variables (A, B, C) can be expressed in SOP and POS form having eight possible combination.

		BC	$\bar{B}C$	$B\bar{C}$	$\bar{B}\bar{C}$
		0	1	3	2
		4	5	7	6
A	0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}B\bar{C}$	$\bar{A}B\bar{C}$
A	1	$A\bar{B}\bar{C}$	$A\bar{B}C$	$AB\bar{C}$	$AB\bar{C}$

a) Minterm

		BC	$\bar{B}C$	$B\bar{C}$	$\bar{B}\bar{C}$
		00	01	11	10
		M ₀	M ₁	M ₃	M ₂
A	0	$A+B+C$	$A+B+\bar{C}$	$A+\bar{B}+\bar{C}$	$A+\bar{B}+C$
A	1	$\bar{A}+B+C$	$\bar{A}+B+\bar{C}$	$\bar{A}+\bar{B}+\bar{C}$	$\bar{A}+\bar{B}+C$

Example: —
Map the expression $f = \bar{A}\bar{B}C + A\bar{B}C + ABC + ABC$

Sol: SOP form expression
 $f = \Sigma m(1, 5, 6, 7)$

BC		00	01	11	10	2
A		0	1	0	0	2
		0	4	5	7	6
		0	1	1	1	1
		1	0	1	1	0

$f = \bar{B}C + AB$

Example:

map the expression
 $f = (A+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+\bar{C})$
 $(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$

POS form expression is
 $f = \prod m(0, 5, 7, 3, 6)$

BC		00	01	11	10	2
A		0	1	0	1	2
		0	4	5	7	6
		0	1	0	1	0
		1	0	1	0	0

Minimization of SOP and POS Expressions:

Steps are:

- * Draw the K-map and place 1's (0's) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- * In the map 1's (0's) which are not adjacent to any other 1's (0's) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
- * For those 1's (0's) which are adjacent to only one another 1's (0's) make them part of 2-squares.
- * For quads (4-squares) and octet (8-squares) which have already been combined. They must which have already been combined. They must be geometrically from a squares or a rectangle if they have not been combined yet then combine them into bigger squares of possible.
- * From the minimal expression by summing multiplying the product (sum) terms of all the groups.

A	00	01	11	10
0	1	1	0	1
1	1	0	1	0
0	0	1	0	1

$$f = \bar{B} + \bar{A}B$$

$$f = \bar{A}B + A\bar{B} + AB$$

Four Variable K-map

A four variable (A, B, C, D) expression can have $2^4 = 16$ possible combinations of input variables. A four variable K-map has $2^4 = 16$ squares or cells and each square on the map represents either a minterm or a maxterm.

SOP Form

		CD	00	01	11	10
		AB	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B\bar{C}\bar{D}$
CD	AB	00	m_0	m_1	m_3	m_2
		01	m_4	m_5	m_7	m_6
CD	AB	11	m_{12}	m_{13}	m_{15}	m_{14}
		10	m_8	m_9	m_{11}	m_{10}

POS Form

		CD	00	01	11	10
		AB	$A+B+C+D$ (M ₀)	$A+B+C+\bar{D}$ (M ₁)	$A+B+\bar{C}+\bar{D}$ (M ₃)	$A+B+\bar{C}+D$ (M ₂)
CD	AB	00	0	1	3	2
		01	4 (M ₄)	5 (M ₅)	7 (M ₇)	6 (M ₆)
CD	AB	11	12 (M ₁₂)	13 (M ₁₃)	15 (M ₁₅)	14 (M ₁₄)
		10	8 $\bar{A}+B+C+D$ (M ₈)	9 $\bar{A}+B+C+\bar{D}$ (M ₉)	11 $\bar{A}+B+C+\bar{D}$ (M ₁₁)	10 $\bar{A}+B+C+D$ (M ₁₀)

Minimization of SOP and POS Expression!

For reducing the Boolean expressions in SOP(POS) form the following steps are given below.

- * Draw the K-map and P.I.s (SOP) corresponding to the minterms (maxterms) of the SOP(POS) expression.
- * In the map P.I.s (SOP) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they cannot be combined even into a 2-square.
- * For those P.I.s (SOP) which are adjacent to only one other 1(0) make them Pairs (2-squares).
- * For quads (4-squares) and octet (8-squares) of adjacent P.I.s (SOP) even if they contain some P.I.s (SOP) which have already been combined. They must geometrically form a square or a rectangle.
- * For any P.I.s (SOP) that have not been combined yet then combine them into bigger squares it is possible.
- * Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

Example :-

Reduce using mapping the expression

$$f = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$$

The given expression in POS form is

$$f = \prod M(4, 6, 11, 14, 15)$$

In SOP form

$$f = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$$

		CD	00	01	11	10	
AB		00	1	2	3	4	
		01	5	7	6	D	
		11	12	13	15	14	
		10	8	9	11	10	

$$f = \bar{A}\bar{B} \bar{D} + \bar{A}\bar{C} + \bar{B}\bar{D}$$

		CD	00	01	11	10	
AB		00	0	1	3	2	
		01	4	5	7	6	
		11	12	13	15	14	
		10	8	9	11	10	

$$f = \prod M(4, 6, 11, 14, 15)$$

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

The minimal SOP expression is

$$f_{\min} = \bar{B}\bar{D} + A\bar{C} + \bar{A}D$$

The minimal POS expression is

$$f_{\max} = (A + \bar{B} + D) (A + C + D) \\ (A + \bar{B} + \bar{C})$$

DON'T CARE COMBINATIONS :-

The combinations for which the values of the expression are not specified are called don't care combinations or optional combinations and such expression stand incompletely and such expression stand incompletely specified. The output is a don't care for these invalid combinations. The don't care terms are denoted by d or x. During the process of designing using SOP maps, each don't care is treated as 1 to reduce the map otherwise it is treated as 0 and left alone. During the process of designing using POS maps, each don't care is treated as 0 to reduce the map otherwise it is treated as 1 and left alone.

Example

Reduce the expression

$$f = \sum m(1, 5, 6, 12, 13, 14)$$

+ d(2, 4) using k-map

In SOP form $f = \sum m(1, 5, 6, 12, 13, 14) + d(2, 4)$

In POS form $f = \prod m(0, 2, 3, 4)$

$$f = \prod m(0, 3, 7, 8, 9, 10, 11, 15) + d(2, 4)$$

SOP Form

		CD	00	01	11	10
		AB	00	01	11	10
		00	0	1	3	x
		01	x	1	7	6
		11	4	5	15	14
		10	12	13	11	10

$$f_{\min} = B\bar{C} + B\bar{D} + \bar{A}\bar{C}D$$

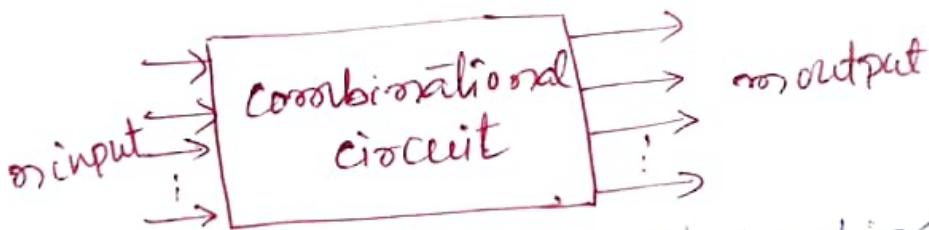
POS Form

		CD	00	01	11	10
		AB	00	01	11	10
		00	0		0	x
		01	x	5	0	6
		11	4	13	15	14
		10	12	0	11	10

$$f_{\max} = (B+D) + (\bar{A}+B) + (\bar{C}+\bar{D})$$

Combinational logic circuit

- * A combinational circuit consists of logic gates whose outputs at any instant time are determined by only the present combination of inputs.
- * A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.
- * It consists of an interconnection of logic gates.



Block diagram of combinational circuit

- * The minput binary variable comes from an external source, the moutput variables are produced by the internal combinational logic circuit and go to an external destination.
- * Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents log₂ 1 and log₂ 0.

Binary adder - Subtractor :-

- * Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations.
- * The most basic arithmetic operation is the addition of two binary digits. The simple addition consists of four possible elementary operations:
 $0+0=0$, $0+1=1$, $1+0=1$, $1+1=10$
- * The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary consists of two digits. The higher significant bit of this result is called carry.
- * A combinational circuit that performs the addition of two bits is called a half adder.
- * One that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

Half-adder :-

- * The circuit needs two binary inputs and two binary outputs.
- * The input variables designate the augend and addend bits. The output variables produce the sum and carry. Symbols x and y are assigned to the two inputs and s (sum) and c (carry) to the outputs.

Truth table of Adder

		Output	
x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

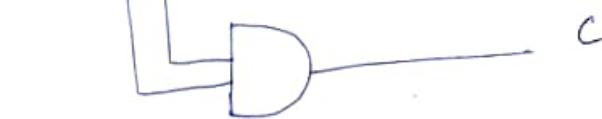
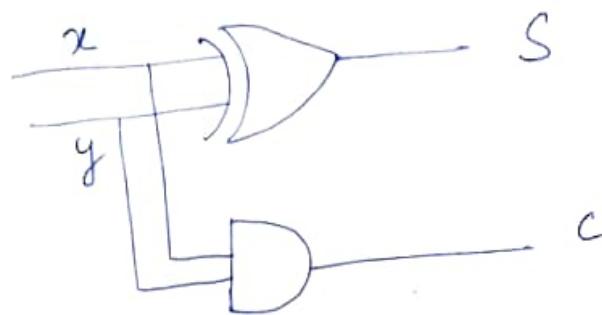
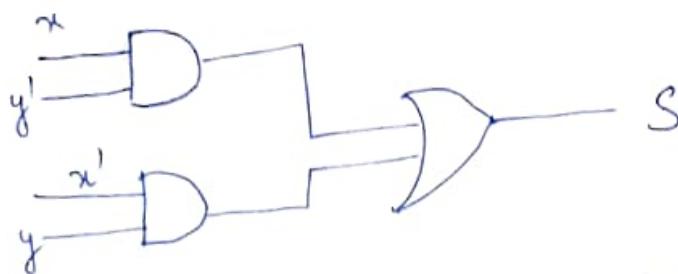
The carry output (c) is 1 only when both inputs are 1.

The simplified Sum-of Product expressions are

$$S = \bar{x}y + x\bar{y}$$

$$C = xy$$

It can be implemented with an exclusive-OR and an AND gate.



Full adder

- * A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- * It consists of three inputs and two outputs. Two of the input variables denoted by x and y , represents the two significant bits to be added. The third input z , represents the carry from the previous lower significant position.

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

* Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3 and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols ~~s and c~~ for sum and c for carry.

		m ₄			
		00	01	11	10
x		m ₀	m ₁	m ₃	m ₂
0			1		1
z	1	m ₄	m ₅	m ₇	m ₆

$$S = \bar{x}\bar{y}z + \bar{x}yz' + xy'z + xyz$$

		m ₄			
		00	01	11	10
x		m ₀	m ₁	m ₃	m ₂
0				1	
z	1	m ₄	m ₅	m ₇	m ₆

$$C = \bar{xy} + xz + \bar{yz}$$

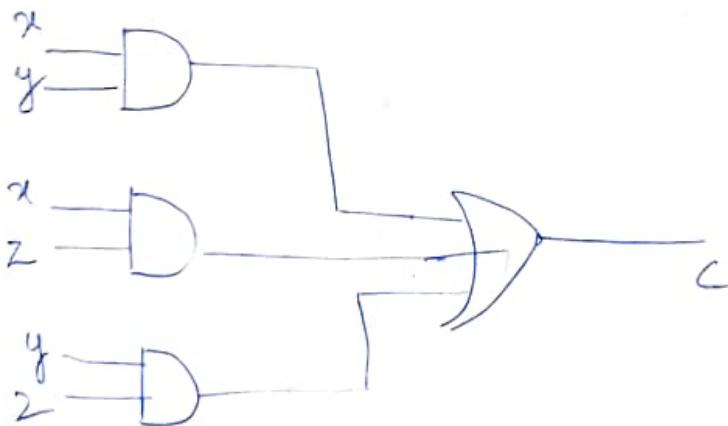
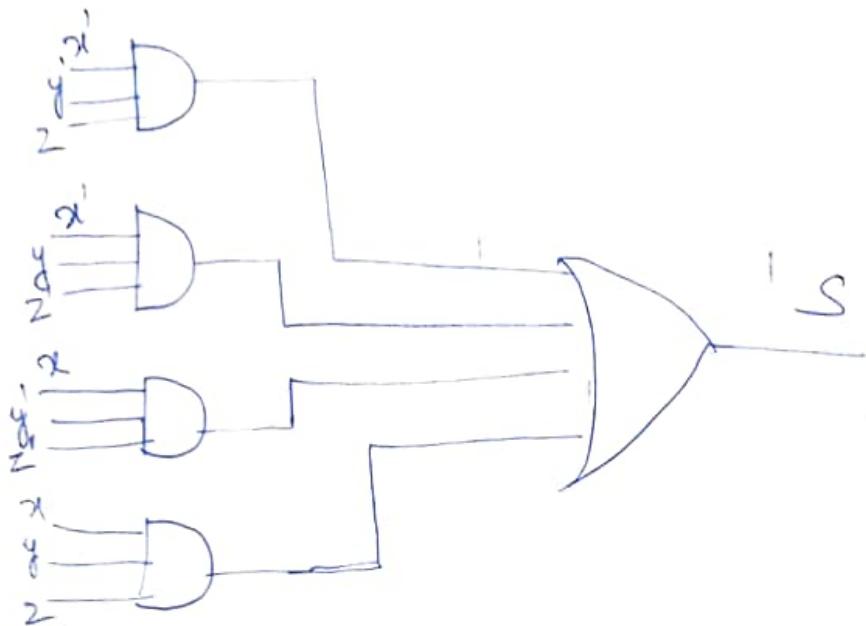
* The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry formed by adding the input carry and the bits of the words.

