LECTURE NOTES

DIGITAL ELECTRONICS & MICROPROCESSOR

PREPARED

BY

MR . SATYABRAT PRADHAN

LECTURER IN ELECTRICAL ENGINEERING

SYNERGY POLYTECHNIC

Unit-1: Basics of Digital Electronics

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form. Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

Number System-Binary, Octal, Decimal, Hexadecimal - Conversion from one system to another number system.

Number :

The way of quantifying anything , represented through various combination of symbols is called number.

Digit :

The various symbols representing a single number in any number system is called digit.
E.g. Decimal number system (Arabic numerals): Digits: 0,1,2,3,4,5,6,7,8,9.

Radix / Base (r) :

The maximum number of different digits of any number system. E.g Decimal NS, r =10

Number system:

The properly structured number formation is called Number system. In number system there are different symbols and each symbol has an absolute value and also has place value.

In general a number in a system having base or radix ' r ' can be written as

Number various combination digits according to position

Nr = [Integer part . Fractional part]

↑ Radix point

$= d_n\, d_{n-1} \ldots d_1 d_0 \,.\, d_{-1} d_{-2} \ldots d_{-m}$

The value,

$$N_{10} = \mathbf{d_n x\, r^n} + \mathbf{d_{n-1} x\, r^{n-1}} + \ldots + \mathbf{d_1 x\, r^1} + \mathbf{d_0 x\, r^0} + \mathbf{d_{-1} x\, r^{-1}} + \mathbf{d_{-2} x\, r^{-2}} + \ldots + \mathbf{d_{-m} x\, r^{-m}}$$

* The right most digit of any number is called Least Significant Digit

* The lef most digit of any number is called Most Significant Digit

TYPES OF NUMBER SYSTEM:-

There are four types of number systems. They are

1. Decimal number system
2. Binary number system
3. Octal number system
4. Hexadecimal number system

DECIMAL NUMBER SYSTEM:- •

The decimal number system contain ten unique symbols 0,1,2,3,4,5,6,7,8 and 9.

- In decimal system 10 symbols are involved, so the base or radix is 10.
- It is a positional weighted system.

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + (d_{n-2} \times 10^{n-2}) + \ldots + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \ldots + (d_{-m} \times 10^{-m})$$

For example:-

$$9256.26 = 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100)$$

$$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$$

BINARY NUMBER SYSTEM:-

- The binary number system is a positional weighted system.
- The base or radix of this number system is 2.
- It has two independent symbols, The symbols used are 0 and 1.
- A binary digit is called a bit

$$(d_n \times 2^n) + (d_{n-1} \times 2^{n-1}) + (d_{n-2} \times 2^{n-2}) + \ldots + (d_0 \times 2^0) + (d_{-1} \times 2^{-1}) + (d_{-2} \times 2^{-2}) + \ldots + (d_{-k} \times 2^{-k})$$

OCTAL NUMBER SYSTEM:-

- It is also a positional weighted system.
- Its base or radix is 8.
- It has 8 independent symbols 0,1,2,3,4,5,6 and 7.
- Its base 8 = 2^3 , every 3- bit group of binary can be represented by an octal digit.

HEXADECIMAL NUMBER SYSTEM:-

- The hexadecimal number system is a positional weighted system.
- The base or radix of this number system is 16.
- The symbols used are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F
- The base 16 = 24 , every 4 – bit group of binary can be represented by an hexadecimal digit.

CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER :-

1. BINARY NUMBER SYSTEM:-

(a) Binary to decimal conversion:- In this method, each binary digit of the number is multiplied by its positional weight and the product terms are added to obtain decimal number.

$(111.101)_2$ $= (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$
$= 4 + 2 + 1 + 0.5 + 0 + 0.125$
$= (7.625)_{10}$

(b) Binary to Octal conversion:- For conversion binary to octal the binary numbers are divided into groups of 3 bits each, starting at the binary point and proceeding towards left and right.

(i) Convert $(101111010110.110110011)_2$ into octal.

Solution :

Group of 3 bits are	101	111	010	110	.	110	110	011
Convert each group into octal =	5	7	2	6	.	6	6	3

The result is **$(5726.663)_8$**

(c) Binary to Hexadecimal conversion:- For conversion binary to hexadecimal number the binary numbers starting from the binary point, groups are made of 4 bits each, on either side of the binary point.

(ii) Convert $(01011111011.011111)_2$ into hexadecimal.

Solution:

Given Binary number		010	1111	1011	.	0111	11
Group of 3 bits are	=	0010	1111	1011	.	0111	1100
Convert each group into octal =		2	F	B	.	7	C

The result is **$(2FB.7C)_{16}$**

2. DECIMAL NUMBER SYSTEM:-

(a) Decimal to binary conversion:- In the conversion the integer number are converted to the desired base using successive division by the base or radix.

For example: (i) Convert $(52)_{10}$ into binary.

(ii) Convert $(105.15)_{10}$ into binary.

Solution:

Integer part		Fraction part
2 | 105		0.15 x 2 = 0.30
2 | 52	— 1	0.30 x 2 = 0.60
2 | 26	— 0	0.60 x 2 = 1.20
2 | 13	— 0	0.20 x 2 = 0.40
2 | 6	— 1	0.40 x 2 = 0.80
2 | 3	— 0	0.80 x 2 = 1.60
2 | 1	— 1	
0	— 1	

Result of $(105.15)_{10}$ is **$(1101001.001001)_2$**

(b) Decimal to octal conversion:- To convert the given decimal integer number to octal, successively divide the given number by 8 till the quotient is 0.

(i) Convert $(378.93)_{10}$ into octal.

Solution:

8 | 378		0.93 x 8 = 7.44
8 | 47	— 2	0.44 x 8 = 3.52
8 | 5	— 7	0.52 x 8 = 4.16
0	— 5	0.16 x 8 = 1.28

Result of $(378.93)_{10}$ is **$(572.7341)_8$**

(c) Decimal to hexadecimal conversion:-

(i) Convert $(2598.675)_{10}$ into hexadecimal.

Solution:

	Remainder Decimal	Hex		Hex
16 | 2598			0.675 x 16 = 10.8	A
16 | 162	— 6	6	0.800 x 16 = 12.8	C
16 | 10	— 2	2	0.800 x 16 = 12.8	C
0	— 10	A	0.800 x 16 = 12.8	C

Result of $(2598.675)_{10}$ is **$(A26.ACCC)_{16}$**

3. OCTAL NUMBER SYSTEM:-

(a) Octal to binary conversion:- To convert a given a octal number to binary, replace each octal digit by its 3- bit binary equivalent.

For example:

Convert $(367.52)_8$ into binary.

Solution:

Given Octal number is	3	6	7	.	5	2	
Convert each group octal to binary	= 011	110	111	.	101	010	

Result of $(367.52)_8$ is **$(011110111.101010)_2$**

(b) Octal to decimal conversion:- For conversion octal to decimal number, multiply each digit in the octal number by the weight of its position and add all the product terms

For example: -

Convert $(4057.06)_8$ to decimal

Solution:

$$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$$
$$= 2048 + 0 + 40 + 7 + 0 + 0.0937$$
$$= (2095.0937)_{10}$$

Result is **$(2095.0937)_{10}$**

(c) Octal to hexadecimal conversion:- For conversion of octal to Hexadecimal, first convert the given octal number to binary and then binary number to hexadecimal

(4) HEXADECIMAL NUMBER SYSTEM :- (a)Hexadecimal to binary conversion:- For conversion of hexadecimal to binary, replace hexadecimal digit by its 4 bit binary group

Convert $(3A9E.B0D)_{16}$ into binary.

Solution:

Given Hexadecimal number is		3	A	9	E	.	B	0	D
Convert each hexadecimal digit to 4 bit binary	=	0011	1010	1001	1110	.	1011	0000	1101

Result of $(3A9E.B0D)_8$ is **$(0011101010011110.101100001101)_2$**

(b)Hexadecimal to decimal conversion:- For conversion of hexadecimal to decimal, multiply each digit in the hexadecimal number by its position weight and add all those product terms.

Convert $(A0F9.0EB)_{16}$ to decimal

Solution:

$$(A0F9.0EB)_{16} = (10 \times 16^3)+(0 \times 16^2)+(15 \times 16^1)+(9 \times 16^0)+(0 \times 16^{-1})+(14 \times 16^{-2})+(11 \times 16^{-3})$$
$$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$$
$$= (41209.0572)_{10}$$

Result is **$(41209.0572)_{10}$**

((c) Hexadecimal to Octal conversion:- For conversion of hexadecimal to octal, first convert the given hexadecimal number to binary and then binary number to octal

Arithmetic Operation-Addition, Subtraction, Multiplication, Division, 1's & 2's complement of Binary numbers& Subtraction using complements method

1. BINARY ADDITION:-

The binary addition rules are as follows

carry

0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 10 , SUM,

carry

1+1+1 = 11 SUM

Add $(100101)_2$ and $(1101111)_2$.
Solution :-

$$\begin{array}{rr} & 100101 \\ + & 1101111 \\ \hline & 10010100 \end{array}$$

Result is **$(10010100)_2$**

2. BINARY SUBTRACTION:-

The binary subtraction rules are as follows

0 - 0 = 0 ; 1 - 1 = 0 ; 1 - 0 = 1 ; 0 - 1 = 1 1 , with a borrow of 1

Substract $(111.111)_2$ from $(1010.01)_2$.
Solution :-

```
  1010.010
- 111.111
  0010.011
```

Result is **$(0010.011)_2$**

3. BINARY MULTIPLICATION:-

The binary multiplication rules are as follows

0 x 0 = 0 ; 1 x 1 = 1 ; 1 x 0 = 0 ; 0 x 1 = 0

Multiply $(1101)_2$ by $(110)_2$.
Solution :-

```
   1101
x   110
   0000
  1101
+ 1101
1001110
```

Result is **$(1001110)_2$**

4. BINARY DIVISION:-

The binary division is very simple and similar to decimal number system.

So we have only 2 rules 0 ÷ 1 = 0 1 ÷ 1 = 1

```
110 ) 101101 ( 111.1
  - 110
    1010
    110
    1001
    110
     110
     110
     000
```

Result is **$(111.1)_2$**

1's COMPLEMENT REPRESENTATION :-

The 1's complement of a binary number is obtained by changing each 0 to 1 and each 1 to 0.

Find $(1100)_2$ 1's complement.

Solution :-

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

Result is $(0011)_2$

2's COMPLEMENT REPRESENTATION :-

The 2's complement of a binary number is a binary number which is obtained by adding 1 to the 1's complement of a number.

2's complement = 1's complement + 1

Find $(1010)_2$ 2's complement.

Solution :-

Given		1	0	1	0
1's complement is		0	1	0	1
	+				1
2's complement		0	1	1	0

Result is $(0110)_2$

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.

0 1 0 1 0 0 1 = +41
↑
Sign bit

1 1 0 1 0 0 1 = -41
↑
Sign bit

SUBSTRACTION 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 the 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.

