

Lecture Notes

Unit:4

Digital Electronics Fundamentals

Topics Covered:

Difference between analog and digital signals, Boolean algebra, Basic and Universal Gates, Symbols, Truth tables, logic expressions, Logic simplification using K- map, Logic ICs, half and full adder/subtractor, multiplexers, demultiplexers, flip-flops, shift registers, counters, Block diagram of microprocessor/microcontroller and their applications.

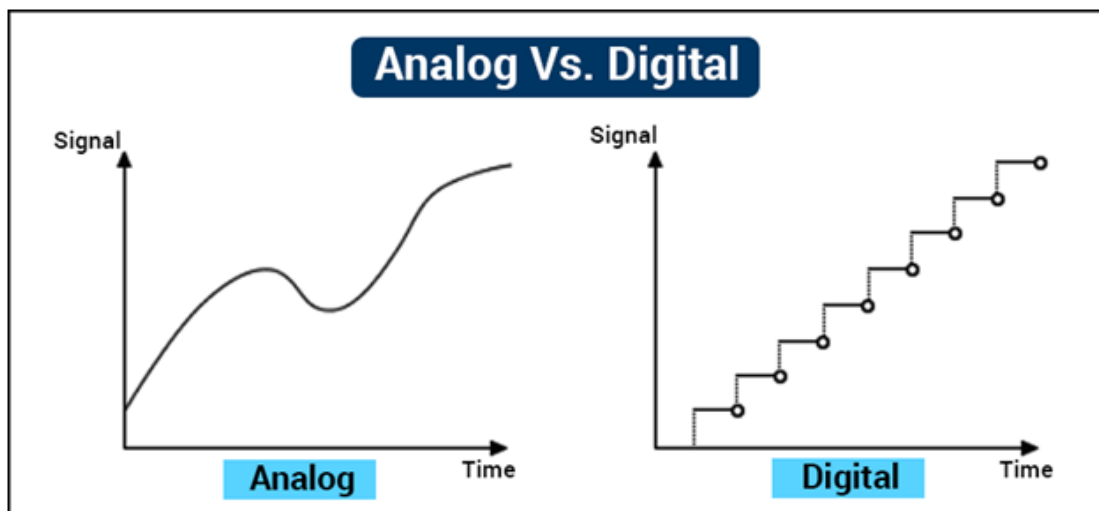
Designed by: Vipra Bohara

Assistant Professor

JECRC, Sitapura, Jaipur.

Analog And Digital Signal

Analog and digital signals are the types of signals carrying information. The major difference between both signals is that the analog signals that have continuous electrical signals, while digital signals have non-continuous electrical signals. The difference between analog and digital signal can be observed by given figure.



Analog Signals

The analog signals were used in many systems to produce signals to carry information. These signals are continuous in both values and time. The use of analog signals has been declined with the arrival of digital signals. In short, to understand the analog signals – all signals that are natural or come naturally are analog signals.

Digital Signals

Unlike analog signals, digital signals are not continuous, but signals are discrete in value and time. These signals are represented by binary numbers and consist of different voltage values.

Difference Between Analog And Digital Signal

Analog Signals	And	Digital Signal
Continuous signals		Discrete signals
Represented by sine waves		Represented by square waves
Human voice, natural sound, analog electronic devices are few examples		Computers, optical drives, and other electronic devices
Continuous range of values		Discontinuous values
Records sound waves as they are		Converts into a binary waveform.
Only be used in analog devices.		Suited for digital electronics like computers, mobiles and more.

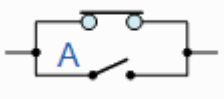
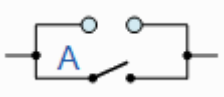
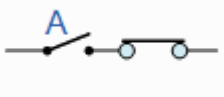

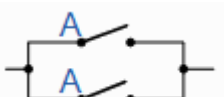
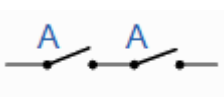
Boolean Algebra

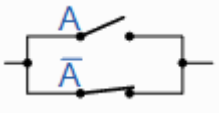
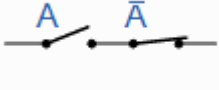
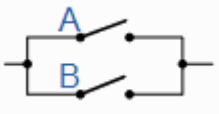
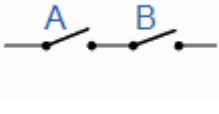
Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these “Laws of Boolean” to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic that has its own set of rules or laws which are used to define and reduce Boolean expressions.