* The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.

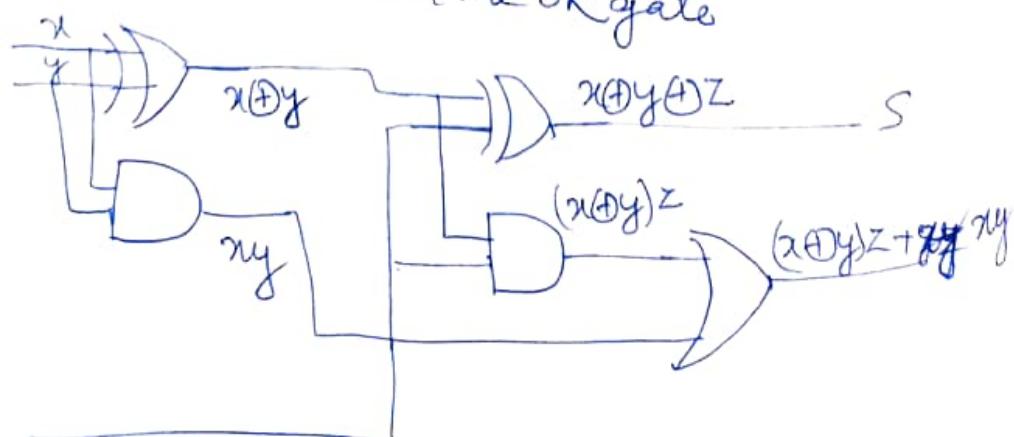
$$S = xy'z + xyz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

Logic diagram for the full adder implemented in sum-of-products form is -



It can also be implemented with two half adders and one OR gate



Binary adder

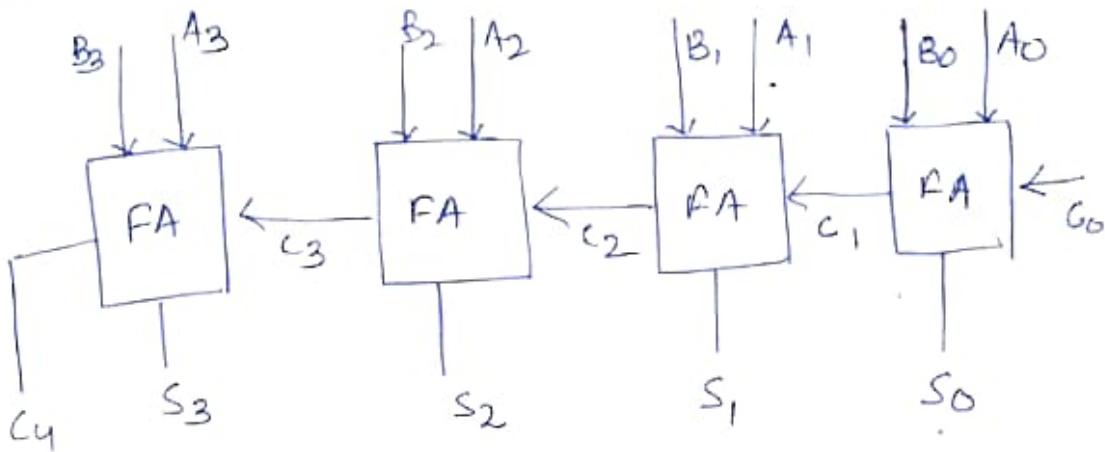
- * A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- * It can be constructed with full adder connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- * Addition of n -bit numbers requires a chain of n full adders or a chain of one half-adder and $n-1$ full adders.

Four full adder (FA) circuits to provide a four-bit binary ripple carry adder.

- * The augend bits of A and the addend bits of B are designated by subscript numbers from right to left with subscript 0 denoting the least significant bit.
- * The carry are connected in a chain through the full adders. The input carry to the adder is C_0 and it ripples through the full adders to the output carry C_4 . The S outputs generates the required sum bit.
- * A n -bit adder requires n full adder with each output carry connected to the input carry of the next higher order full adder.
- * consider the two binary numbers $A = 1011$ and $B = 0011$, Their sum = 1110 is formed with the four bit adder as follows:

	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

- * The bits are added with full adders, starting from the least significant position (Subscript 0), to form the sum bit and carry bit. The input carry C_0 in the least significant position must be 0.
- * The value of C_{i+1} in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left.
- * The sum bits are thus generated starting from the right most position and are available as soon as the corresponding previous carry bit is generated. All the carries must be generated for the correct sum bits to appear at the outputs.



Four bit Binary adder

Half SUBTRACTOR:

- * This circuit needs two binary inputs and two binary outputs.
- * Symbols x and y are assigned to the two inputs and D (for difference) and B (for borrow) to the outputs.
- * The truth table for the half subtractor

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

- * The B output is 1 only when the inputs are 0 and 1. The D output represents the least significant bit of the subtraction.
- * The subtraction operation is done by using the following rules as

$$0 - 0 = 0$$

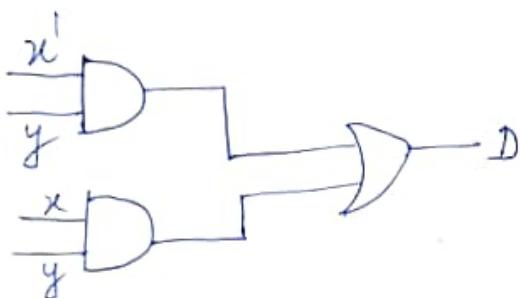
$$0 - 1 = 0 \text{ with borrow 1}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are :

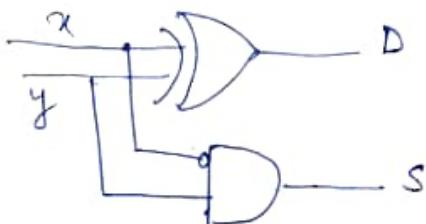
$$D = \bar{x}y + xy' \text{ and } B = \bar{x}y$$



$$\begin{aligned} & \bar{x} \\ & \bar{y} \end{aligned}$$

$$D = \bar{x}y + xy'$$

$$B = \bar{x}y$$



Full Subtractor:

- * A full subtractor is a combinational circuit that performs the arithmetic subtraction operation of three bits.
- * It consists of three inputs and two outputs. Two of the inputs variables, denoted by x and y represent the two significant bits to be subtracted. The third input z is subtracted from the result of the first subtraction.

<u>x</u>	<u>y</u>	<u>z</u>	<u>D</u>	<u>B</u>
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

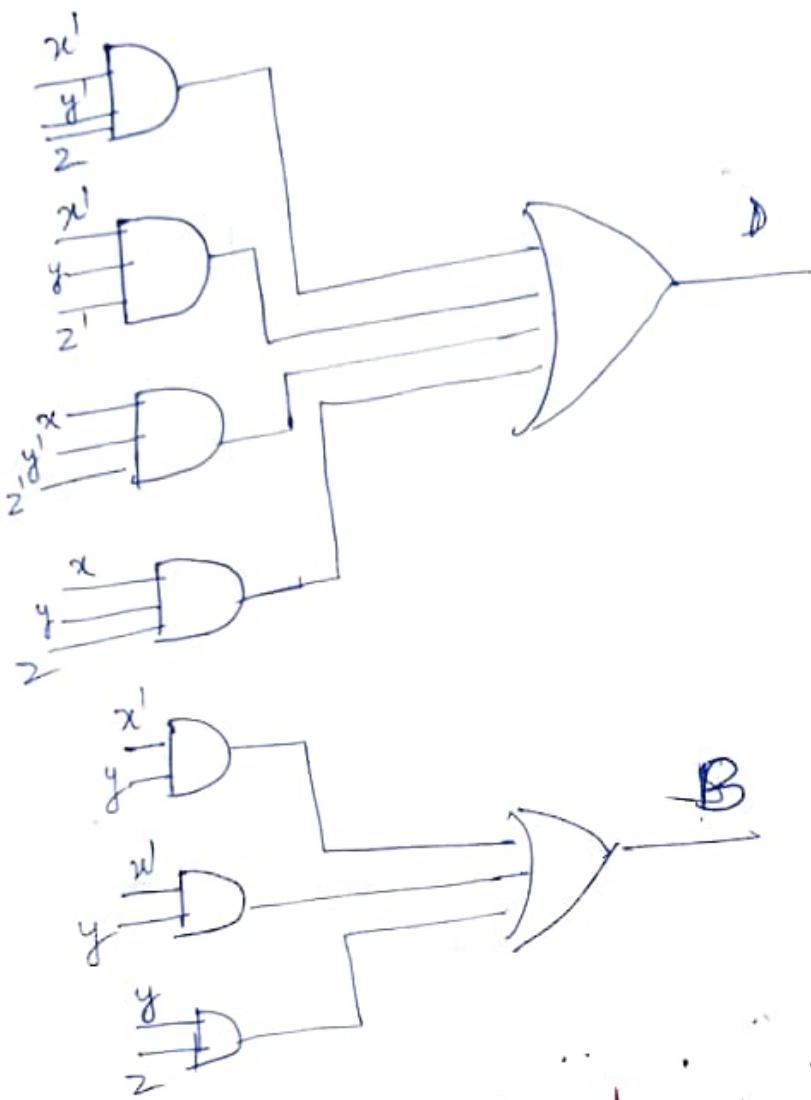
The binary variable D gives the value of the least significant bit of the difference. The binary variables B gives the output borrow needed during the subtraction process.

<u>xz</u>	<u>00</u>	<u>01</u>	<u>11</u>	<u>10</u>
<u>y</u>	0	1	1	1
<u>z</u>	0	1	1	1

$$D = \bar{x}y'z + x'y'z + xy'z + xyz$$

<u>xz</u>	<u>00</u>	<u>01</u>	<u>11</u>	<u>10</u>
<u>y</u>	0	1	1	1
<u>z</u>	0	1	1	1

$$B = \bar{x}'z + \bar{x}'y + yz$$



Magnitude comparators

- * A magnitude comparator ~~circuit~~ is a combinational circuit that compares two numbers A and B determines their relative magnitude.
- * Comparison of three binary variables that indicates $A < B$, $A = B$, or $A > B$.
- * Consider two numbers A and B with two digits each. Now writing the coefficients of the numbers in descending order of significance

$$A = A_1 A_0$$

$$B = B_1 B_0$$

- * The two numbers are equal if all pairs of significant digits are equal i.e. if and only if $A_1 = B_1$ and $A_0 = B_0$.

- * When the numbers are binary, the digits are either 1 or 0 and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as

$$\therefore X_1 = A_1 B_1 + \overline{A_1} \overline{B_1}$$

$$\text{and } X_0 = A_0 B_0 + \overline{A_0} \overline{B_0}$$

- * The equality of two numbers A and B is displayed in a combinational circuit by an output binary variable that we designate by the symbol $(A=B)$

- * The binary variable $(A=B)$ is equal to 1 if the input numbers A and B are equal and is equal to 0 otherwise.

- * For equality exists, all X_i variables must be equal to 1 a condition that dictates and AND operation of all variables.

$$(A=B = X_1 X_0)$$

- * The binary variable $(A=B)$ is equal to 1 only if all pairs of digits of the two numbers are equal.