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

Solution:-

```
  1 1 0 1 0                  1 1 0 1 0                    =  26
- 1 0 0 0 0      =>      + 0 1 1 1 1  (1's complement)    = - 16
                 Carry →   1 0 1 0 0 1                     + 10
                         +         1
                           0 1 0 1 0   = +10
```

Result is **+10**

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.

Subtract $(1010100)_2$ from $(1010100)_2$ using 2's complement.

Solution:-

```
  1 0 1 0 1 0 0                1 0 1 0 1 0 0                =   84
- 1 0 1 0 1 0 0      =>     + 0 1 0 1 1 0 0  (2's complement) = - 84
                             1 0 0 0 0 0 0 0 ( Ignore the carry)   0
                        =          0 (result = 0)
```

Hence MSB is 0. The answer is positive. So it is +0000000 = **0**

Digital Code & its application & distinguish between weighted & non-weight Code, Binary codes, excess-3 and Gray codes.

DIGITAL CODES:-

In practice the digital electronics 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. There is various possible ways of doing this and this process is called 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** : In weighted codes, for each position (or bit) ,there is specific weight attached. For example, in binary number, each bit is assigned particular weight 2^n where 'n' is the bit number for n = 0,1,2,3,4 the weights are 1,2,4,8,16 respectively. Example :- BCD

2) Non- weighted binary codes:

Non-weighted codes are codes which are not assigned with any weight to each digit position, i.e., each digit position within the number is not assigned fixed value. Example:- Excess – 3 (XS -3) code and Gray codes

BINARY CODED DECIMAL (BCD):- BCD is a weighted code. In weighted codes, each successive digit from right to left represents weights equal to some specified value and to get the equivalent decimal number 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.

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 add the resulting carry to the next most significant.

Add 679.6 from 536.8 using BCD addition.

Solution:-

```
  6 7 9.6        0110 0111 1001 . 0110   ( 679.6 in BCD)
+ 5 3 6.8   =>+ 0101 0011 0110 . 1000   (536.8 in BCD)
  1 2 1 6.4      1011 1010 1111 . 1110   ( All are illegal codes)
               + 0110 0110 0110 . 0110   ( Add 0110 to each)
               0001 0010 0001 0110 . 0100
                 1    2    1    6   .  4   ( corrected sum = 1216.4)
```

Result is **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, then 6 (0110) is subtracted from the difference term of this group.

Subtract 147.8 from 206.7 using 8421 BCD code.

Solution:-

```
  2 0 6.7        0010 0000 0110 . 0111   ( 206.7 in BCD)
- 1 4 7.8   =>- 0001 0100 0111 . 1000   (147.8 in BCD)
  5 8.9          0000 1011 1110 . 1111   ( Borrows are present)
               - 0110 0110 0110 . 0110
                 0101 1000 0101 . 1001
                   5    8    9   .  9   ( corrected difference = 58.9)
```

Result is $(58.9)_{10}$

EXCESS THREE(XS-3) CODE:-

The Excess-3 code, also called XS-3, 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.

Excess-3 code is non-weighted and self complementary code. A self complementary binary codes are always compliment themselves. The complement of a binary number can be obtained from that number by replacing 0's with 1's and 1's with 0's. The sum of binary number and its complement is always equal to decimal 9. In other words, the 1's complement of an excess-3 code is the excess-3 code for the 9's complement of the corresponding decimal number.

For example, the excess-3 code for decimal number 5 is 1000 and 1's complement of 1000 is 0111, which is excess-3 code for decimal number 4, and it is 9's complement of number 5.

ASCII CODE:-

The American Standard Code for Information Interchange (ASCII) pronounced as 'ASKEE' is widely used alphanumeric code. This is basically a 7 bit code. The number of different bit patterns that can be created with 7 bits is 27 = 128 , the ASCII can be used to encode both the uppercase and lowercase characters of the alphabet (52 symbols) and some special symbols in addition to the 10 decimal digits. It is used extensively for printers and terminals that interface with small computer systems. The table shown below shows the ASCII groups.

GRAY CODE:-

The gray code is a non-weighted code. It is not a BCD code. It is cyclic code because successive words 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.

BINARY- TO – GRAY CONVERSION:-

If an n-bit binary number is represented by $B_n\ B_{n-1}$ ----- B_1 and its gray code equivalent by $G_n\ G_{n-1}$ ----- G_1, where B_n and G_n are the MSBs , then gray code bits are obtained from the binary code as follows

$G_n = B_n$

$G_{n-1} = B_n \oplus B_{n-1}$

.

.

.

.

$G_1 = B_2 \oplus B_1$

Where the symbol $\oplus$ stands for Exclusive OR (X-OR)

GRAY- TO - BINARY CONVERSION:-

If an n-bit gray number is represented by $G_n\ G_{n-1}$ ------- G_1 and its binary equivalent by $B_n\ B_{n-1}$ ----- B_1, then binary bits are obtained from Gray bits as follows :

$B_n = G_n$

$B_{n-1} = B_n \oplus G_{n-1}$

.

.

.

.

$B_1 = B_2 \oplus G_1$

Logic gates: AND, OR, NOT, NAND, NOR, Exclusive-OR, Exclusive-NOR-- Symbol, Function, expression, truth table & timing diagram

LOGIC GATES:-

- Logic gates are the fundamental building blocks of digital systems.
- There are 3 basic types of gates 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 in 2 levels(logic 1, logic 0). These two levels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF
- *The table which lists all the possible combinations of input variables and the corresponding output of any logic circuit/device, called a* ***truth table***.

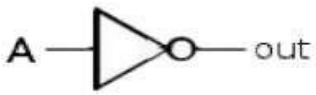
DIFFERENT TYPES OF LOGIC GATES

NOT GATE (INVERTER):-

- A NOT gate, also called and 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 inputs is in logic 1 state.

IC No. :- 7404

Logic Symbol



Truth table

INPUT A	OUTPUT $\overline{A}$
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 at logic 1 state.
- The output is logic 0 state even if one of its inputs is at logic 0 state.

IC No.:- 7408

Logic Symbol

Truth Table

		OUTPUT
A	B	Q=A . B
0	0	0
0	1	0
1	0	0
1	1	1

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 input is 1
- The output is logic 0 state, only when each one of its inputs is in logic state.

IC No.:- 7432

Logic Symbol

Truth Table

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

NAND GATE:-

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

IC No.:- 7400 two input NAND gate

Logic Symbol

Truth Table

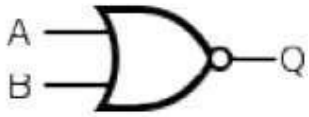
INPUT		OUTPUT
A	B	Q= $\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

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 its 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

Logic Symbol



Truth Table

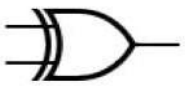
INPUT		OUTPUT
A	B	Q= $\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXCLUSIVE – OR (X-OR) GATE

- An X-OR gate is a two input, one output logic circuit.
- 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 the inputs is logic 1, the output is logic 0.

IC No.:- 7486

Logic Symbol



INPUTS are **A** and **B**

OUTPUT is **Q** = $A \oplus B$

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

Truth Table

INPUT		OUTPUT
A	B	Q = A⊕ 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 NOT gate

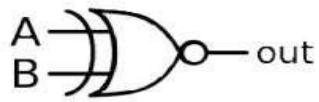
- An X-NOR gate is a two input, one output logic circuit.

The output is logic 1 only when both the inputs are logic 0 or when both the inputs is 1.

- The output is logic 0 when one of the inputs is logic 0 and other is 1

IC No.:- 74266

Logic Symbol



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

$= A\ XNOR\ B$

INPUT		OUTPUT
A	B	OUT =A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

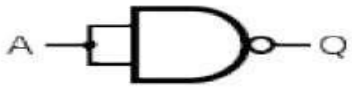
Universal Gates & its Realisation

UNIVERSAL GATES:-

There are 3 basic gates AND, OR and NOT, there are two universal gates NAND and NOR. Both NAND and NOR gates can perform all logic functions i.e. AND, OR, NOT, EXOR and EXNOR.

NAND GATE:-

a) **Inverter from NAND gate**



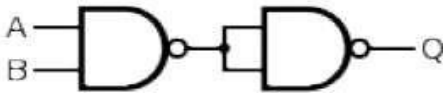
Input = A

Output $Q = \overline{A}$

b) **AND gate from NAND gate**

Input s are **A** and **B**

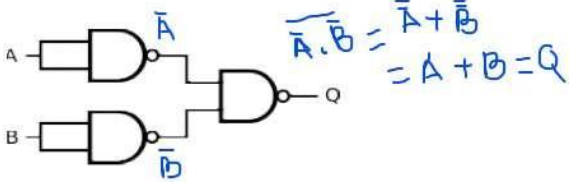
Output **Q = A.B**



c) **OR gate from NAND gate**

Inputs are **A** and **B**

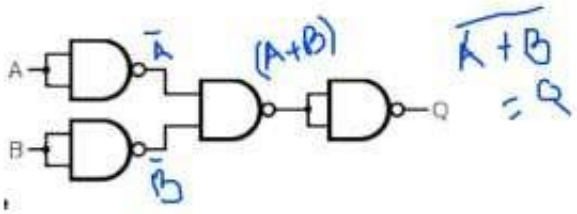
Output **Q = A+B**



d) **NOR gate from NAND gate**

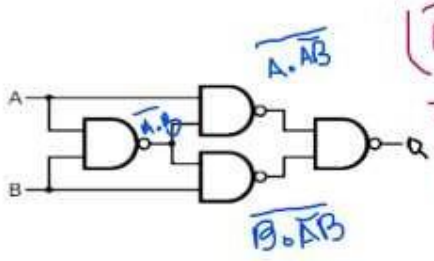
Inputs are **A** and **B**

Output $Q = \overline{A+B}$



e) **EX-OR gate from NAND gate**

Inputs are **A** and **B**
Output **Q = A $\bar{B}$ + $\bar{A}$B**



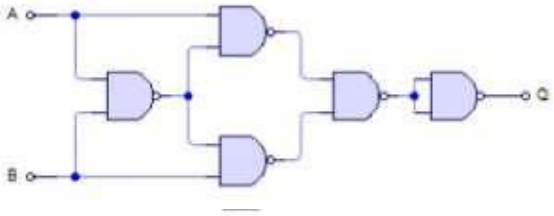
$$\overline{(A.\overline{AB})(B.\overline{AB})}$$
$$= \overline{A.\overline{AB}} + \overline{B.\overline{AB}}$$
$$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$$
$$= A\bar{B} + \bar{A}B = Q$$

e) **EX-OR g**

Inputs ar
Output Q

f) **EX-NOR gate From NAND gate**

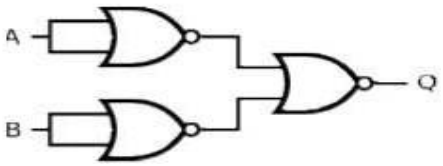
Inputs are **A** and **B**
Output **Q = A B + $\bar{A}$ $\bar{B}$**