The variables used in **Boolean Algebra** only have one of two possible values, a logic “0” and a logic “1” but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression, For example, variables A, B, C etc, giving us a logical expression of $A + B = C$, but each variable can ONLY be a 0 or a 1.

Examples of these individual laws of Boolean, rules and theorems for Boolean Algebra are given in the following table.

Truth Tables for the Laws of Boolean

Boolean Expression	Description	Equivalent Switching Circuit	Boolean Algebra Law or Rule
$A + 1 = 1$	A in parallel with closed = "CLOSED"		Annulment
$A + 0 = A$	A in parallel with open = "A"		Identity
$A \cdot 1 = A$	A in series with closed = "A"		Identity
$A \cdot 0 = 0$	A in series with open = "OPEN"		Annulment
$A + A = A$	A in parallel with A = "A"		Idempotent
$A \cdot A = A$	A in series with A = "A"		Idempotent

NOT A = A	NOT NOT A (double negative) = "A"		Double Negation
$A + \bar{A} = 1$	A in parallel with NOT A = "CLOSED"		Complement
$A \cdot \bar{A} = 0$	A in series with NOT A = "OPEN"		Complement
$A+B = B+A$	A in parallel with B = B in parallel with A		Commutative
$A \cdot B = B \cdot A$	A in series with B = B in series with A		Commutative
$\overline{(A + B)} = \bar{A} \cdot \bar{B}$	invert and replace OR with AND		de Morgan's Theorem
$\overline{(A \cdot B)} = \bar{A} + \bar{B}$	invert and replace AND with OR		de Morgan's Theorem

Description of the Laws of Boolean Algebra

- Annulment Law – A term AND'ed with a "0" equals 0 or OR'ed with a "1" will equal 1

- $A \cdot 0 = 0$ A variable AND'ed with 0 is always equal to 0
- $A + 1 = 1$ A variable OR'ed with 1 is always equal to 1

- Identity Law – A term OR'ed with a "0" or AND'ed with a "1" will always equal that term

- $A + 0 = A$ A variable OR'ed with 0 is always equal to the variable
- $A \cdot 1 = A$ A variable AND'ed with 1 is always equal to the variable

- Idempotent Law – An input that is AND'ed or OR'ed with itself is equal to that input

- $A + A = A$ A variable OR'ed with itself is always equal to the variable
- $A \cdot A = A$ A variable AND'ed with itself is always equal to the variable

- Complement Law – A term AND'ed with its complement equals "0" and a term OR'ed with its complement equals "1"

- $A \cdot \bar{A} = 0$ A variable AND'ed with its complement is always equal to 0
- $A + \bar{A} = 1$ A variable OR'ed with its complement is always equal to 1

- Commutative Law – The order of application of two separate terms is not important

- $A \cdot B = B \cdot A$ The order in which two variables are AND'ed makes no difference
- $A + B = B + A$ The order in which two variables are OR'ed makes no difference

- Double Negation Law – A term that is inverted twice is equal to the original term

- $\overline{\bar{A}} = A$ A double complement of a variable is always equal to the variable

- de Morgan's Theorem – There are two “de Morgan's” rules or theorems,

(1) Two separate terms NOR'ed together is the same as the two terms inverted (Complement) and AND'ed for example: $A+B = A \cdot B$

(2) Two separate terms NAND'ed together is the same as the two terms inverted (Complement) and OR'ed for example: $A \cdot B = A + B$

Other algebraic Laws of Boolean not detailed above include:

- Boolean Postulates – While not Boolean Laws in their own right, these are a set of Mathematical Laws which can be used in the simplification of Boolean Expressions.

$0 \cdot 0 = 0$ A 0 AND'ed with itself is always equal to 0

$1 \cdot 1 = 1$ A 1 AND'ed with itself is always equal to 1

$1 \cdot 0 = 0$ A 1 AND'ed with a 0 is equal to 0

$0 + 0 = 0$ A 0 OR'ed with itself is always equal to 0

$1 + 1 = 1$ A 1 OR'ed with itself is always equal to 1

$1 + 0 = 1$ A 1 OR'ed with a 0 is equal to 1

$\bar{1} = 0$ The Inverse (Complement) of a 1 is always equal to 0

$\bar{0} = 1$ The Inverse (Complement) of a 0 is always equal to 1

- Distributive Law – This law permits the multiplying or factoring out of an expression.