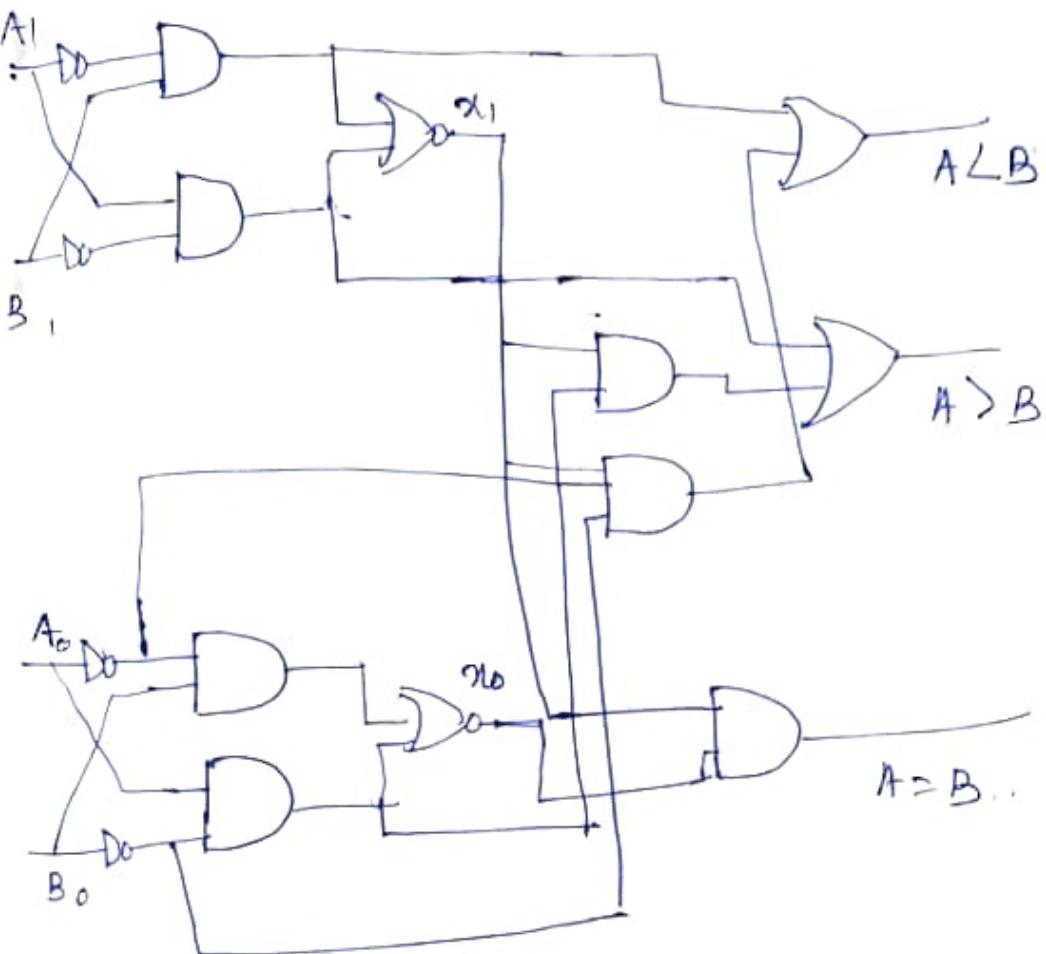
- * To determine whether A is greater or less than B , we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal, we compare the next lower significant pair of digits. If the corresponding digit of A is 1 and that of B is 0 we

We conclude that $A > B$. If the corresponding digit of A is 0 and that of B is 1, we have $A < B$. The sequential comparison can be expressed logically by the two Boolean functions,

$$(A > B) = A_1 B_1 + A_1 X_1 A_0 B_1'$$

$$(A < B) = A_1' B_1 + X_1 A_0' B_0'$$

$A_1 \ A_0$	$B_1 \ B_0$	$A > B$	$A < B$	$A = B$
0 0	0 0	0	0	1
0 0	0 1	0	1	0
0 0	1 0	0	1	0
0 0	1 1	0	1	0
0 1	0 0	1	0	0
0 1	0 1	0	0	1
0 1	1 0	0	1	0
0 1	1 1	0	1	0
1 0	0 0	1	0	0
1 0	0 1	1	0	0
1 0	1 0	0	0	0
1 0	1 1	0	1	0
1 1	0 0	1	0	0
1 1	0 1	1	0	0
1 1	1 0	1	0	0
1 1	1 1	0	0	1



Logic Diagram of 2 Bit-Magnitude Comparators.

Decoder

Decoder is a multi-input, multi-output combinational logic circuit which decodes n inputs into 2^n possible output.



Where E

Enable

$E=0$ Decoder is disabled
 $E=1$ Decoder is enabled

2 to 4 bit Decoder Design

	E	A	B		2^3	2^2	2^1	2^0
					y_3	y_2	y_1	y_0
0	0	0	0		0	0	0	0
1	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

If input is 2, ($n = 2$)

$$\text{output } 2^n = 2^2 = 4$$

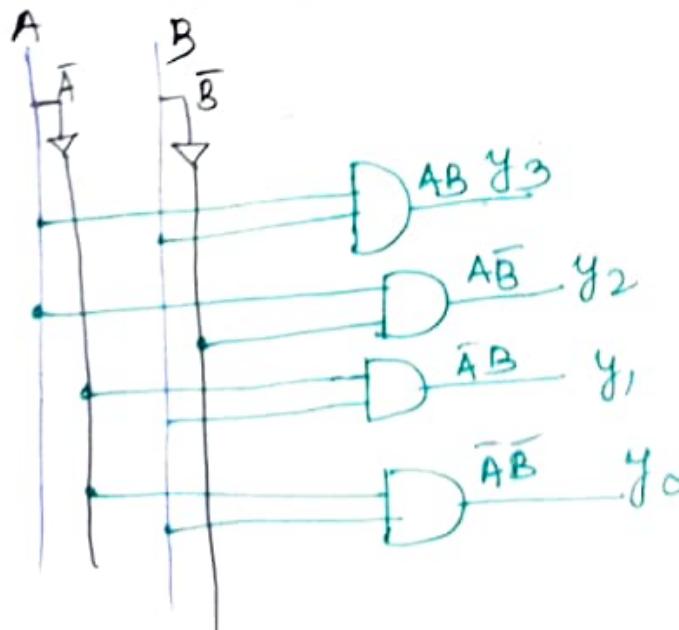
From the truth table

$$y_0 = \bar{A} \bar{B}$$

$$y_1 = \bar{A} B$$

$$y_2 = A \bar{B}$$

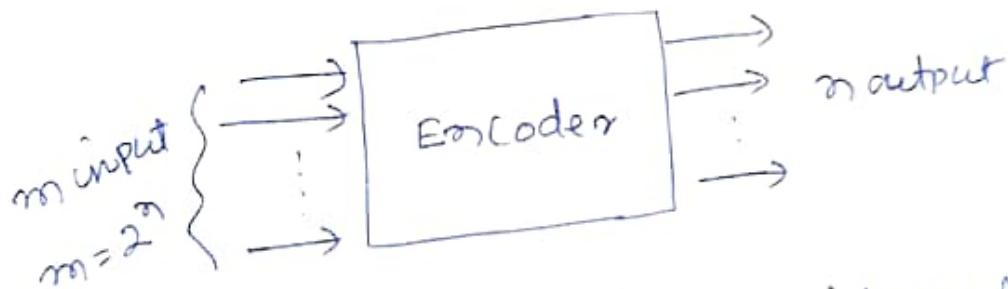
$$y_3 = A B$$



Encoder

Encoder is a logic circuit which performs the opposite action of the decoder. An encoder has a number of input lines only one of which is activated at a time. At the output it displays a value corresponding to activated output.

An encoder has 2^n input lines and n output lines.

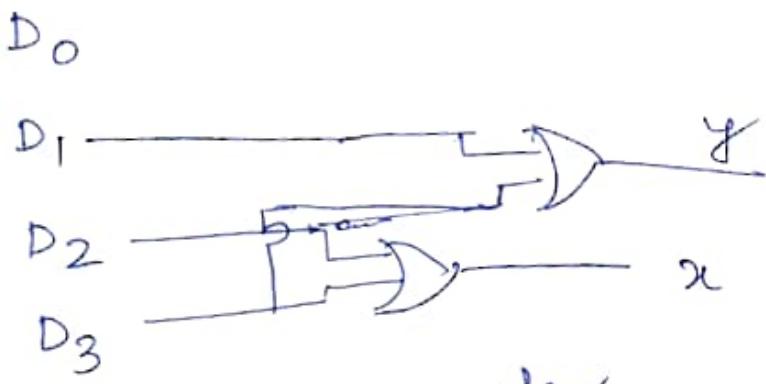


Encoder converts binary information in the form of 2^n i/p lines into n o/p lines which represent m bit code for the i/p.

D_3	D_2	D_1	D_0	X	Y
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$X = D_2 + D_3$$

$$Y = D_1 + \underline{D_2}$$



8 to 3 line encoder

Encoder has eight inputs (one for each of the octal digits) and three output that generates the corresponding binary number

Inputs								Outputs		
D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	X	Y	Z
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

only one input has a value 1 at any given time.

output Z is equal to one (1) when the input octal digit is 1, 3, 5, 7

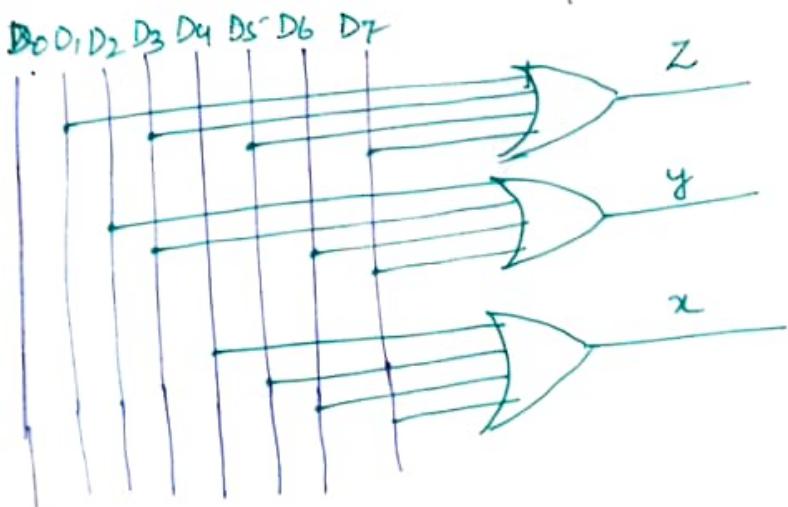
output y is 1 when the input octal digit is 2, 3, 6, 7

output x is 1 when 4, 5, 6 or 7

$$Z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$



Priority encoder:

A Priority encoder is a encoder circuit that includes the Priority function.

The operation of the Priority encoder is such that if two or more inputs are equal to 1 at the same time the input having the highest Priority will take precedence.

Inputs	Outputs
D ₀ D ₁ D ₂ D ₃	X Y V
0 0 0 0	X X 0
1 0 0 0	0 0 1
X 1 0 0	0 1 1
X X 1 0	1 0 1
X X X 1	1 1 1

In addition to the two outputs X and Y, the circuit has a third output designated by V this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.

X - do not care

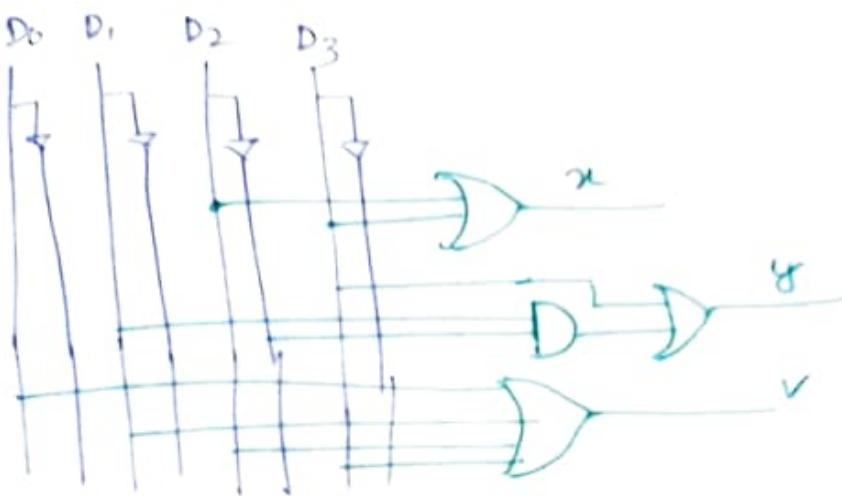
If higher the subscript number, the higher the priority of the input.