NOR GATE:-

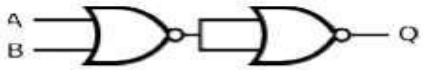
a) **Inverter from NOR gate**
Input = **A**
Output **Q = $\bar{A}$**

b) **AND gate from NOR gate**
Input s are **A** and **B**
Output **Q = A.B**



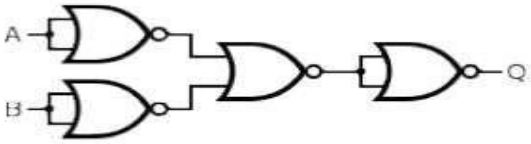
c) **OR gate from NOR gate**

Inputs are **A** and **B**
Output **Q = A+B**



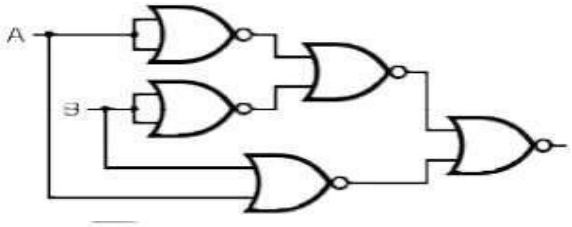
d) **NAND gate from NOR gate**

Inputs are **A** and **B**
Output **Q = $\overline{A.B}$**



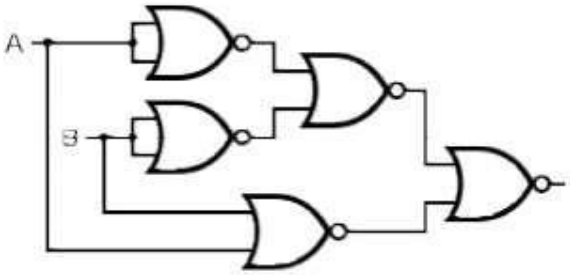
e) **EX-OR gate from NOR gate**

Inputs are **A** and **B**
Output **Q = A $\overline{B}$ + $\overline{A}$B**



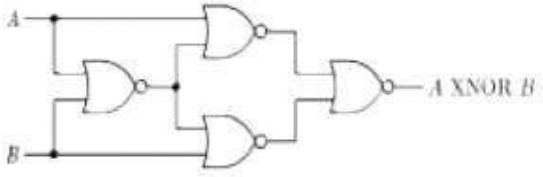
e) **EX-OR gate from NOR gate**

Inputs are **A** and **B**
Output **Q = A $\overline{B}$ + $\overline{A}$B**



f) **EX-NOR gate From NOR gate**

Inputs are **A** and **B**
Output **Q = A B + $\overline{A}$ $\overline{B}$**



Boolean algebra, Boolean expressions, Demorgan's Theorems.

BOOLEAN ALGEBRA INTRODUCTION:-

- Switching circuits are also called logic circuits, gates circuits and digital circuits.

- Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0,1), two binary operators called OR and AND and unary operator called NOT.
- It is the basic mathematical tool in the analysis and synthesis of switching circuits.
- It is a way to express logic functions algebraically.

AXIOMS AND LAWS OF BOOLEAN ALGEBRA:-

Axioms or postulates of Boolean algebra are set of logical expressions that are accepted without proof and upon which we can build a set of useful theorems.

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

1. **Complementation Laws**:-

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

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

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

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

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

   Law 5: $\bar{\bar{A}} = 0$ (double complementation law)

2. **OR Laws**:-

   The four OR laws are as follows

   Law 1: A + 0 = A (Null law)

   Law 2: A + 1 = 1(Identity law)

   Law 3: A + A = A

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

3. **AND Laws**:-

   The four AND laws are as follows

   Law 1: A . 0 = 0 (Null law)

   Law 2: A . 1 = A (Identity law)

   Law 3: A . A = A

   Law 4: A . $\bar{A} = 0$

4. **Commutative Laws:-**

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

   Law 1: A + B = B + A

   Law 2: A . B = B . A

5. **Associative Laws:-**

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

   Law 1: (A + B) + C = A + (B + C)

   Law 2: (A .B) C = A (B .C)

6. **Distributive Laws:-**

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

Law 1: A (B + C) = AB + AC

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

Proof:

RHS = (A+B) (A+C) = AA + AC + BA + BC

= A + AC + AB + BC

= A (1+ C + B) + BC

= A. 1 + BC (1 +C + B = 1 + B = 1, FROM OR Law 2)

= A + BC = LHS

7. Redundant Literal Rule (RLR):-

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

Proof

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

= 1. (A + B)

= A +B

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

Proof

$A(\bar{\square} + B) = A\bar{\square} + AB = 0 + AB = AB$

8. Idempotence Laws:-

Idempotence means same value.

Law 1: A. A = A

Law 2: A + A = A

9. Absorption Laws:-

There are two 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 + AB

= A(1 + B)

$= A \cdot 1 = A$

12. De Morgan's Theorem:-

De Morgan's theorem represents two laws in Boolean algebra.

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

Law 1: $\overline{\square + \square} = \bar{\square} \cdot \bar{\square}$

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

Law 2: $\overline{A.B} = \overline{A} + \overline{B}$

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

Proof

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

=

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

1.1 Represent Logic Expression: SOP & POS forms

SUM - OF - PRODUCTS FORM:-

- This is also called disjunctive Canonical Form (DCF) or Expanded Sum of Products Form or Canonical Sum of Products Form.
- In this form, the function is the sum of a number of products terms where each product term contains all variables of the function either in complemented or uncomplemented form.

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

- The minterm is denoted as mo, m1, m2 … . An 'n' variable function can have 2^n minterms.

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

PRODUCT- OF - SUMS FORM:-

- This form is also called as Conjunctive Canonical Form (CCF) or Expanded Product - of – Sums
- This is by considering the combinations for which f = 0
- Each term is a sum of all the variables.

The sum term which contains each of the 'n' variables in either complemented or uncomplemented form is called a **maxterm.**

- Maxterm is represented as M0, M1, M2, …….

$f(A, B, C) = \Pi M(0, 4, 6, 7)$

Karnaugh map (3 & 4 Variables) & Minimization of logical expressions, don't care conditions

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.

Mapping of SOP Expression:-

• The n variable K-map has 2^n squares. These squares are called cells.

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

Unit-2: Combinational logic circuits

COMBINATIONAL LOGIC CIRCUIT

• A combinational circuit consists of logic gates whose outputs at any time are determined from 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. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.

Half adder

The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 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 sum consists of two digits. The higher significant bit of this result is called a carry.

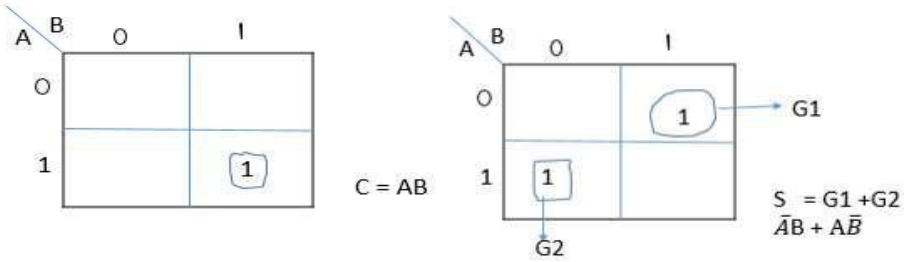
When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

A combinational circuit that performs the addition of two bits is called a half adder.

• This 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 (for sum) and C (for carry) to the outputs. The truth table for the half adder is listed in the below table.

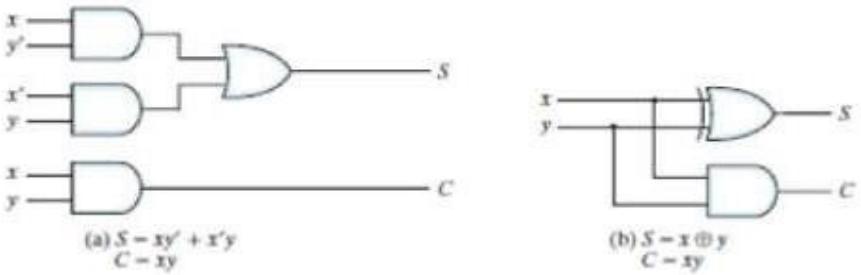
Input		Output	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



• The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum. • The simplified Boolean functions for the two outputs can be obtained directly from the truth table.

The simplified sum-of-products expressions are S = A'B + AB' C = AB

• The logic diagram of the half adder implemented in sum of products is shown in the below figure. It can be also implemented with an exclusive-OR and an AND gate.



Full 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.

• 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 , represent the two significant bits to be added. The third input, z , represents the carry from

the previous lower significant position. The two outputs are designated by the symbols S for sum and C for carry. The simplified expressions are

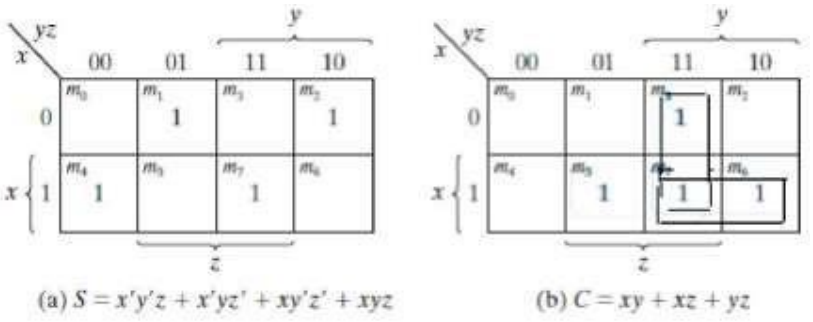
S = x'y'z + x'yz' + xy'z' + xyz

C = xy + xz + yz

- The logic diagram for the full adder implemented in sum-of-products form is shown in figure

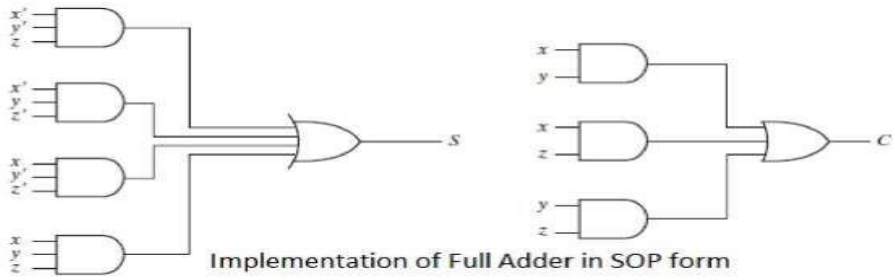
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth Table



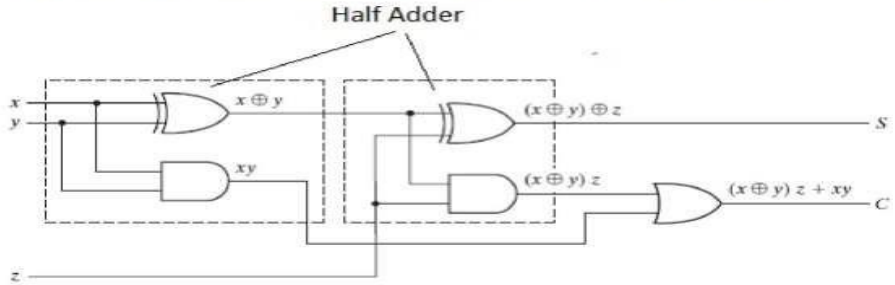
(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

K-Map for full adder



Implementation of Full Adder in SOP form

It can also be implemented with two half adders and one OR gate as shown in the figure.