- $A(B + C) = A.B + A.C$ (OR Distributive Law)
- $A + (B.C) = (A + B).(A + C)$ (AND Distributive Law)

- Absorptive Law – This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.

- $A + (A.B) = (A.1) + (A.B) = A(1 + B) = A$ (OR Absorption Law)
- $A(A + B) = (A + 0).(A + B) = A + (0.B) = A$ (AND Absorption Law)

- Associative Law – This law allows the removal of brackets from an expression and regrouping of the variables.

- $A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law)
- $A(B.C) = (A.B)C = A . B . C$ (AND Associate Law)

EXAMPLE

Using the above laws, simplify the following expression: $(A + B)(A + C)$

$$\begin{aligned} Q &= (A + B)(A + C) \\ &= A.A + A.C + A.B + B.C && \text{- Distributive law} \\ &= A + A.C + A.B + B.C && \text{- Idempotent AND law (A.A = A)} \\ &= A(1 + C) + A.B + B.C && \text{- Distributive law} \\ &= A.1 + A.B + B.C && \text{- Identity OR law (1 + C = 1)} \\ &= A(1 + B) + B.C && \text{- Distributive law} \\ &= A.1 + B.C && \text{- Identity OR law (1 + B = 1)} \\ Q &= A + (B.C) && \text{- Identity AND law (A.1 = A)} \end{aligned}$$

Basic Logic Gates

Logic gates are an important concept if you are studying electronics. These are important digital devices that are mainly based on the Boolean function. Logic gates are used to carry out logical operations on single or multiple binary inputs and give one binary output. In simple terms, logic gates are the electronic circuits in a digital system.

In this lesson, we will further look at the different types of basic logic gates with their truth table and understand what each one is designed for.

Table of Content

- Types of Basic Logic Gates
- Truth Table
- Symbolic Representation
- Logic Expressions

Types of Basic Logic Gates

There are several basic logic gates used in performing operations in digital systems. The common ones are;

- **OR Gate**
- **AND Gate**
- **NOT Gate**
- **XOR Gate**

Additionally, these gates can also be found in a combination of one or two. Therefore we get other gates such as: NAND Gate, NOR Gate, EXOR Gate, EXNOR Gate.

OR Gate

In OR gate the output of an OR gate attains the state 1 if one or more inputs attain the state 1.



The Boolean expression of OR gate is $Y = A + B$, read as Y equals A 'OR' B.

The truth table of a two-input OR basic gate is given as;

A	B	Y (OUTPUT)
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate

In AND gate the output of an AND gate attains the state 1 if and only if all the inputs are in state 1.



The Boolean expression of AND gate is $Y = A.B$

The truth table of a two-input AND basic gate is given as;

A	B	Y(OUTPUT)
0	0	0
0	1	0
1	0	0
1	1	1

NOT Gate

In NOT gate the output of a NOT gate attains the state 1 if and only if the input does not attain the state 1.



The Boolean expression $Y = \bar{A}$, read as Y equals NOT A.

The truth table of NOT gate is as follows;

A	Y(OUTPUT)
0	1
1	0

The three gates (OR, AND and NOT), when connected in various combinations, give us basic logic gates such as NAND, NOR gates, which are the universal building blocks of digital circuits.

NAND Gate

This basic logic gate is the combination of AND and NOT gate.



The Boolean expression of NAND gate is $Y = \overline{(A \cdot B)}$

The truth table of a NAND gate is given as;

A	B	Y(OUTPUT)
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

This gate is the combination of OR and NOT gate.



The Boolean expression of NOR gate is $Y = \overline{A + B}$

The truth table of a NOR gate is as follows;

A	B	Y(OUTPUT)
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR gate (XOR Gate)

In XOR gate the output of a two-input XOR gate attains the state 1 if one adds only input attains the state 1.



The Boolean expression of the XOR gate is $A.\bar{B} + \bar{A}.B$ Or

$$Y = A \oplus B$$

The truth table of an XOR gate is;

A	B	Y(OUTPUT)
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR Gate (XNOR Gate)

In XNOR gate the output is in state 1 when its both inputs are the same that is, both 0 or both 1.