Input D₃ has the highest priority \Rightarrow regardless of the value of the other inputs when this input is 1 the output may be 1 (binary 3)

$$X = D_2 + D_3$$

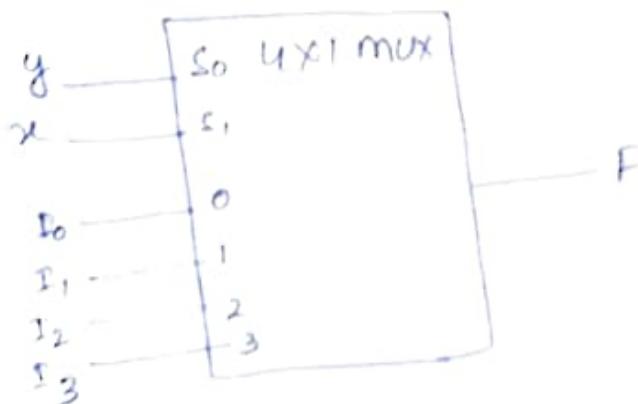
$$Y = D_3 + D_2 \bar{D}_1$$

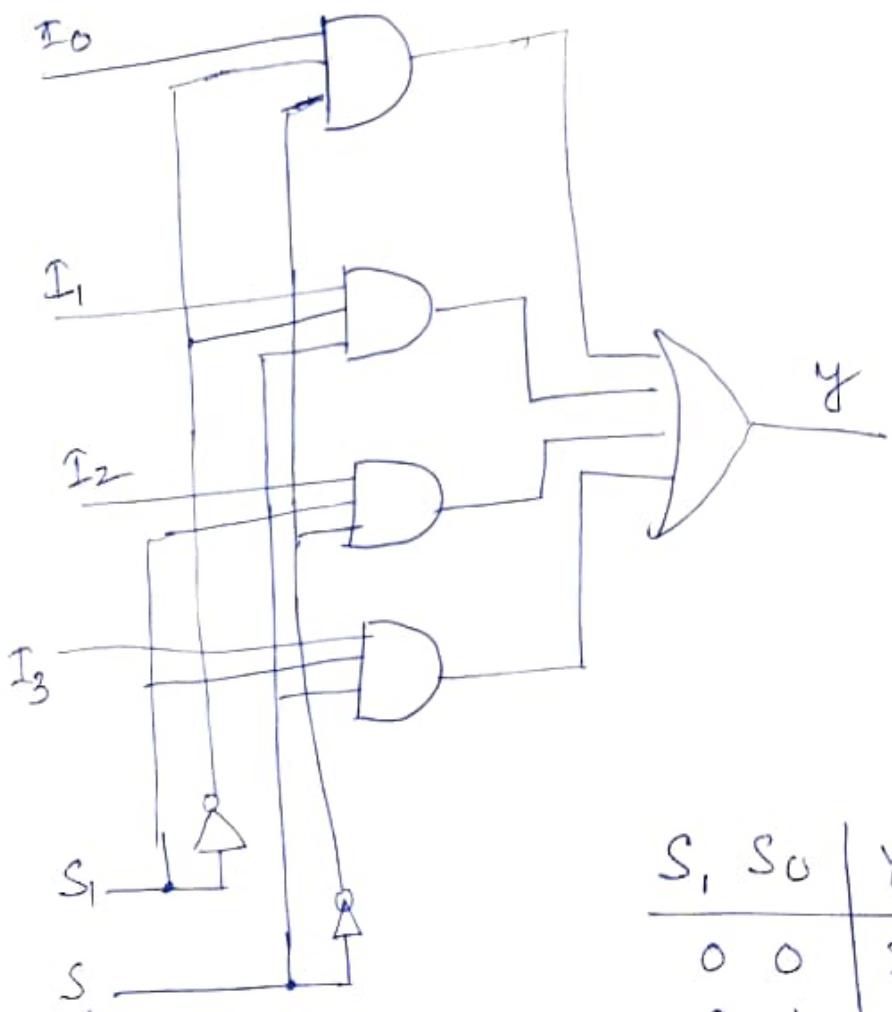
$$V = D_0 + D_1 + D_2 + D_3$$



Multiplexers !

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The particular selection of a particular input lines is controlled by a set of selection lines.
- Normally there are 2^n input lines and n selection lines whose bit combination determine which input is selected.





S_1, S_0	y
0 0	I_0
0 1	I_1
1 0	I_2
1 1	I_3

Each of the four inputs I_0 through I_3 is applied to one input of an AND gate.

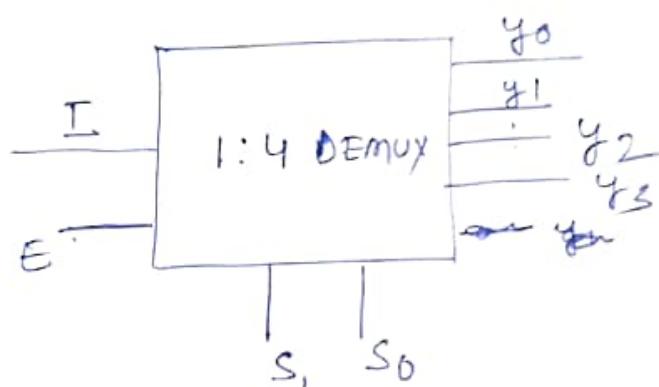
Selection line S_1 and S_0 are deodes to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the one line output.

A multiplexer is also called a data selector, since it selects one of many input and steers the binary information to the output lines.

Demultiplexor

The demultiplexer takes one single input data line and then switches it to any one of number of individual output line one at a time.

The function of the demultiplexer is to switch one common data input line to any one of the 4 input output data line ~~one at a time~~
 y_0 to y_4



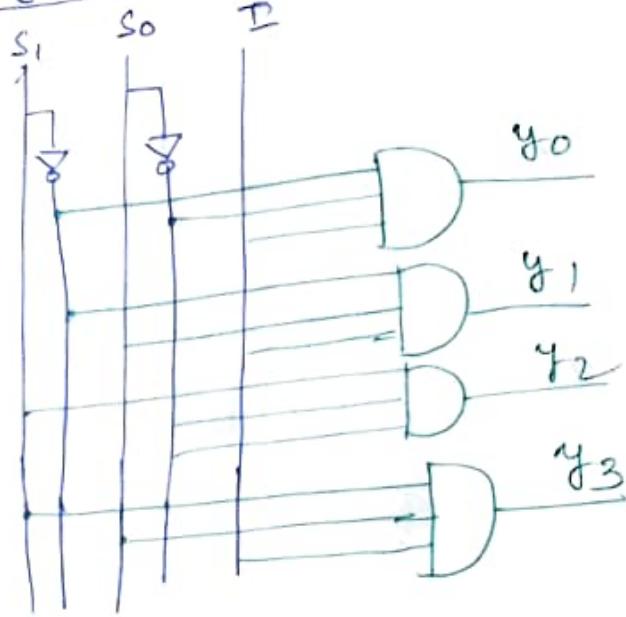
Truth table

E_i	$S_1 - S_0$	y_0	y_1	y_2	y_3
0	0 0	0	0	0	0
1	0 0	1	0	0	0
	0 1	0	1	0	0
	1 0	0	0	1	0
	1 1	0	0	0	1

$$y_0 = \bar{S}_1 \bar{S}_0 I, \quad y_1 = \bar{S}_1 S_0 I \quad y_2 = S_1 \bar{S}_0 I$$

$$y_3 = S_1 S_0 I$$

Logic gate



$$2^m = n$$

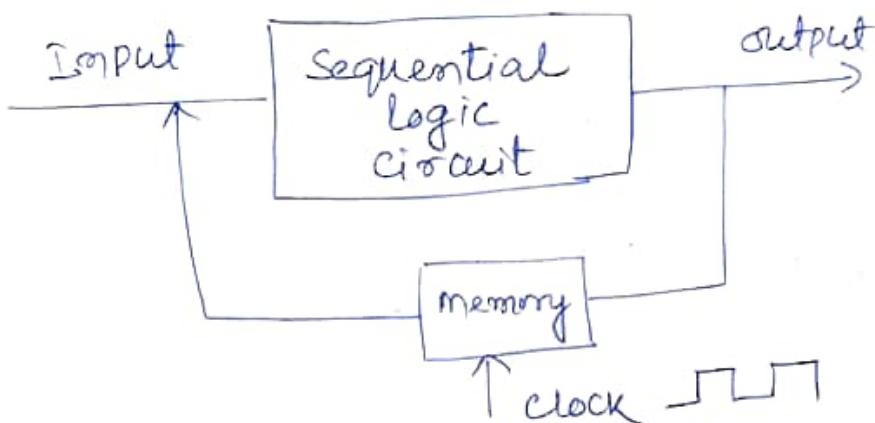
m = no. of selector

n = no. of output

Niranjjan Bhowmik
Sri Leet PKAIEE, B.A.E.

SEQUENTIAL LOGIC CIRCUIT

It is a circuit whose output depends upon the present input, previous output and the sequence in which the inputs are applied.



In combinational circuit output depends upon the present input at any instant of time and do not use memory. So previous input does not have any effect on the circuit. But sequential circuit has memory and depends upon ~~not~~ present input and previous output.

Sequential circuits are slower than combinational circuits and these sequential circuit are harder to design.

The data stored by the memory element at any given instant of time is called the present state of sequential circuit.

Types!

- 1) Synchronous SLC
- 2) Asynchronous SLC

The sequential logic circuit (SLC) that are controlled by clock are called synchronous SLC and those which are not controlled by a clock are asynchronous SLC.

A recurring pulse is called a clock! - A recurring pulse is called a clock.

Flip-Flop and Latch!

- A flip-flop and latch is a circuit that has two stable states and can be used to store information.
- A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- Latch is a non-clocked flip-flop and it is the building block for the flip-flop.
- A ~~storage~~ storage element in digital circuit can maintain a binary state indefinitely until directed by an input signal to switch state.
- Storage element that operate with level are called latches and those operate with clock transition are called as flip-flop.
- A flip-flop is called so because the output either flips or bops & meaning to switch back and forth.

* A flip-flop is also called bi-stable multivibrator as it has two stable states. The input signals which command command the flip-flop to change state are called excitations.

- * clock-signals may be positive-edge triggered or negative-edge triggered.
- * Positive-edge triggered flip-flops are those in which state transitions take place only at positive-going edge of the clock pulse.

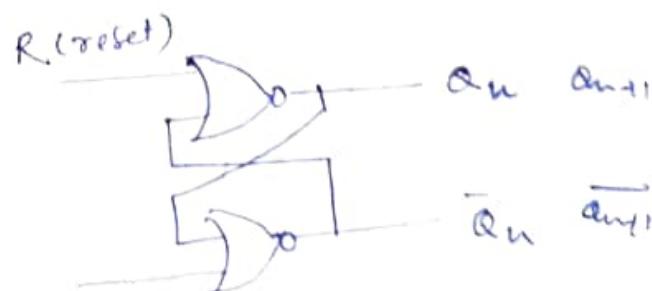


* Negative-edge triggered flip-flops are those in which state transition take place only at negative-going edge of the clock pulse.

SR Latch

The SR latch is a circuit with two cross coupled NOR gates or two cross coupled NAND gates.

It has two outputs labeled Q and \bar{Q} . Two inputs are labeled S for set and R for Reset.



The latch has two useful states. When $Q=0$ and $\bar{Q}=1$ the condition is called reset state and when $Q=1$ and $\bar{Q}=0$ the condition is called set state.