Half Subtractor

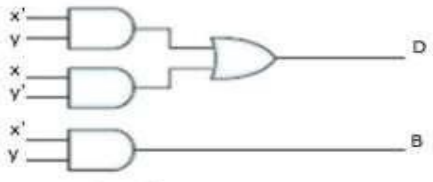
This circuit needs two binary inputs and two binary outputs. The subtraction operation is done by using the following rules as: 0 – 0 = 0; 0 – 1 = 1 with borrow 1; 1 – 0 = 1; 1 – 1 = 0

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 is listed in the below table

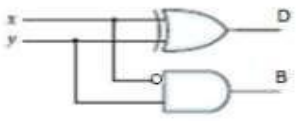
x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are D = x'y + xy' and B =x'y

The logic diagram of the half adder implemented in sum of products is shown in the figure. It can be also implemented with an exclusive-OR and an AND gate with one inverted input.



D = x'y+xy'
B = x'y



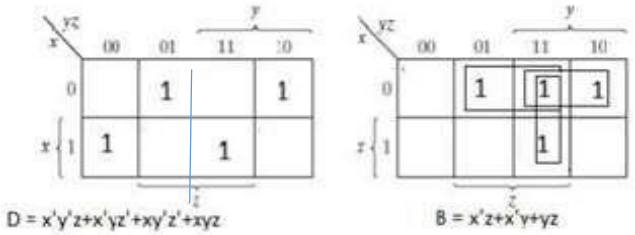
D = x ⊕ y
B=x'y

Full Subtractor

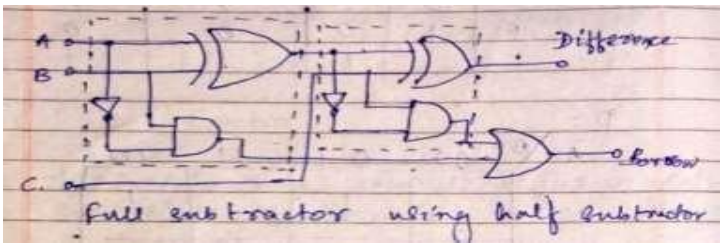
A full subtractor is a combinational circuit that forms the arithmetic subtraction operation of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be subtracted. The third input, z , is subtracted from the result 0f the first subtraction.

x	y	z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth Table



K-Map for full Subtractor



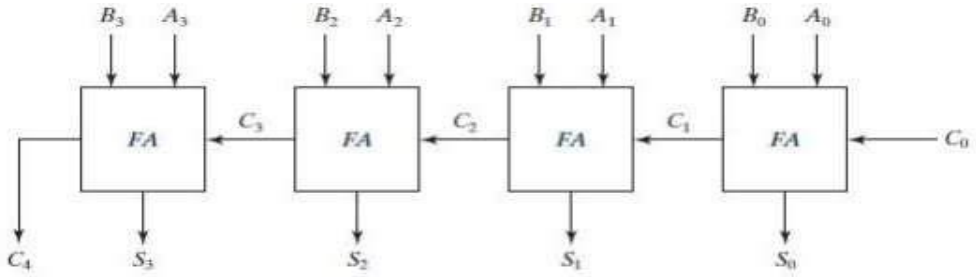
The binary variable D gives the value of the least significant bit of the difference. The binary variable B gives the output borrow formed during the subtraction process.

Parallel Binary 4 bit adder

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.

It can be constructed with full adders 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. The interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder is shown in the figure.



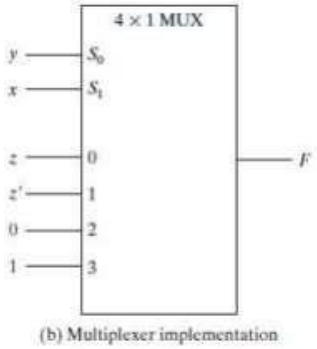
Four Bit Binary Adder

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 carries are connected in a chain through the full adders. The input carry to the adder is C0, and it ripples through the full adders to the output carry C4. The S outputs generate the required sum bits.

Multiplexer (4:1)

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.

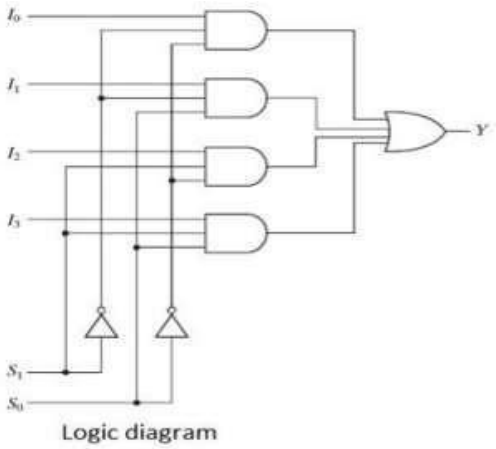
• The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. A four-to-one-line multiplexer is shown in the below figure.



(b) Multiplexer implementation

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

Truth table



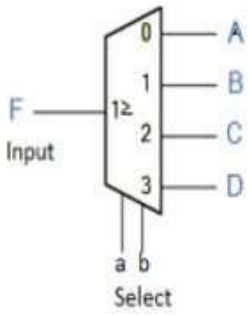
Logic diagram

Each of the four inputs, I0 through I3, is applied to one input of an AND gate. Selection lines S1 and S0 are decoded 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 inputs and steers the binary information to the output line.

De- multiplexer (1:4)

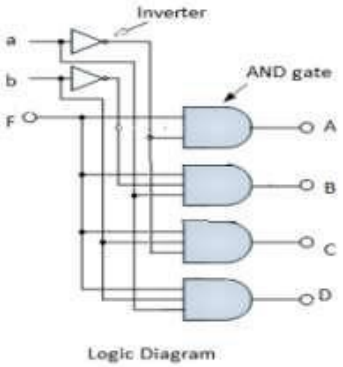
The data distributor, known more commonly as a Demultiplexer or "Demux" for short, is the exact opposite of the Multiplexer. • The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown below.



Output

Output Select		Data output Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

Truth Table



Logic Diagram

The function of the demultiplexer is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" as shown

Decoder

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

• The decoders presented here are called n -to- m -line decoders, where m … 2n. Their purpose is to generate the 2^n (or fewer) minterms of n input variables. Each combination of inputs will assert a unique output. The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.

A two-to-four-line decoder with an enable input constructed with NAND gates is shown in Fig. • The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0 (i.e., active-low enable). As indicated by the truth table, only one output can be equal to 0 at any given time; all other outputs are equal to 1. • The output whose value is equal to 0 represents the minterm selected by inputs A and B. • The circuit is disabled when E is equal to 1, regardless of the values of the other two inputs. • When the circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected.

Unit-3: Sequential logic Circuits

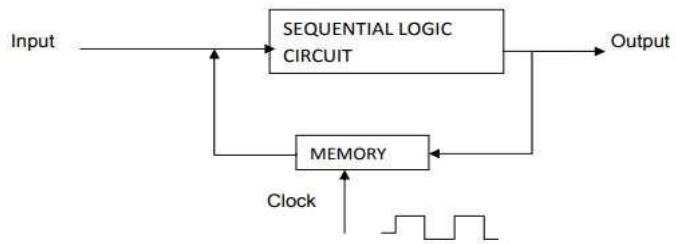
SEQUENTIAL LOGIC CIRCUIT SEQUENTIAL CIRCUIT:

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

HOW THE SEQUENTIAL CIRCUIT IS DIFFERENT FROM COMBINATIONAL CIRCUIT?

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

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



TYPES:-

Sequential logic circuits (SLC) are classified as (i) Synchronous SLC (ii) Asynchronous SLC

- The SLC that are controlled by clock are called synchronous SLC and those which are not controlled by a clock are asynchronous SLC.

- **Clock**:- A recurring pulse is called a clock.

FLIP-FLOP AND LATCH:-

- A flip-flop or 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 element in digital circuit can maintain a binary state indefinitely until directed by an input signal to switch state.

- Storage element that operate with signal level are called latches and those operate with clock transition are called as flip-flops. SEQUENTIAL LOGIC CIRCUIT MEMORY

- The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs.

- A flip-flop is called so because its output either flips or flops meaning to switch back and forth.

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

- Flip-flops are storage devices and can store 1 or 0.

- Flip-flops using the clock signal are called clocked flip-flops. Control signals are effective only if they are applied in synchronization with the clock signal.

- 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.

Types of flip-flops include

a) SR (set-reset) F-F

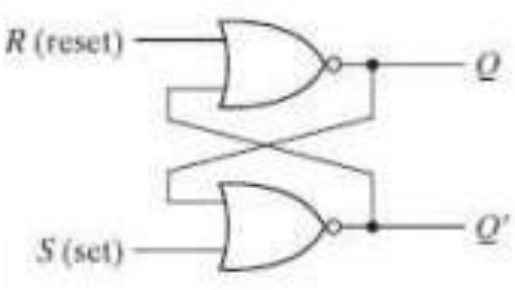
b) D (data or delay) F-F

c) T (toggle) F-F and

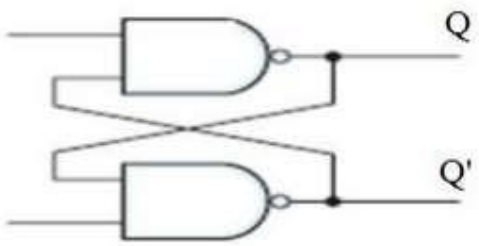
d) JK F-F

SR latch using NOR gate:-

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 Q'. Two inputs are there labeled S for set and R foe reset.
• The latch has two useful states. When Q=0 and Q'=1 the condition is called reset state and when Q=1 and Q'=0 the condition is called set state. • Normally Q and Q' are complement of each other



SR latch using NAND gate:-

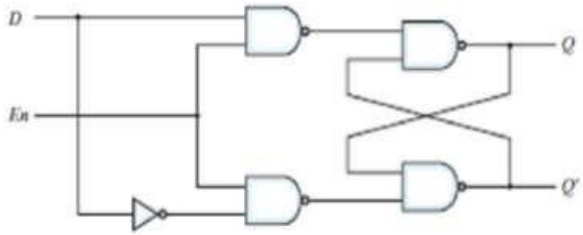


Input		Output		Comment
S	R	Q	Q_{Next}	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Prohibited state
1	1	1	X	

Racing Condition:-

In case of a SR latch when S=R=1 input is given both the output will try to become 0. This is called Racing condition.