The Boolean expression of XNOR gate $Y = \overline{(A \oplus B)} = (A.B + \bar{A}.\bar{B})$

The truth table of an XNOR gate is given below;

A	B	Y(OUTPUT)
0	0	1
0	1	0
1	0	0
1	1	1

De Morgan's Theorem

First theorem – It states that the NAND gate is equivalent to a bubbled OR gate.

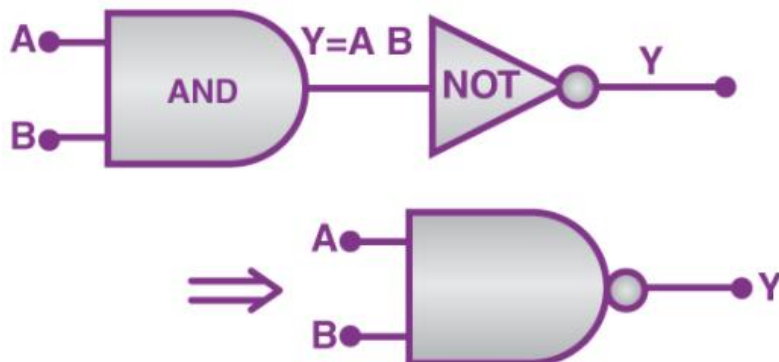
$$\overline{(A \cdot B)} = \bar{A} + \bar{B}$$

Second theorem – It states that the NOR gate is equivalent to a bubbled AND gate.

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

Important Conversions

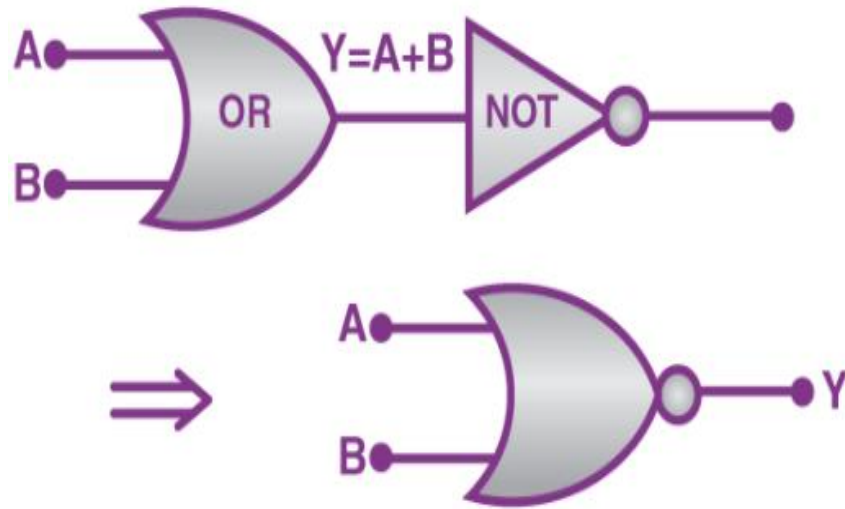
1) The 'NAND' gate: From 'AND' and 'NOT' gate.



Boolean expression and truth table : $Y = \overline{A \cdot B}$

A	B	A · B	$Y = \overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

(2) The 'NOR' gate: From 'OR' and 'NOT' gate

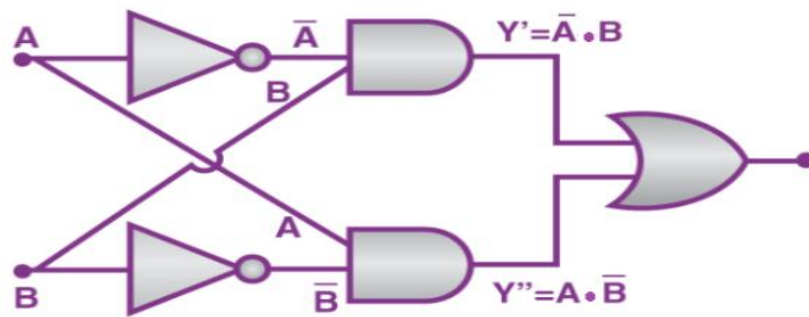


Boolean expression and truth table : $Y=\overline{A+B}$

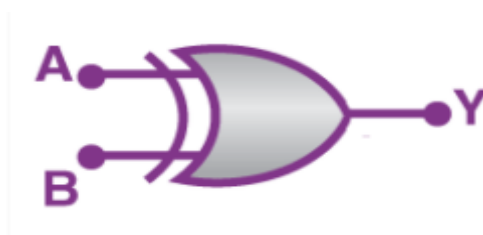
A	B	A+B	$Y=\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

(3) **The 'XOR' gate:** From 'NOT', 'AND' and 'OR' gate.