Truth table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
<hr/>			
0	1	0	0
0	1	1	0
<hr/>			
1	0	0	1
1	0	1	1
<hr/>			
1	1	0	X
1	1	1	X
<hr/>			

1st case $S=0, R=0$

$$Q_{n+1} \text{ (Next stage)} = \overline{0 + \bar{Q}_n} = (\bar{Q}_n) = Q_n$$

No change

2nd case $S=0, R=1$

$$Q_{n+1} \text{ (Next stage)} = \overline{(1 + \bar{Q}_n)} = (\bar{1}) = 0$$

$$\overline{Q_{n+1}} \text{ (")} = \overline{(0 + \bar{Q}_n)} = \bar{Q}_n$$

Reset

3rd case

$$S=1 \quad R=0$$

$$Q_{n+1} = \overline{(0 + \bar{Q}_n)} = (\bar{Q}_n) = Q_n$$

$$\overline{Q_{n+1}} = \overline{1 + \bar{Q}_n} = (\bar{1}) = 0$$

Set

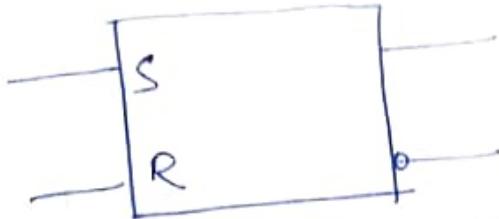
4th case

$$S=1 \quad R=1$$

$$Q_{n+1} = \overline{(1 + \bar{Q}_n)} = (\bar{1}) = 0$$

$$\overline{Q_{n+1}} = \overline{1 + \bar{Q}_n} = (\bar{1}) = 0$$

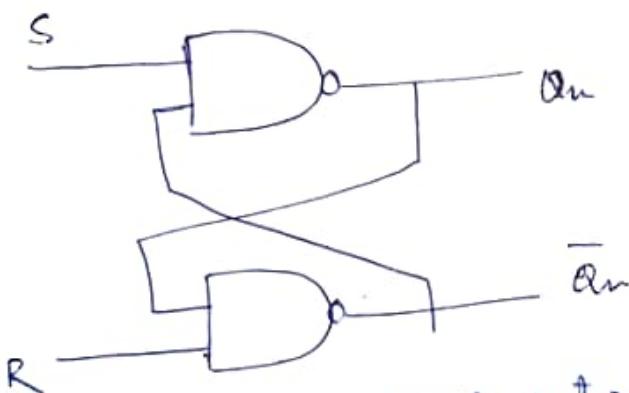
Indeterminate state



Symbol of SR NOR Latch

Racing Condition : In case of a SR latch when $S=R=1$ input is given both the output will try to be come 0. This is called racing condition.

S-R Latch using NAND gate



The circuit has NAND gates and we know if any one of the input for a NAND gate is Low then the output will be High and if both the inputs are HIGH then only the output will be Low.

It operates with both inputs normally at 1, unless the state of the latch has to be changed. The application of 0 to the S input causes output Q to go to 1, putting the latch in the set state. When S input goes back to 1, the circuit remains in the set state. After both inputs go back to 1, we are allowed to change the state of the latch by placing 0 in the input R.

This action causes the circuit to go to reset state and stay there even after both inputs return to 1.

The condition that is forbidden for the NAND latch is both inputs being equal to 0 at the same time an input combination that should be avoided.

Truth table

S	R	Q_n	Q_{n+1}
0	0	x	Raling (ID)
0	1	x	0
1	0	x	1
1	1	x	No change

$$\textcircled{1} \quad S=0 \quad R=0$$

$$Q_{n+1} = (\overline{0} \cdot \overline{Q_n}) = \overline{0} = 1$$

$$\overline{Q_{n+1}} = (\overline{0} \cdot \overline{Q_n}) = \overline{0} = 1$$

Raling condition

$$\textcircled{2} \quad S=0 \quad R=1$$

$$Q_{n+1} = (\overline{0} \cdot \overline{Q_n}) = \overline{0} = 1$$

Reset condition

$$\textcircled{3} \quad S=1 \quad R=0$$

$$Q_{n+1} = (\overline{1} \cdot \overline{Q_n}) = (\overline{Q_n}) = Q_n$$

$$\overline{Q_{n+1}} = (\overline{0} \cdot \overline{Q_n}) = \overline{Q}(\overline{0}) = 1$$

Set condition

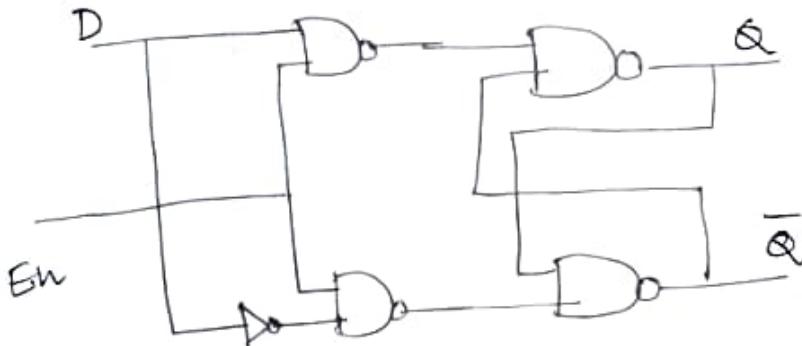
$$\textcircled{1} \quad S=1 \quad R=1$$

$$Q_{n+1} = (\overline{1 \cdot \bar{Q}_n}) = \bar{\bar{Q}_n} = Q_n$$

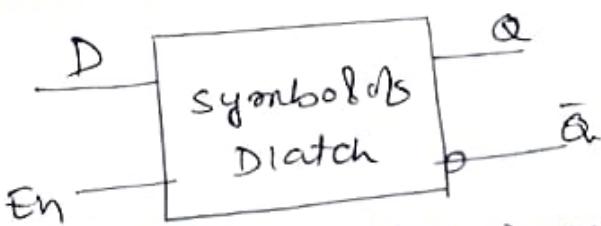
$$(\overline{Q_{n+1}}) = \overline{(\overline{1 \cdot Q_n})} = \overline{\bar{\bar{Q}_n}} = \bar{Q}_n$$

No change

D-Latch



- * One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs S and R never equal to 1 at the same time.
- * This is done in the D latch. This latch has only two inputs D (data) and En (Enable).
- * The D input goes directly to the S input, and its complement is applied to the R input.



As long as the enable input is at 0, the cross-coupled SR latch has both inputs at the 1 level and the circuit can't change state regardless of the value D.

Truth table of D-latch

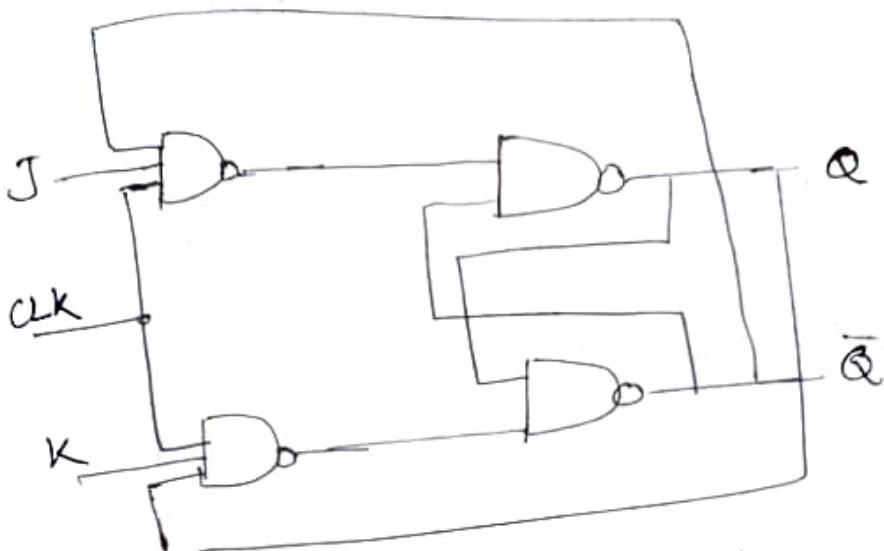
En	D	Next state of Q
0	X	No change
1	0	$Q = 0$. Reset state
1	1	$Q = 1$ Set state

The D input is sampled when En=1. If D=1 the Q output goes to 1, placing the circuit in the set state. If D=0 output Q goes to 0, placing the circuit in the reset state.

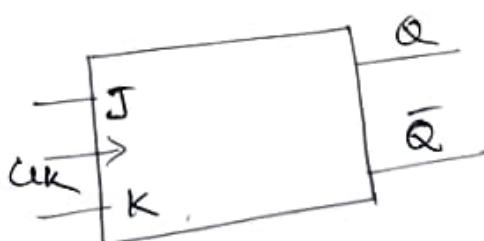
J-K Flip-Flop

- The JK flip-flop can be constructed by using basic SR latch and a clock. In this case the output Q and \bar{Q} are referred back and connected to inputs of NAND gates.
- The simple JK flip-flop is most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit.
- The sequential operation of the flip-flop is exactly the same as for previous SR flip-flop with the same set and R-set inputs.

The difference this time is that the JK flip-flop has no invalid or forbidden input state of the SR Latch even when S and R are both at logic 1



The JK flip-flop is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level 1.



Both the S and R inputs of the SR bi-stable have been replaced by two inputs called J and K inputs respectively after its invertor; Jack and Kilby. Then this equates to $J = S$ and $K = R$.

The two 2 input NAND gates of the gated SR bi-stable have now been replaced by 3 input NAND gates with the third input

at each gate connected to the outputs, Q and \bar{Q} .

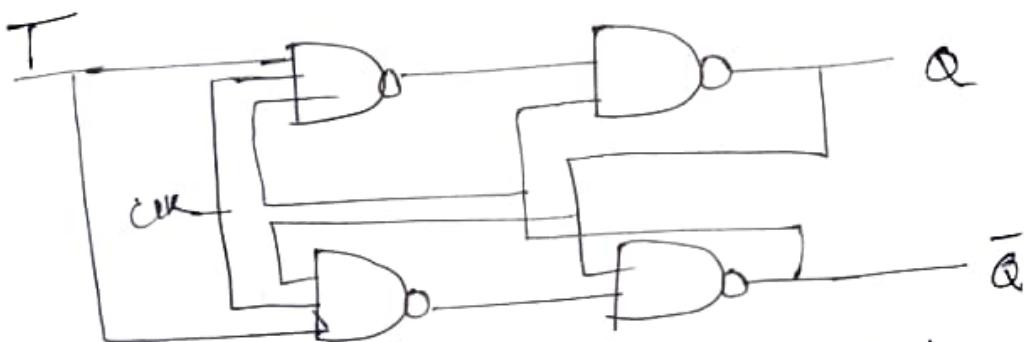
- This cross coupling at the SR flip-flop allows the previously invalid condition of $S = '1'$ and $R = '1'$ state to be used to produce a toggle action as the two inputs are now interlocked.
- If the circuit is now set the J input is inhibited by the 0 status of Q through the lower NAND gate. If the circuit is RESET the K input is inhibited by the 0 status of \bar{Q} through the upper NAND gate. As Q and \bar{Q} are always different we can use them to control the inputs.

(Truth table for JK flip-flop)

Input		Output		Comment
J	K	Q	Q _{next}	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	RESET
0	1	1	0	
1	0	0	1	SET
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

T - Flip-Flop

- Toggle flip-flop or commonly known as T flip-flop.
- This flip-flop has the similar operation as that of JK flip-flop with the both the inputs J and K are shorted i.e. both are given the common input.



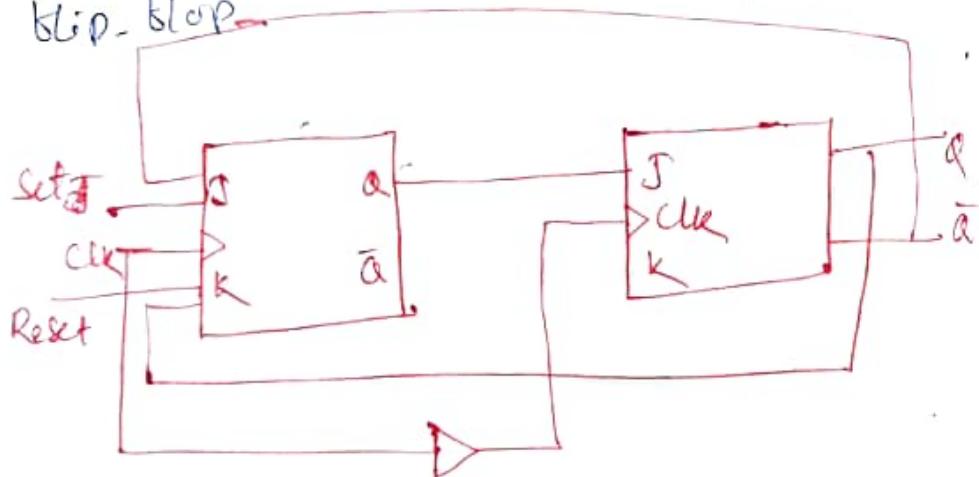
The truth table is same as that of JK flip-flop when $J=K=0$ and $J=K=1$.