D LATCH:-



Unit-4: Registers, Memories & PLD

Shift Registers-Serial in Serial -out, Serial- in Parallel-out, Parallel in serial out and Parallel in parallel out

REGISTERS INTRODUCTION:-

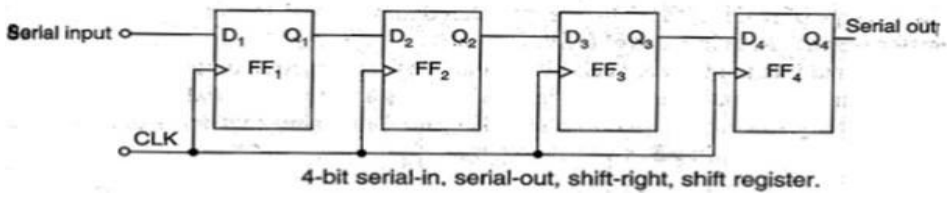
• The sequential circuits known as register, are used for storage and transfer of binary information in a digital system.

• A register has no characteristics internal sequence of states.

• The storage capacity of a register is defined as the number of bits of digital data, it can store or retain.

SERIAL IN, SERIAL OUT SHIFT REGISTER:-

• This type of shift register accepts data serially, i.e., one bit at a time and also outputs data serially. • The logic diagram of a four bit serial in, serial out shift register is shown in below figure:



4-bit serial-in, serial-out, shift-right, shift register.

• In 4 stages i.e. with 4 FFs, the register can store upto 4 bits of data.

• Serial data is applied at the D input of the first FF. The Q output of the first FF is connected to the D input of the second FF, the output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF.

The data is outputted from the Q terminal of the last FF.

• When a serial data is transferred to a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse.

• The bit that is previously stored by the first FF is transferred to the second FF. • The bit that is stored by the second FF is transferred to the third FF, and so on.

• The bit that was stored by the last FF is shifted out.

COUNTER

A **digital counter**, or simply counter, is a semiconductor device that is used for counting the number of times that a digital event has occurred. The counter's output is indexed by one LSB every time the counter is clocked.

A counter is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. A counter that follows a binary number is called binary counter.

There are two types of counters

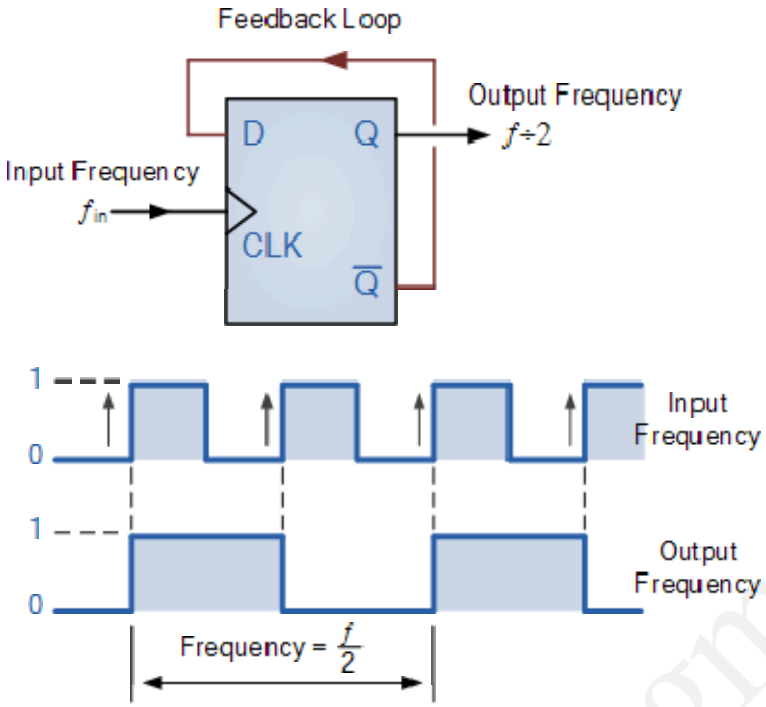
1. Asynchronous Counter
2. Synchronous Counter

In ***Asynchronous counters***, (ripple counter) the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. In ***Synchronous counters,*** the clock input is connected to all of the flip-flop so that they are clocked simultaneously.

Frequency Division

First, let us discuss about principle behind the counters. In the **Sequential Logic** tutorials we discussed how D-type Flip-Flop´s work and how they can be connected together to form a Data Latch. Another useful feature of the D-type Flip-Flop is as a binary divider, for **Frequency Division** or as a "divide-by-2" counter. Here the inverted output terminal Q (NOT-Q) is connected directly back to the Data input terminal D giving the device "feedback" as shown below.

Divide-by-2 Counter



It can be seen from the frequency waveforms above, that by "feeding back" the output from Q to the input terminal D, the output pulses at Q have a frequency that are exactly one half (f÷2) that of the input clock frequency. In other words the circuit produces **Frequency Division** as it now divides the input frequency by a factor of two (an octave). This then produces a type of counter called a "ripple counter" and in ripple counters, the clock pulse triggers the first flip-flop whose output triggers the second flip-flop, which in turn triggers the third flip-flop and so on through the chain.

Toggle Flip-Flop

Another type of device that can be used for frequency division is the T-type or Toggle flip-flop. With a slight modification to a standard JK flip-flop, we can construct a new type of flip-flop called a **Toggle flip-flop,** where the two inputs J and k of a JK flip-flop are connected together resulting in a device with only two inputs, the "**Toggle**" input itself and the controlling "Clock" input. The name "Toggle flip-flop" indicates the fact

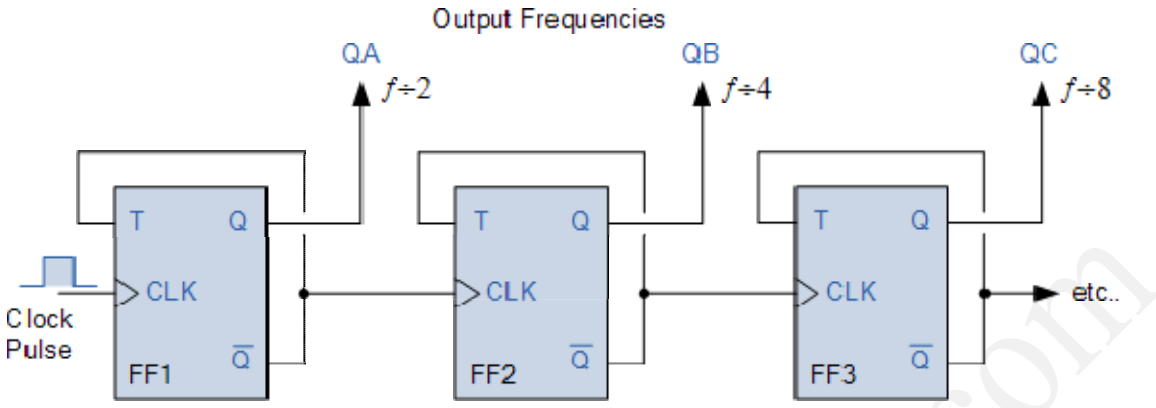
that the flip-flop has the ability to toggle between its two states, the "toggle state" and the "memory state". Since there are only two states, a T-type flip-flop is ideal for use in frequency division and counter design.

Binary ripple counters can be built using "Toggle" or "T-type flip-flops" by connecting the output of one to the clock input of the next. Toggle flip-flops are ideal for building ripple counters as it toggles from one state to the next, (HIGH to LOW or LOW to HIGH) at every clock cycle so simple frequency divider and ripple counter circuits can easily be constructed using standard T-type flip-flop circuits.

If we connect together in series, two T-type flip-flops the initial input frequency will be "divided-by-two" by the first flip-flop (f÷2) and then "divided-by-two" again by the second flip-flop (f÷2)÷2, giving an output frequency which has effectively been divided four times, then its output frequency becomes one quarter value (25%) of the original clock frequency, (f÷4). Each time we add another toggle or "T-type" flip-flop the output clock frequency is halved or divided-by-2 again and so on, giving an output frequency of 2^n where "n" is the number of flip-flops used in the sequence.

Then the Toggle or T-type flip-flop is an edge triggered divide-by-2 device based upon the standard JK-type flip flop and which is triggered on the rising edge of the clock signal. The result is that each bit moves right by one flip-flop. All the flip-flops can be asynchronously reset and can be triggered to switch on either the leading or trailing edge of the input clock signal making it ideal for **Frequency Division**.

Frequency Division using Toggle Flip-flops



This type of counter circuit used for frequency division is commonly known as an **Asynchronous 3-bit Binary Counter** as the output on Q_A to Q_C, which is 3 bits wide, is a binary count from 0 to 7 for each clock pulse. In an asynchronous counter, the clock is applied only to the first stage with the output of one flip-flop stage providing the clocking signal for the next flip-flop stage and subsequent stages derive the clock from the previous stage with the clock pulse being halved by each stage.

This arrangement is commonly known as ***Asynchronous*** as each clocking event occurs independently as all the bits in the counter do not all change at the same time. As the counter counts sequentially in an upwards direction from 0 to 7. This type of counter is also known as an "up" or "forward" counter (**CTU**) or a **"3-bit Asynchronous Up Counter"**. The three-bit asynchronous counter shown is typical and uses flip-flops in the toggle mode. Asynchronous "Down" counters (**CTD**) are also available.

Then we can see that the output from the D-type flip-flop is at half the frequency of the input, in other words it counts in 2's. By cascading together more D-type or Toggle Flip-Flops, we can produce a divide-by-2, divide-by-4, divide-by-8, etc. circuit which will divide the input clock frequency by 2, 4 or 8 times, in fact any value to the power-of-2 we want making a binary counter circuit.

Truth Table for a 3-bit Asynchronous Up Counter

Clock Cycle	Output bit Pattern		
	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

Modulo Counters

A counter is nothing more than a specialized register or pattern generator that produces a specified output pattern or sequence of binary values (or states) upon the application of an input pulse signal called the "Clock". The clock is actually used for data transfer in these applications. Typically, counters are logic circuits that can increment or decrement a count by one but when used as asynchronous divide-by-n counters they are able to divide these input pulses producing a clock division signal.

Counters are formed by connecting flip-flops together and any number of flip-flops can be connected or "cascaded" together to form a "divide-by-n" binary counter where "n" is the number of counter stages used and which is called the **Modulus**. The modulus or simply "MOD" of a counter is the number of output states the counter goes through before returning itself back to zero, i.e., one complete cycle.