The logic gate which gives high output (i.e., 1) if either input A or input B but not both are high (i.e. 1) is called exclusive OR gate or the XOR gate. It may be noted that if both the inputs of the XOR gate are high, then the output is low (i.e., 0).



OR



Boolean expression and truth table: $A \cdot \bar{B} + \bar{A} \cdot B$

Or $Y = A \oplus B$

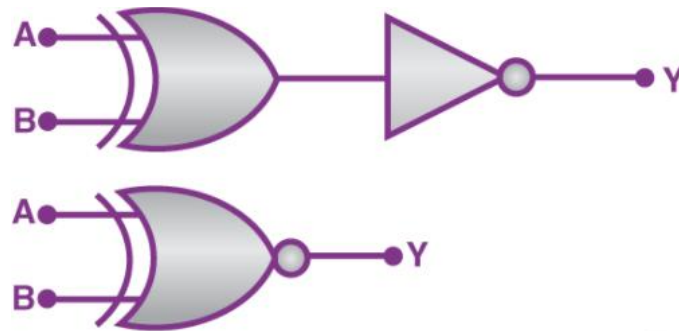
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

For $A = 0, B = 1$ the value of $\bar{A} = 1$ and $\bar{B} = 0$

Now $A \cdot \bar{B} = 0 \cdot 0 = 0$ and $\bar{A} \cdot B = 1 \cdot 1 = 1$

Thus, $Y = A \cdot \bar{B} + \bar{A} \cdot B = 0 + 1 = 1$

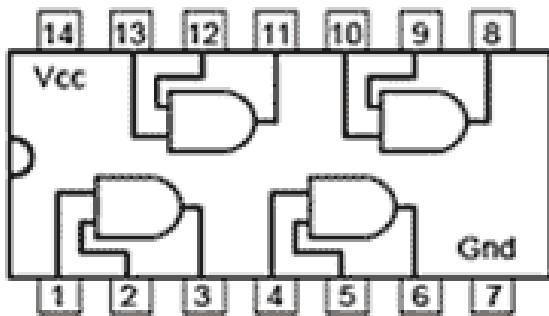
(4) The Exclusive nor (XNOR) gate XOR + NOT



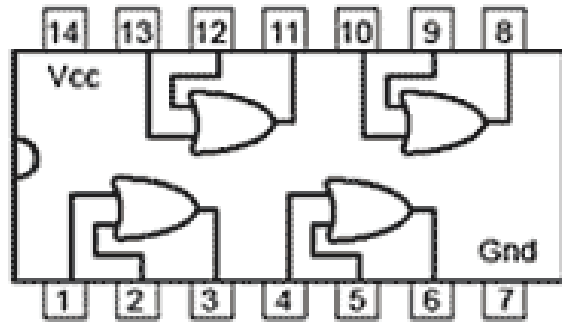
Boolean expression: $Y = (A \oplus B)$

Truth table of Exclusive nor (XNOR)

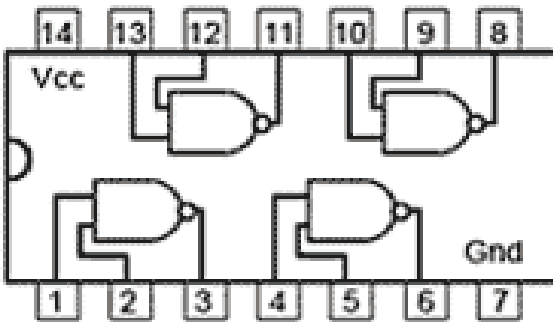
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1