Truth table of ~~JK~~ T flip-flop

T	Q	Q _{next}	Comment
0	0	0	No change
-	1	1	
-	0	1	Toggles
1	1	0	

MASTER-SLAVE JK FLIP-FLOP

- * The master-slave flip-flop is basically two gated SR flip-flops connected together in series configuration with the slave an inverted clock pulse.
- * The outputs from Q and \bar{Q} from the "slave" flip-flop are fed back to the input of the master with the outputs of the master flip flop being connected to the two inputs of the slave flip-flop.
- * This feed back configuration between the slave output to the master's input gives the characteristic toggle of the JK flip-flop.



- * The input signals J and K are connected to the gated master SR flip-flop which locks the input condition while the clock (Clk) input is HIGH at logic level 1.
- * As clock input of the slave flip-flop is the inverse (complement) of the master clock input the slave SR flip-flop does not toggle.

- * The outputs from the master flip-flop are only seen by the gated slave flip-flop when the ~~clock~~ clock input goes Low to logic level '0'.
 - * When the clock is Low the outputs from the master flip-flop are latched and any additional changes to its inputs are ignored.
 - * The gated slave flip-flop now responds to the state of its input passed over by the master section.
 - * Then on the low-to-high transition of the clock pulse the inputs of the master flip-flop are fed through to the gated inputs of the slave flip-flop and on the High-to-low transition the same inputs are reflected on the output of the slave making this type of flip-flop edge or pulse triggered.
 - * The circuit accepts input data when the clock signal is 'HIGH' and passes the data to the ~~input~~ output on the falling-edge of the clock signal.
 - * Master-slave JK flip-flop is a synchronous device as it only passes data with the timing of the clock signal.

FLIP-FLOP conversion! -

Steps for the conversion:

- * Consider truth table for destination flip-flop.
- * Extend it as Excitation table of source flip-flop.
- * Draw K-map for source flip-flop input variables it will provide expression for conversion.
- * Draw its circuit according to K-map expression.

$$SR \rightarrow D$$

$$JK \rightarrow T$$

$$T \rightarrow D$$

$$SR \rightarrow JK$$

$$JK \rightarrow D$$

$$D \rightarrow T$$

$$SR \rightarrow T$$

SR FLIP-FLOP to JK FLIP-FLOP.

Truth table of JK

J	K	Qn	Qn+1
0	0	X	Qn
0	1	X	0
1	0	X	1
1	1	X	\bar{Q}_n

Excitation table of SR

Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Truth table

J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

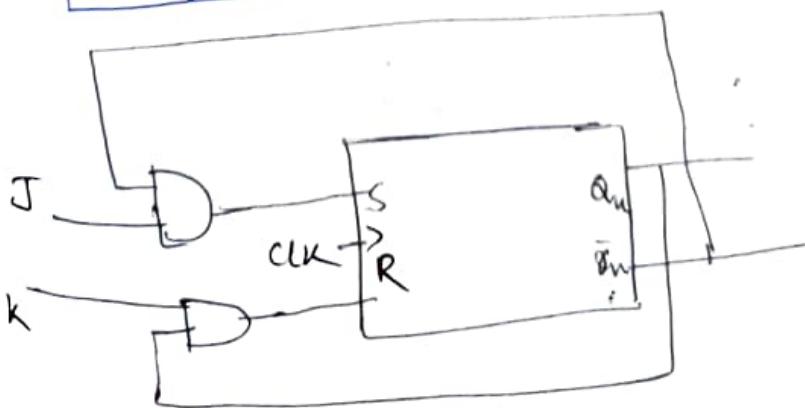
K-map for S

K-map for S		00	01	11	10
J	K	00	01	11	10
0	0	0	0	(1)	(1)
1	X	0	0	0	X

$$S = J \bar{Q}_n$$

K-map for R		00	01	11	10
J	K	00	01	11	10
0	X			(1)	X
1			(1)	(1)	

$$R = K \bar{Q}_n$$

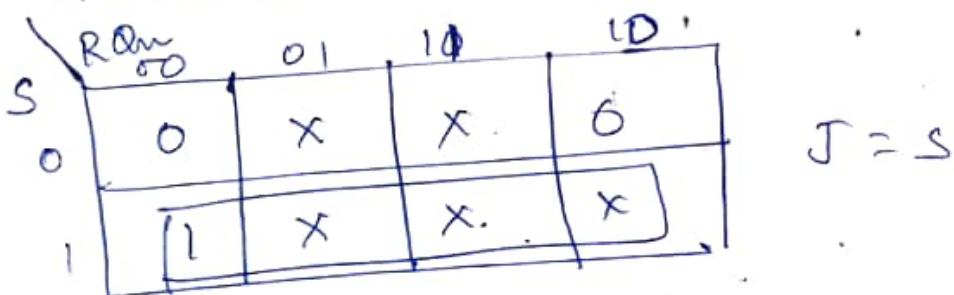


JK flip-flop to SR flip-flop

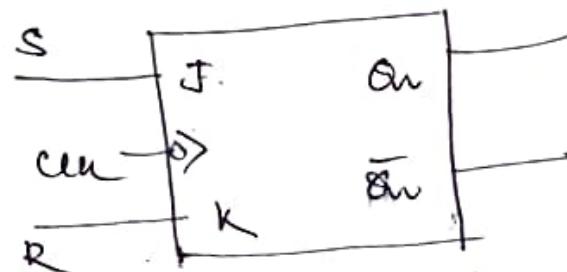
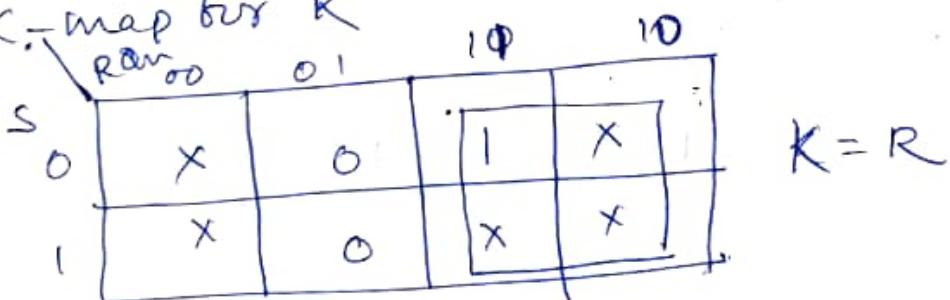
conversion table

S	R	Q _n	Q _{n+1}	J	K
0	0	0	0	0	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	0	x	1
1	0	0	1	1	x
1	0	1	1	x	0
1	1	0	Invalid		x
1	1	1	Invalid		x

K-map for J



K-map for K

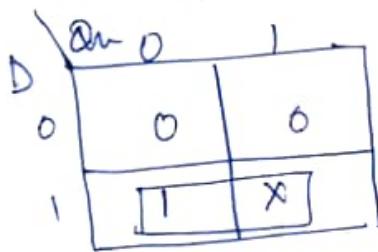


SR Flip-Flop to D Flip-Flop

conversion table

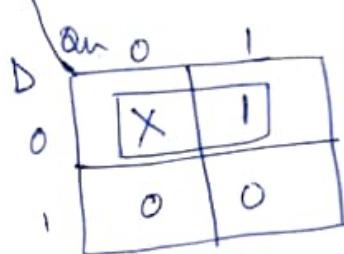
D	Q_{in}	Q_{out+1}	SR
0	0	0	0 X
0	1	0	0 1
1	0	1	1 0
1	1	1	X 0

K-map for S



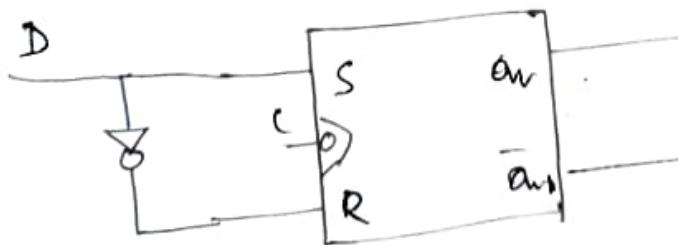
$$S = \bar{D}$$

K-map for R



$$R = \bar{D}$$

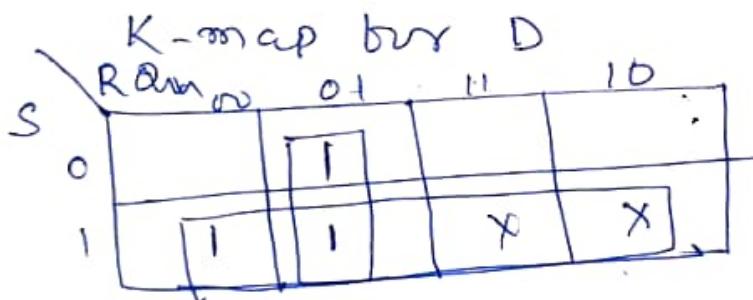
Logic Diagram



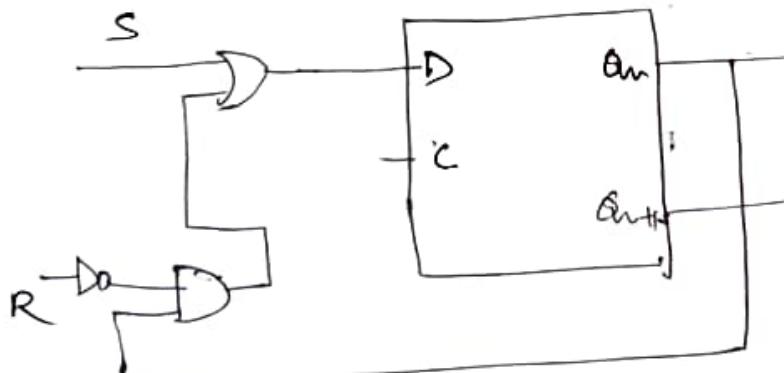
D Flip-Flop to SR Flip-Flop

Conversion table

S	R	Q_m	Q_{m+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	0	1	Invalid State	X X
1	1	1	1	X X



$$D = S + \bar{R}Q_m$$

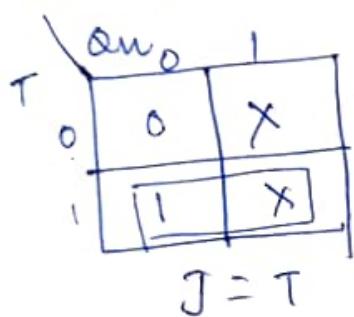


JK Flip-flop to T Flip-Flop

conversion table

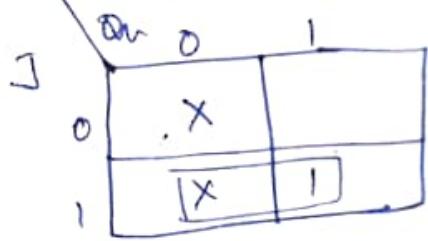
T	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-map for J