Then a counter with three flip-flops like the circuit above will count from 0 to 7 i.e., 2^n-1. It has eight different output states representing the decimal

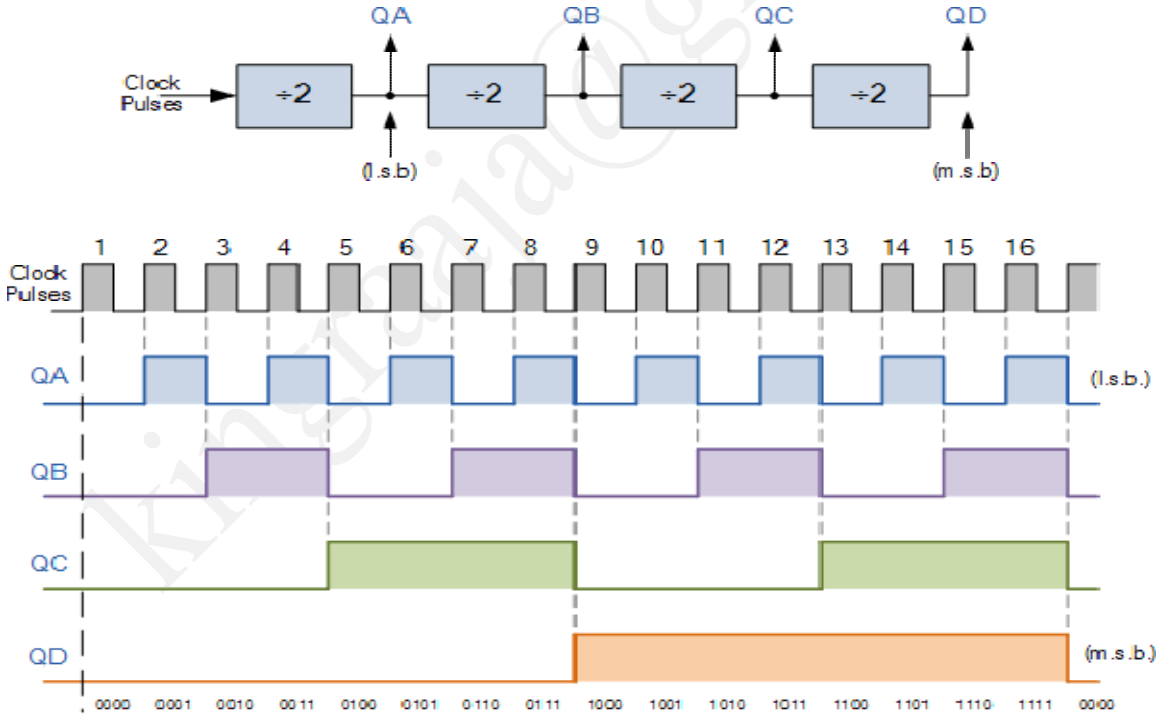
numbers 0 to 7 and is called a **Modulo-8** or **MOD-8** counter. A counter with four flip-flops will count from 0 to 15and is therefore called a **Modulo-16** counter and so on.

An example of this is given as.

- 3-bit Binary Counter = 2^3 = 8 (modulo-8 or MOD-8)
- 4-bit Binary Counter = 2^4 = 16 (modulo-16 or MOD-16)
- 8-bit Binary Counter = 2^8 = 256 (modulo-256 or MOD-256)

The Modulo number can be increased by adding more flip-flops to the counter and cascading is a method of achieving higher modulus counters. Then the modulo or MOD number can simply be written as: MOD number = 2^n

4-bit Modulo-16 Counter



Multi-bit asynchronous counters connected in this manner are also called **"Ripple Counters"** or ripple dividers because the change of state at each stage appears to "ripple" itself through the counter from the LSB output to its MSB output connection.

Frequency Division Summary

For **frequency division**, toggle mode flip-flops are used in a chain as a divide by two counter. One flip-flop will divide the clock, f in by 2, two flip-flops will divide f in by 4 (and so on). One benefit of using toggle flip-flops for frequency division is that the output at any point has an exact 50% duty cycle.

The final output clock signal will have a frequency value equal to the input clock frequency divided by the MOD number of the counter. Such circuits are known as "divide-by-n" counters. Counters can be formed by connecting individual flip-flops together and are classified according to the way they are clocked.

In ***Asynchronous counters***, (ripple counter) the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. In ***Synchronous counters,*** the clock input is connected to all of the flip-flop so that they are clocked simultaneously.

****

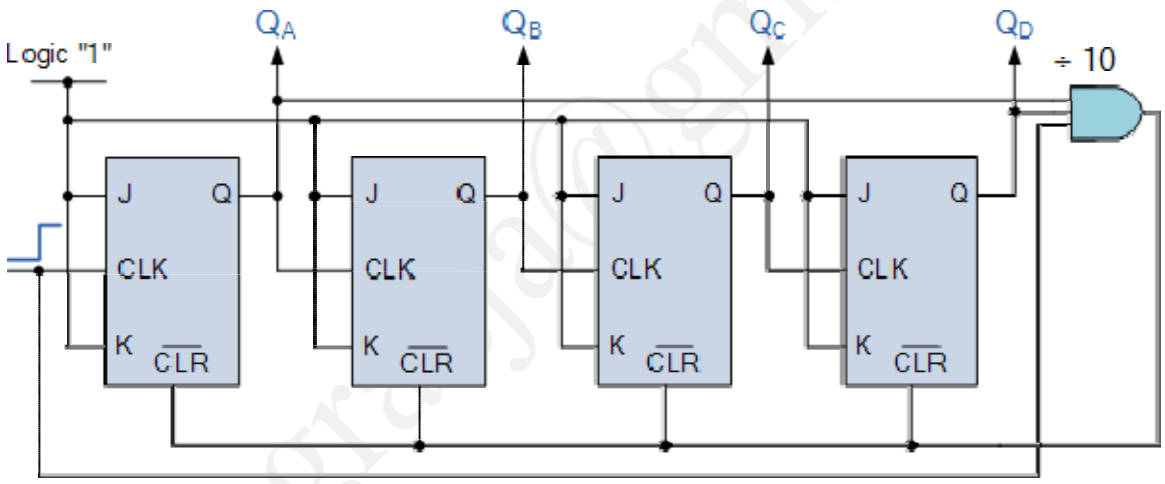
Asynchronous Counter

In the previous tutorial we discussed that an **Asynchronous counter** can have 2^n-1 possible counting states e.g. MOD-16 for a 4-bit counter, (0-15) making it ideal for use in **Frequency Division**. But it is also possible to use the basic asynchronous counter to construct special counters with counting states less than their maximum output number by forcing the counter to reset itself to zero at a pre-determined value producing a type of asynchronous counter that has truncated sequences. Then an n-bit counter that counts up to its maximum modulus (2^n) is called a full sequence counter and a n-bit counter whose modulus is less than the maximum possible is called a **truncated counter**.

If we take the modulo-16 asynchronous counter and modified it with additional logic gates it can be made to give a decade (divide-by-10) counter output for use in standard decimal counting and arithmetic circuits.

Such counters are generally referred to as **Decade Counters**. A decade counter requires resetting to zero when the output count reaches the decimal value of 10, i.e. when DCBA = 1010 and to do this we need to feed this condition back to the reset input. A counter with a count sequence from binary "0000" (BCD = "0") through to "1001" (BCD = "9") is generally referred to as a BCD binary-coded-decimal counter because its ten state sequence is that of a BCD code but binary decade counters are more common.

Asynchronous Decade Counter (mod-10)



This type of asynchronous counter counts upwards on each leading edge of the input clock signal starting from "0000" until it reaches an output "1010" (decimal 10). Both outputs Q_B and Q_D are now equal to logic "1" and the output from the NAND gate changes state from logic "1" to a logic "0" level and whose output is also connected to the CLEAR (CLR) inputs of all the J-K Flip-flops.

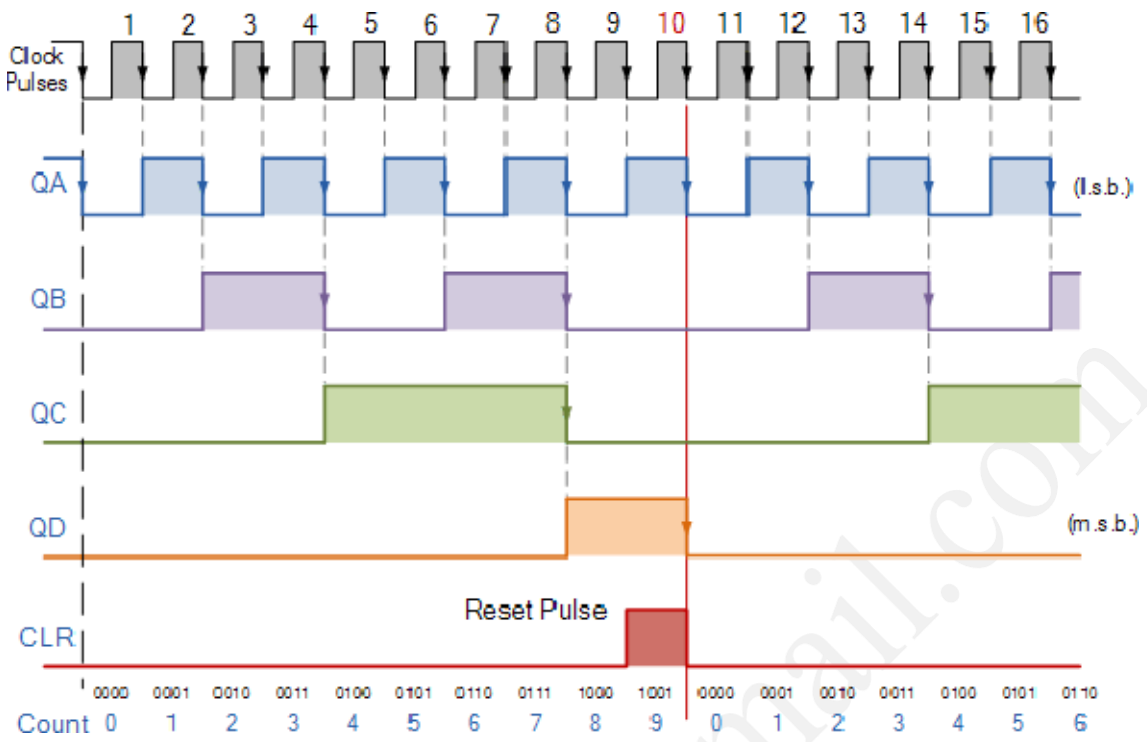
This signal causes all of the Q outputs to be reset back to binary "0000" on the count of 10. Once Q_B and Q_D are both equal to logic "0" the output of the NAND gate

returns back to a logic level "1" and the counter restarts again from "0000". We now have a decade or **Modulo-10** counter.

Decade Counter Truth Table

Clock Count	Output bit Pattern				Decimal Value
	Q_D	Q_C	Q_B	QA	
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9
11	Counter Resets its Outputs back to Zero				

Decade Counter Timing Diagram



Using the same idea of truncating counter output sequences, the above circuit could easily be adapted to other counting cycles be simply changing the connections to the AND gate. For example, a scale-of-twelve (modulo-12) can easily be made by simply taking the inputs to the AND gate from the outputs at "Q_C" and "Q_D", noting that the binary equivalent of 12 is "1100" and that output "Q_A" is the least significant bit (LSB).