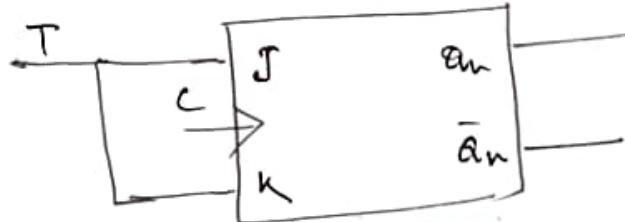


$$J = T$$

K-map for T



$$K = T$$

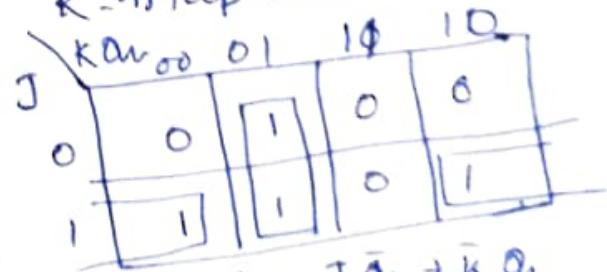


D-Flip-Flop to JK Flip-Flop

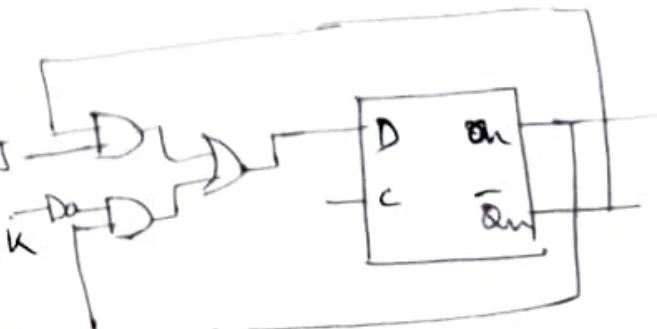
conversion table

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

K-map for D



$$D = \bar{J}\bar{Q}_n + \bar{K}Q_n$$

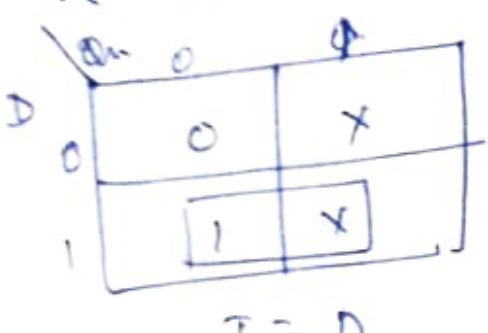


J K FLIP-FLOP to D FLIP-Flop

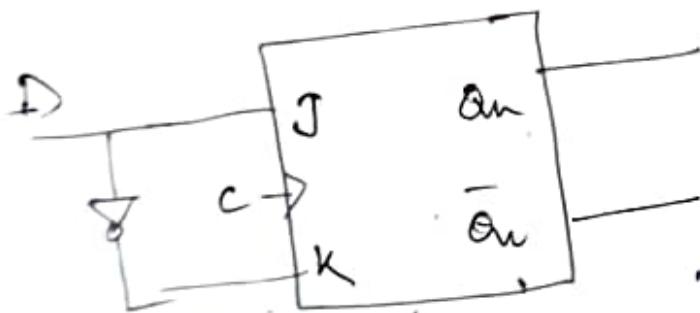
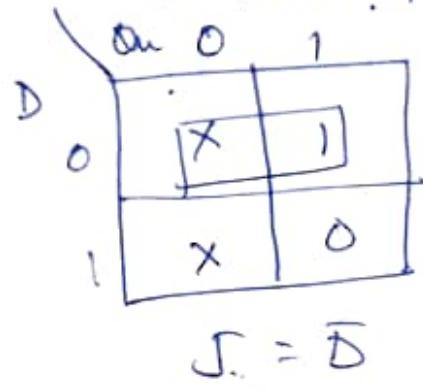
conversion table

D	Qn	Qn+1	J	K
0	0	0	0	x
0	1	0	x	1
1	0	1	1	x
1	1	1	x	0

K-map for J



K-map for K



Excitation table for SR F/F

Truth table of SR F/F

S	R	PS Q_n	NS Q_{n+1}
0	0	X \leq^0	Q_n^0
0	1	X \leq^0	0
1	0	X \leq^0	1
1	1	X \leq^0	1

Excitation table of RF/F

PS Q_n	NS Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Niranjana Behere
Sr. Lect P.K.A.I.E.T.
BARGARH

Counter

Nirajgian Behan
Sr. Lect PHA2E

Counter is an instrument used for time and frequency. A counter is simply a device that counts. They will count up or down by one's, two's or ~~three's~~ three's.

Counters are a series of flip-flops wired together to perform the type of counting desired.

The total number of counts or states a counter can indicate is called Modulus.

E.g. - Modulus of four bit counter is 16
(it is capable of counting 0000₂ to 1111₂)

Binary counters can be classified as.

① Asynchronous counter or Ripple counter.

② Synchronous counter.

A Synchronous counter / Ripple counter

is constructed by use of clocked

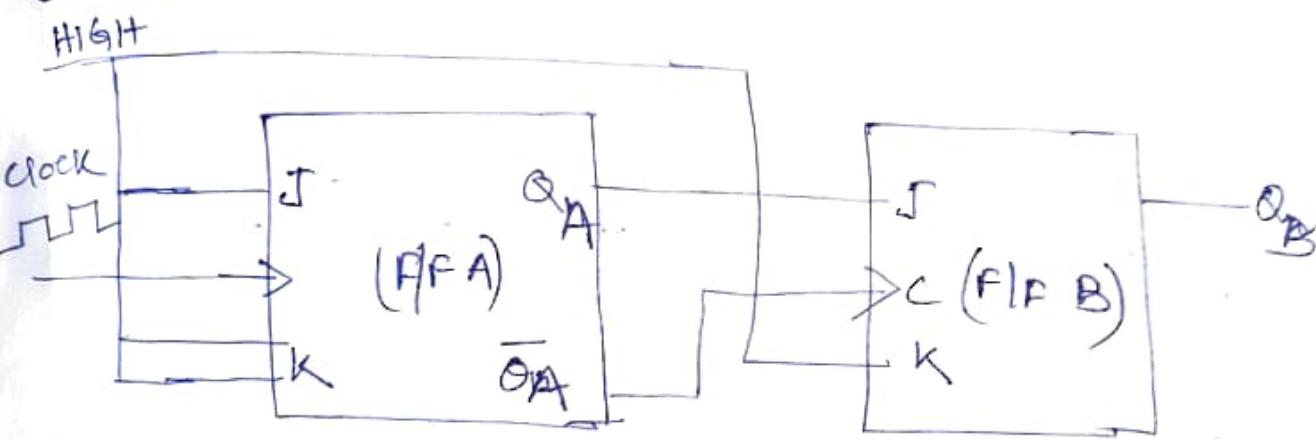
T- flip-flop.

In this, the output of a flip-flop is used as clock inputs of next flip-flop.

The term Asynchronous refers to events that do not occurs at same time.

The clock is connected to flip-flop A. The flip-flop 'B' is triggered by Q_A (output of flip-flop A)

The clock to all flip-flop in this counter is not applied at the same time instead of this, output of one flip-flop is clock pulse applied to another flip-flop.



Two bit Asynchronous Binary Counter

Initially $Q_A = 0$, $Q_B = 0$, when the first clock Pulse is applied, the output of FF A (Q_A) goes to high and $\bar{Q}_A = 0$.

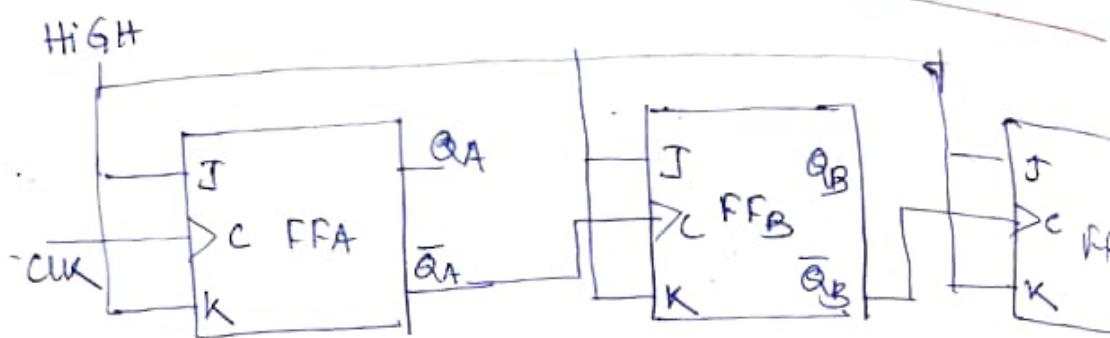
After first clock Pulse $Q_A = 1$, $Q_B = 0$

After second clock pulse $Q_A = 0$ &
therefore $\bar{Q}_A = 1$

This $\bar{Q}_A = 1$ acts as clock Pulse for FF B and the output of FF B $Q_B = 1$

clock pulse	Q_B	Q_A	Count
0	0	0	0
1	0	1	1
2	1	0	2
3	1	1	3

Three Bit Asynchronous Binary Counter



clock	Q_C	Q_B	Q_A	Counter
0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7

Modulus Counter OR Divide By N counter

A four bit i.e four flip-flop counter is referred as mod 16 or modulus 16 counter since it has 16 states i.e 2^4 . Similarly a 3 bit flip-flop counter is called modulus 8 or mod 8 counter since it has 8 states only (2^3)

Modulus or mod of counter is defined as number of states through which the counter progresses.

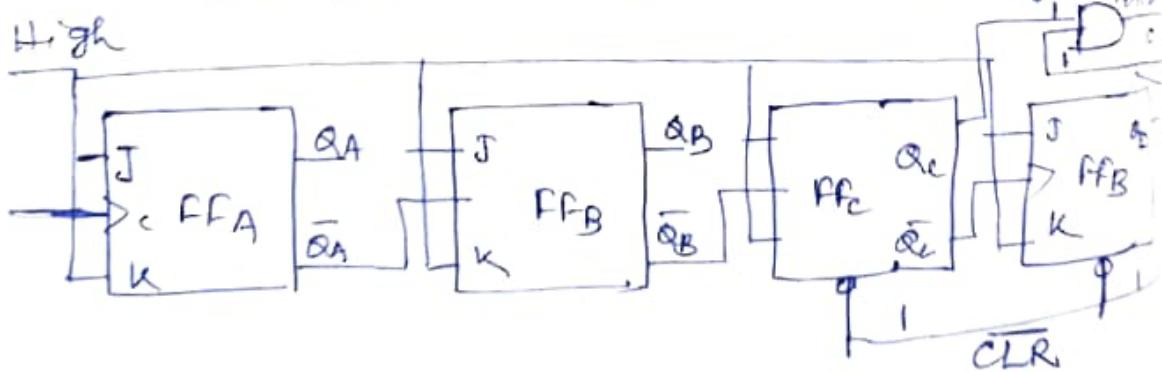
But sometimes for a particular digital system we may require to construct a counter having 3, 5, 7 or 9 i.e not equal to normal binary number.

A Synchronous mod 12 counter \rightarrow

For constructing a mod-12 counter we require four flip-flops (2^4) mod-12 Counter will count 12 states from 0000 to 1100

Q_D	Q_C	Q_B	Q_A	counter
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
1	0	1	1	11
1	0	0	0	12
0	0	0	0	Reset / Recycle

High



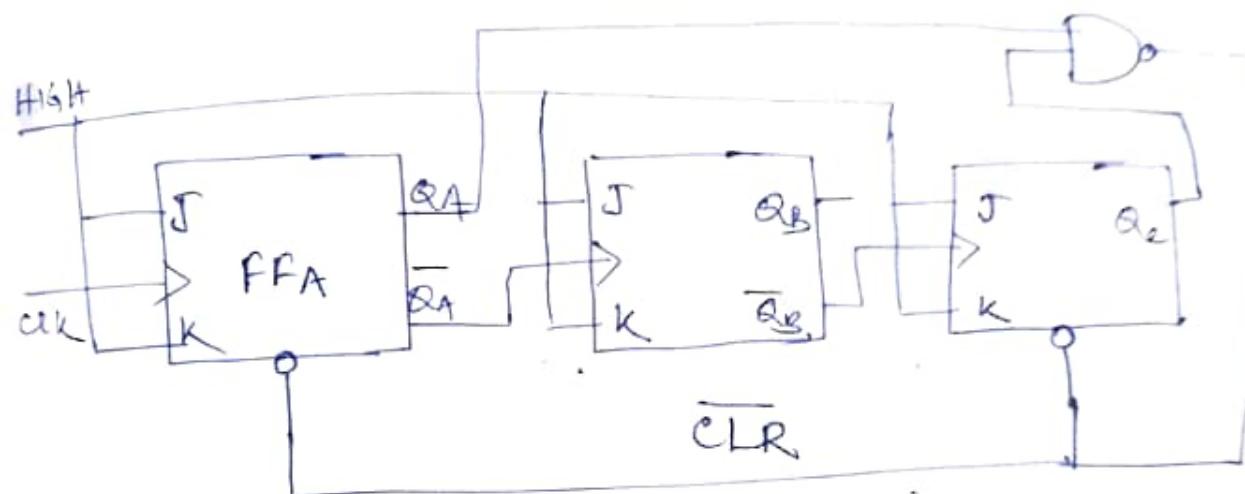
A Synchronous mod-5 Counter :-

3 F/Fs $\Rightarrow 2^3$ counter = 8 counts

2 F/Fs $\Rightarrow 2^2$ counters = 4 counts

$$2^2 < 5 < 2^3 \text{ max}.$$

For constructing mod-5 counter, we require 3 F/Fs.



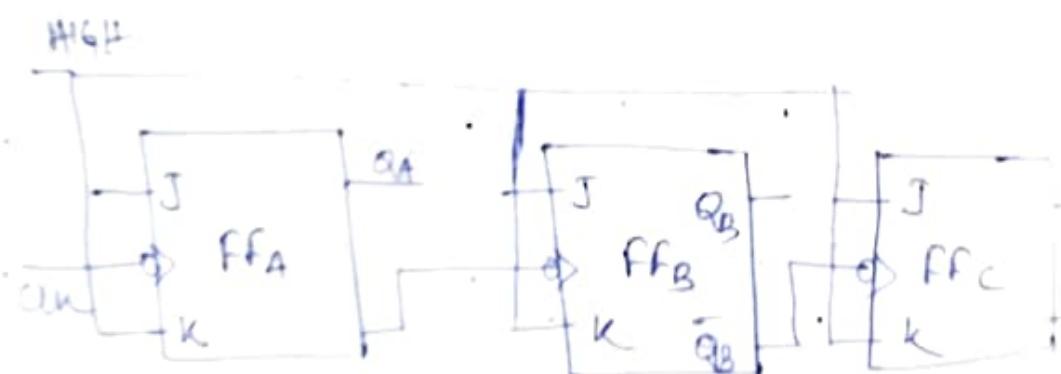
Q_C	Q_B	Q_A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
\rightarrow_0		Reset	

Q Draw Asynchronous Decade Counter
mod-10 counter

Asynchronous Down Counter

A down counter is a counter that will count downward from maximum to zero.

C	B	A	Count
1	1	1	7
1	1	0	6
1	0	1	5
1	0	0	4
0	1	1	3
0	1	0	2
0	0	1	1
0	0	0	0

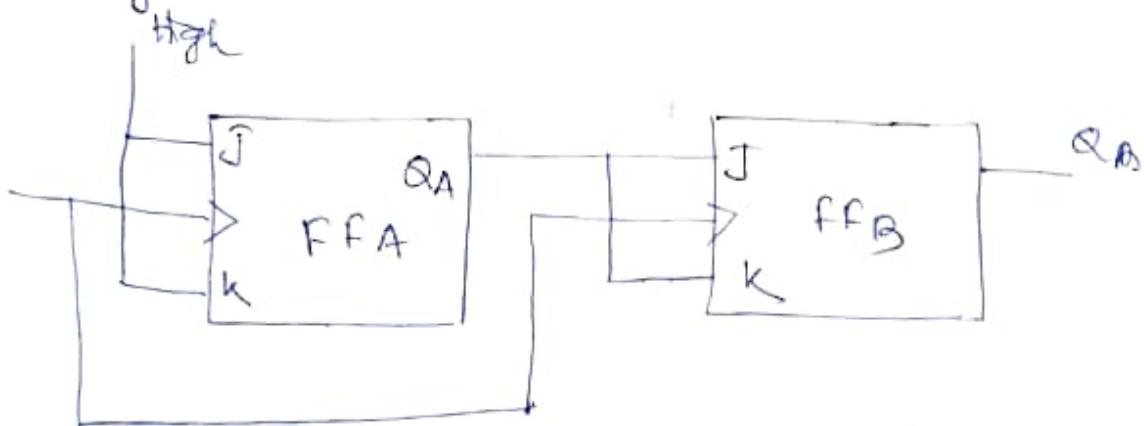


Mod-8 Counter / Down Counter
Negative edge trigger clock pulse.

Synchronous counter:-

Synchronous means at the same time. In synchronous counter, clock pulse is applied to all the flip-flops at the same time.

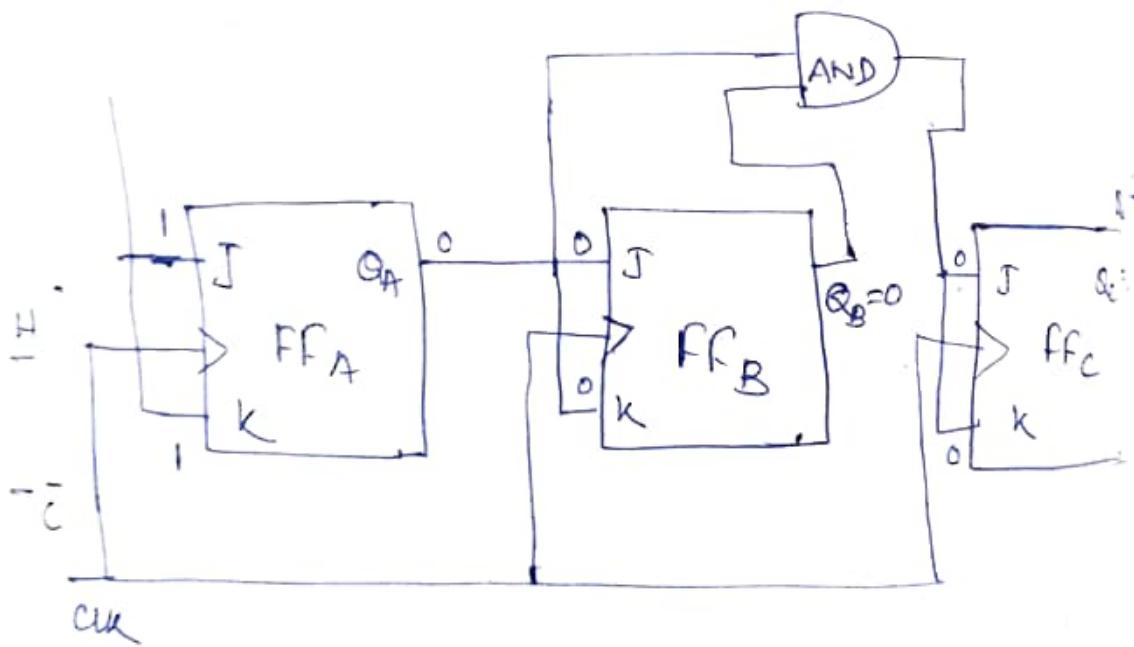
- * - A synchronous counters are simple but speed is low, since output of one flip-flop drives the other flip-flop. Each flip-flop has a delay time (i.e. time required for getting the output from the instant the input is applied) & when we use a number of flip-flops, the total delay or cumulative delay is the sum of delay time of each flip-flop used. This results in speed limitation for asynchronous counter.
- * The speed of operating improves significantly if all the flip-flops are clocked simultaneously as in case of synchronous counters.



Two bit synchronous Binary Counter

Three Bits Synchronous Binary Counter

CLK	Q_C	Q_B	Q_A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

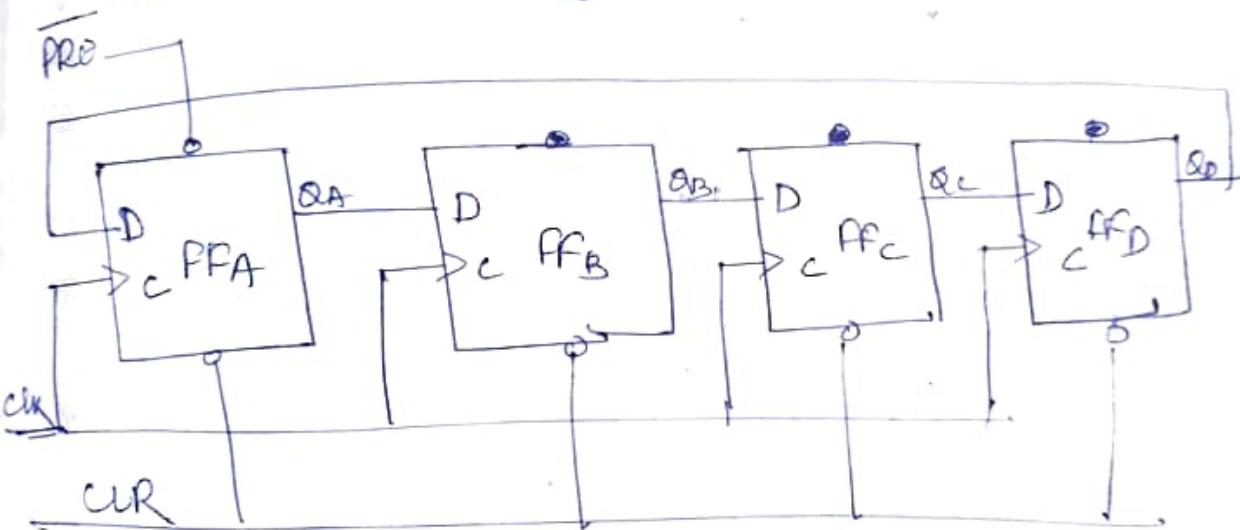


Ring counters! →

It works on the principle of feedback.

Four bit Ring counter:

In this, 1 circulates around the registers as long as clock pulse keep arriving. For this reason these counters are called circulating register as ring ~~counters~~ counters.



It is constructed using D-F/F

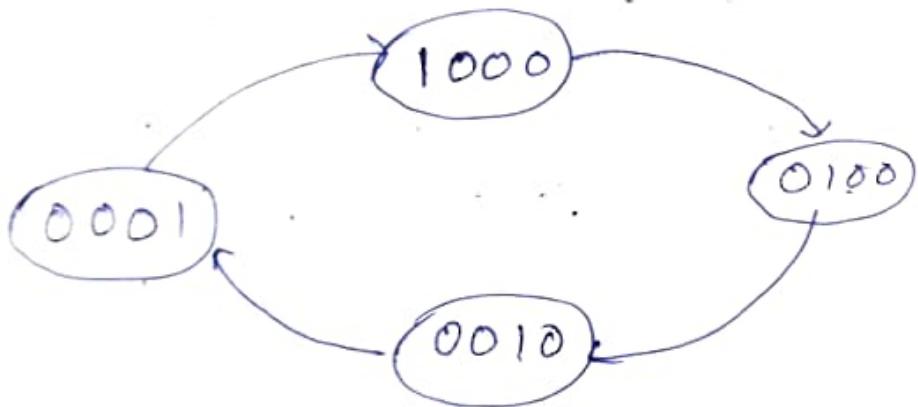
In D-F/F

$$D = 0 \quad Q_{IP} = 0 \quad (\text{reset})$$

$$D = 1 \quad Q_{IP} = 1 \quad (\text{set})$$

Initially $Q_A = 1$ and other F/F ~~Q~~
 $Q_B = Q_C = Q_D = 0$

After clk pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1

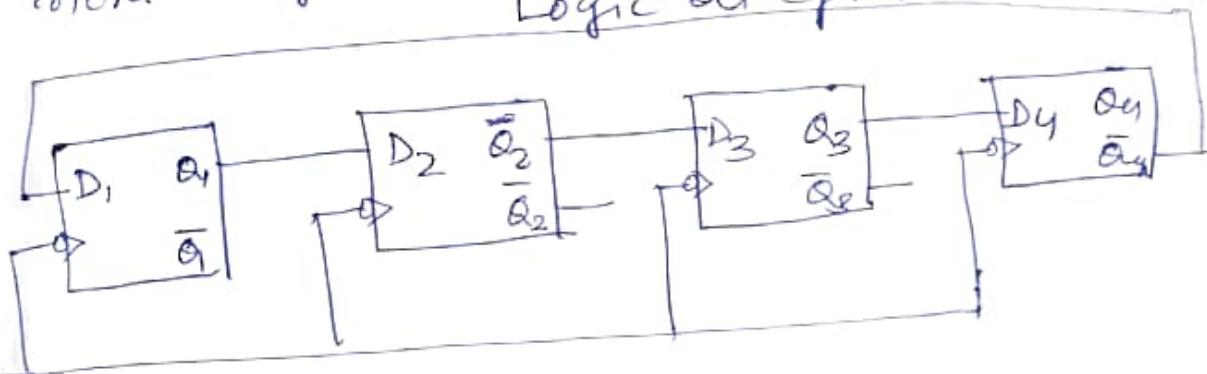


State diagram of 4 bit ring counter.

Twisted Ring counter / Johnson counter

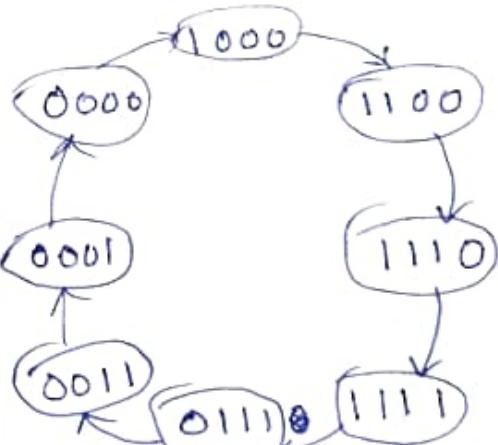
This counter is obtained by providing a feedback from the inverted output of last feedback flip-flop to the D input of the first flip-flop. The Q output of each stage is connected to the D input of next stage.

Logic diagram



clk

After clock pulse	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0



state diagram