Since the maximum modulus that can be implemented with n flip-flops is 2n, this means that when you are designing truncated asynchronous counters you should determine the lowest power of two that is greater than or equal to your desired modulus. For example, if you wish to count from 0 to 39, or mod-40. Then the highest number of flip-flops required would be six, n = 6 giving a maximum MOD of 64 as five flip-flops would only equal MOD-32.

Unfortunately one of the main disadvantages with asynchronous counters is that there is a small delay between the arrival of the clock pulse at its input and it being

present at its output due to the internal circuitry of the gate. In asynchronous circuits this delay is called the **Propagation Delay** giving the asynchronous ripple counter the nickname of **"propagation counter"** and in some high frequency cases this delay can produce false output counts.

In large bit ripple counter circuits, if the delay of the separate stages are all added together to give a summed delay at the end of the counter chain the difference in time between the input signal and the counted output signal can be very large. This is why the **Asynchronous Counter** is generally not used in high frequency counting circuits were large numbers of bits are involved.

Also, the outputs from the counter do not have a fixed time relationship with each other and do not occur at the same instant in time due to their clocking sequence. In other words the output frequencies become available one by one, a sort of domino effect. Then, the more flip-flops that are added to an asynchronous counter chain the lower the maximum operating frequency becomes to ensure accurate counting. To overcome the problem of propagation delay Synchronous Counters were developed.

Summary:

- **Asynchronous Counters** can be made from Toggle or D-type flip-flops.
- They are called asynchronous counters because the clock input of the flip-flops are not all driven by the same clock signal.
- Each output in the chain depends on a change in state from the previous flip-flops output.
- Asynchronous counters are sometimes called ripple counters because the data appears to "ripple" from the output of one flip-flop to the input of the next.
- They can be implemented using "divide-by-n" circuits.
- Truncated counters can produce any modulus number count.

Disadvantages of Asynchronous Counters:

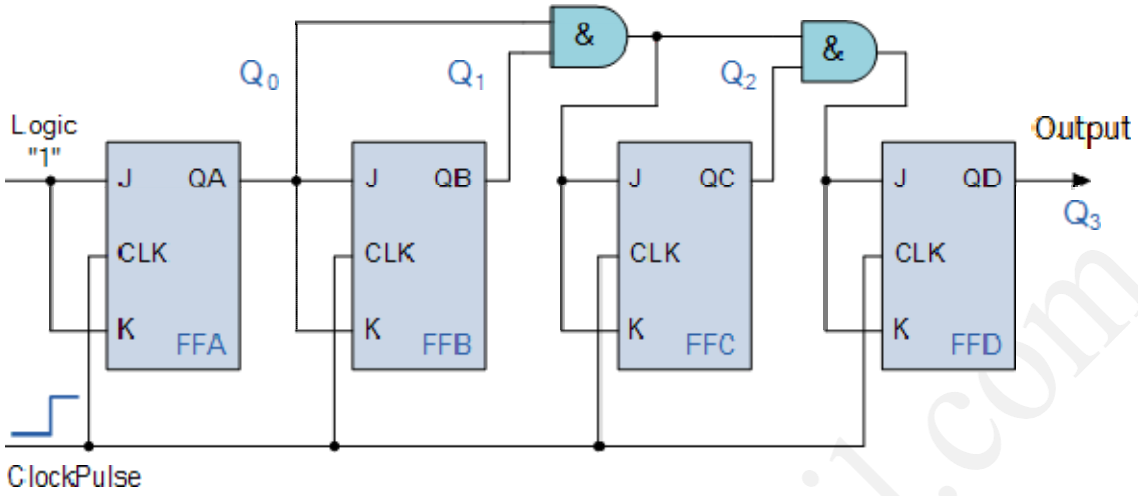
- An extra "re-synchronizing" output flip-flop may be required.
- To count a truncated sequence not equal to 2^n, extra feedback logic is required.
- Counting a large number of bits, propagation delay by successive stages may become undesirably large.
- This delay gives them the nickname of "Propagation Counters".
- Counting errors at high clocking frequencies.
- Synchronous Counters are faster using the same clock signal for all flip-flops.

Synchronous Counter

In the previous Asynchronous binary counter tutorial, we discussed that the output of one counter stage is connected directly to the clock input of the next counter stage and so on along the chain, and as a result the asynchronous counter suffers from what is known as "Propagation Delay" in which the timing signal is delayed a fraction through each flip-flop.

However, with the **Synchronous Counter**, the external clock signal is connected to the clock input of EVERY individual flip-flop within the counter so that all of the flip-flops are clocked together simultaneously (in parallel) at the same time giving a fixed time relationship. In other words, changes in the output occur in "synchronization" with the clock signal. This results in all the individual output bits changing state at exactly the same time in response to the common clock signal with no ripple effect and therefore, no propagation delay.

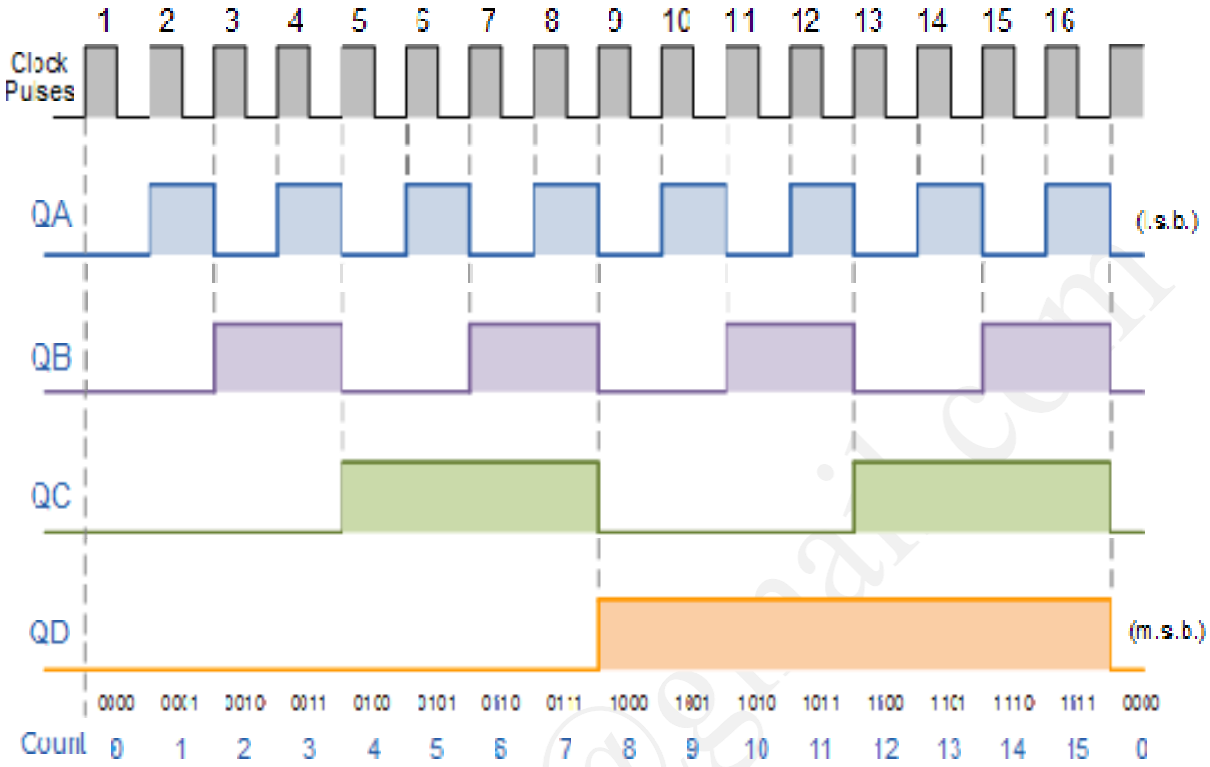
Binary 4-bit Synchronous Counter



It can be seen that the external clock pulses (pulses to be counted) are fed directly to each **J-K flip-flop** in the counter chain and that both the J and K inputs are all tied together in toggle mode, but only in the first flip-flop, flip-flop A (LSB) are they connected HIGH, logic "1" allowing the flip-flop to toggle on every clock pulse. Then the synchronous counter follows a predetermined sequence of states in response to the common clock signal, advancing one state for each pulse.

The J and K inputs of flip-flop B are connected to the output "Q" of flip-flop A, but the J and K inputs of flip-flops C and D are driven from AND gates which are also supplied with signals from the input and output of the previous stage. If we enable each J-K flip-flop to toggle based on whether or not all preceding flip-flop outputs (Q) are "HIGH" we can obtain the same counting sequence as with the asynchronous circuit but without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time. As there is no propagation delay in synchronous counters because all the counter stages are triggered in parallel the maximum operating frequency of this type of counter is much higher than that of a similar asynchronous counter.

4-bit Synchronous Counter Timing Diagram.



Because this 4-bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 ("0000") to 15 ("1111"). Therefore, this type of counter is also known as a **4-bit Synchronous Up Counter**.

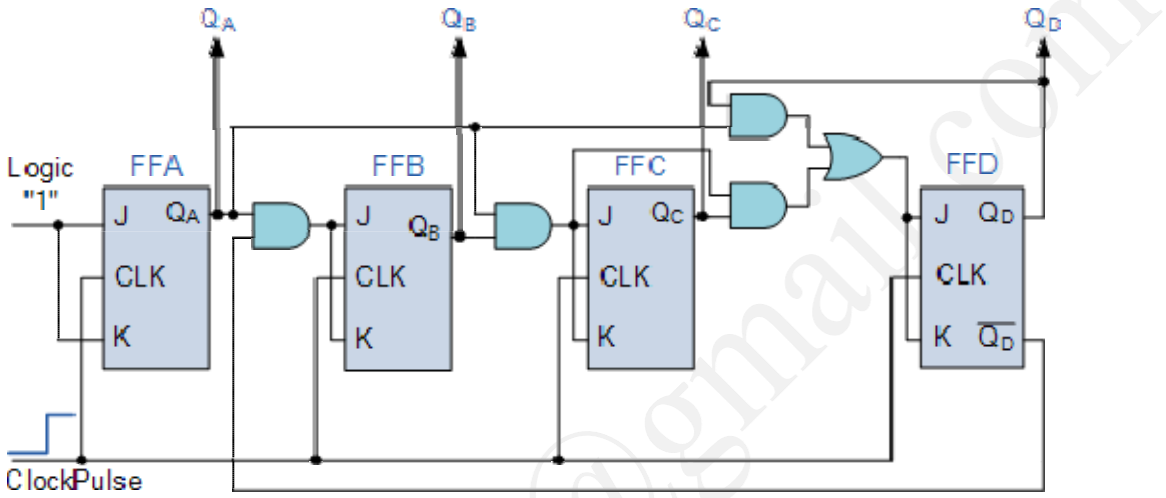
As synchronous counters are formed by connecting flip-flops together and any number of flip-flops can be connected or "cascaded" together to form a "divide-by-n" binary counter, the modulo's or "MOD" number still applies as it does for asynchronous counters so a Decade counter or BCD counter with counts from 0 to 2^n-1 can be built along with truncated sequences.

Decade 4-bit Synchronous Counter

Another name for Decade Counter is Modulo 10 counter. A 4-bit decade synchronous counter can also be built using synchronous binary counters to produce a

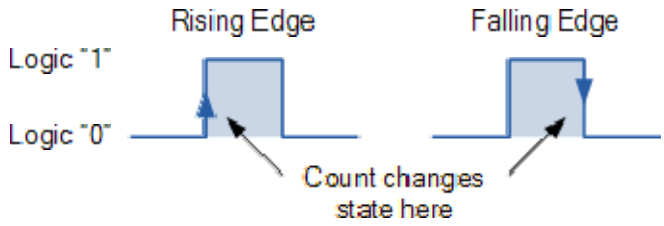
count sequence from 0 to 9. A standard binary counter can be converted to a decade (decimal 10) counter with the aid of some additional logic to implement the desired state sequence. After reaching the count of "1001", the counter recycles back to "0000". We now have a decade or **Modulo-10** counter.

Decade 4-bit Synchronous Counter



The additional AND gates detect when the sequence reaches "1001", (Binary 10) and causes flip-flop FF3to toggle on the next clock pulse. Flip-flop FF0 toggles on every clock pulse. Thus, the count starts over at "0000" producing a synchronous decade counter. We could quite easily re-arrange the additional ANDgates to produce other counters such as a Mod-12 Up counter which counts 12 states from"0000" to "1011" (0 to 11) and then repeats making them suitable for clocks.

Synchronous Counters use edge-triggered flip-flops that change states on either the "positive-edge" (rising edge) or the "negative-edge" (falling edge) of the clock pulse on the control input resulting in one single count when the clock input changes state. Generally, synchronous counters count on the rising-edge which is the low to high transition of the clock signal and asynchronous ripple counters count on the falling-edge which is the high to low transition of the clock signal.



It may seem unusual that ripple counters use the falling-edge of the clock cycle to change state, but this makes it easier to link counters together because the most significant bit (MSB) of one counter can drive the clock input of the next. This works because the next bit must change state when the previous bit changes from high to low - the point at which a carry must occur to the next bit. Synchronous counters usually have a carry-out and a carry-in pin for linking counters together without introducing any propagation delays.

Summary:

- **Synchronous Counters** can be made from Toggle or D-type flip-flops.
- They are called synchronous counters because the clock input of the flip-flops are clocked with the same clock signal.
- Due to the same clock pulse all outputs change simultaneously.
- Synchronous counters are also called parallel counters as the clock is fed in parallel to all flip-flops.
- Synchronous binary counters use both sequential and combinational logic elements.
- The memory section keeps track of the present state.
- The sequence of the count is controlled by combinational logic.

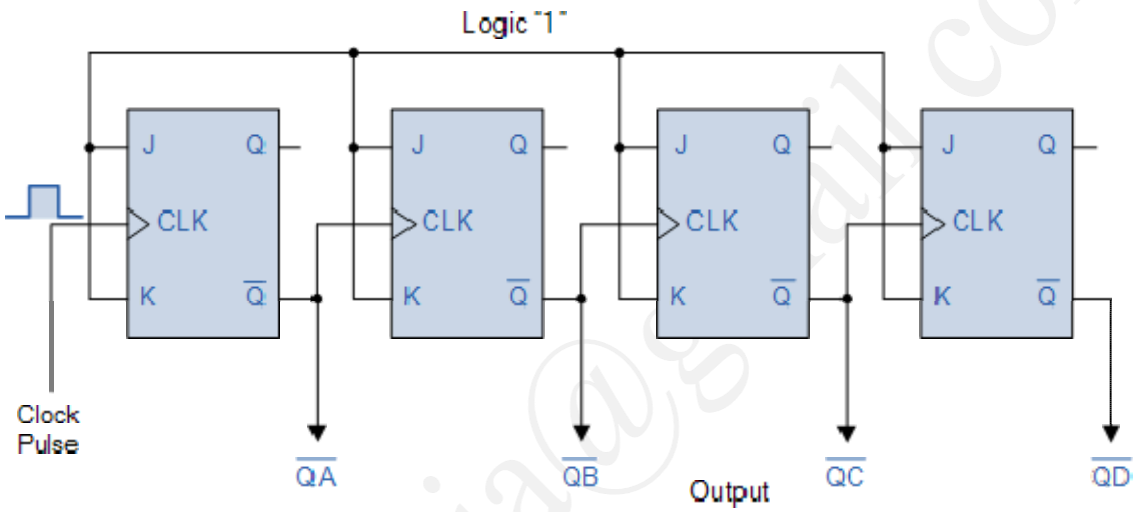
Advantages of Synchronous Counters:

- Synchronous counters are easier to design.
- With all clock inputs wired together there is no inherent propagation delay.
- Overall faster operation may be achieved compared to Asynchronous counters.

Count Down Counter

As well as counting "up" from zero and increase, or increment to some value, it is sometimes necessary to count "down" from a predetermined value to zero and to produce an output that activates when the zero count or other pre-set value is reached. This type of counter is normally referred to as a **Down Counter**, (**CTD**).

4-bit Count Down Counter



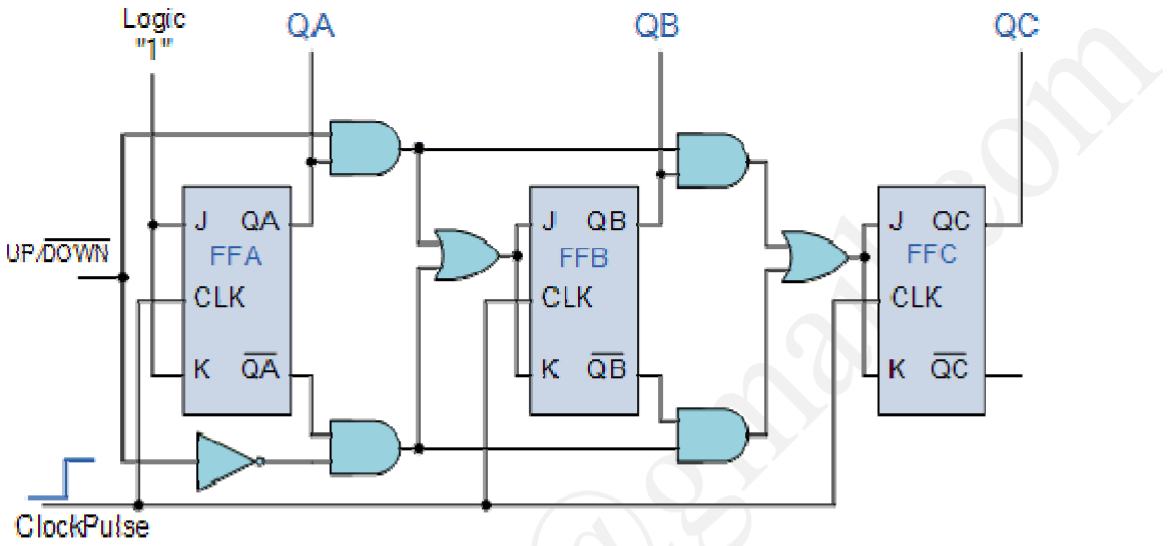
In the 4-bit counter above the output of each flip-flop changes state on the falling edge (1-to-0 transition) of the CLK input which is triggered by the Q output of the previous flip-flop, rather than by the Q output as in the up counter configuration. As a result, each flip-flop will change state when the previous one changes from 0 to 1 at its output, instead of changing from 1 to 0.

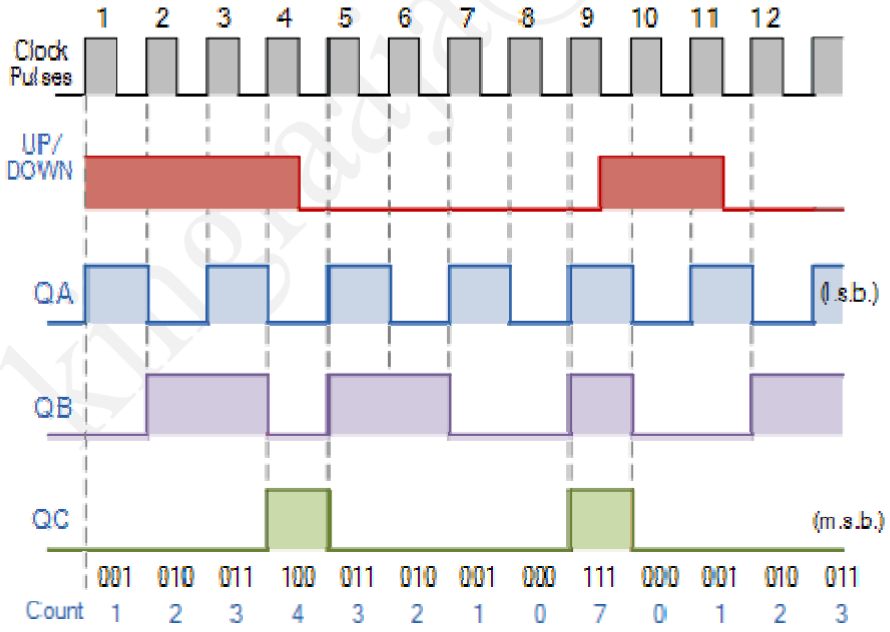
Bidirectional Counter(up/down)

Both Synchronous and Asynchronous counters are capable of counting "Up" or counting "Down", but their is another more "Universal" type of counter that can count in both directions either Up or Down depending on the state of their input control pin and these are known as **Bidirectional Counters**. Bidirectional counters, also known as

Up/Down counters, are capable of counting in either direction through any given count sequence and they can be reversed at any point within their count sequence by using an additional control input as shown below.

Synchronous 3-bit Up/Down Counter





The circuit above is of a simple 3-bit Up/Down synchronous counter using JK flip-flops configured to operate as toggle or T-type flip-flops giving a maximum count of zero

(000) to seven (111) and back to zero again. Then the 3-Bit counter advances upward in sequence (0,1,2,3,4,5,6,7) or downwards in reverse sequence (7,6,5,4,3,2,1,0) but generally, bidirectional counters can be made to change their count direction at any point in the counting sequence. An additional input determines the direction of the count, either Up or Down and the timing diagram gives an example of the counters operation as this Up/Down input changes state.

Nowadays, both up and down counters are incorporated into single IC that is fully programmable to count in both an "Up" and a "Down" direction from any preset value producing a complete **Bidirectional Counter** chip.

****