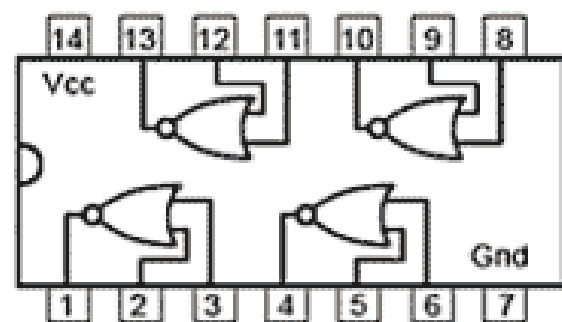
**7408 Quad 2 input
AND Gates**



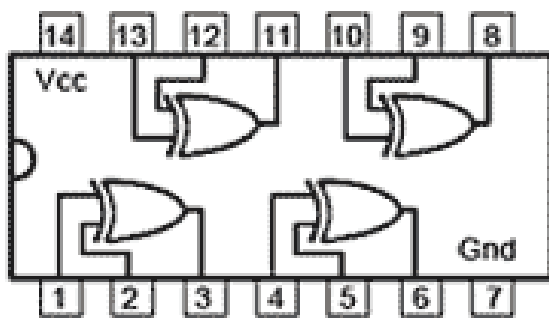
**7432 Quad 2 input
OR Gates**



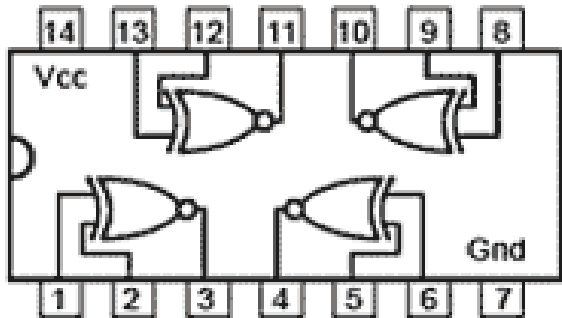
**7400 Quad 2 input
NAND Gates**



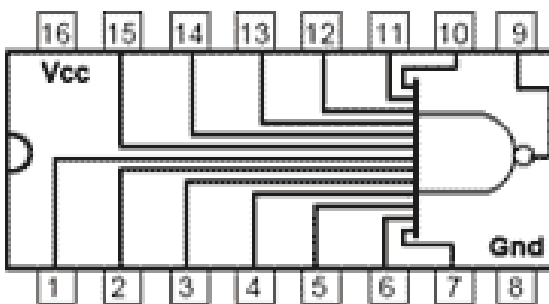
**7402 Quad 2 input
NOR Gates**



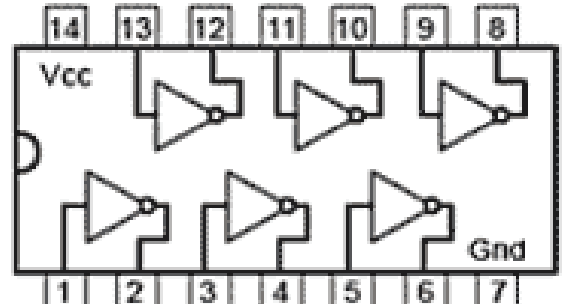
**7486 Quad 2 input
XOR Gates**



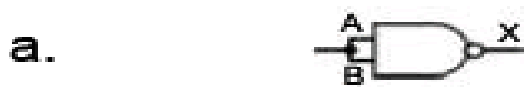
**747266 Quad 2 input
XNOR Gates**



**74133 Single 13 input
NAND Gate**



**7404 Hex NOT Gates
(Inverters)**



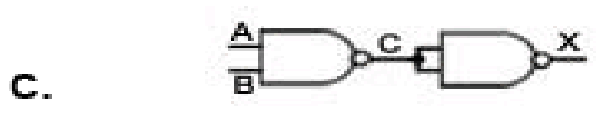
NOT

A	B	X
0	0	1
1	1	0



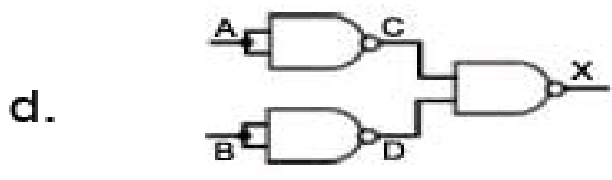
NOT

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



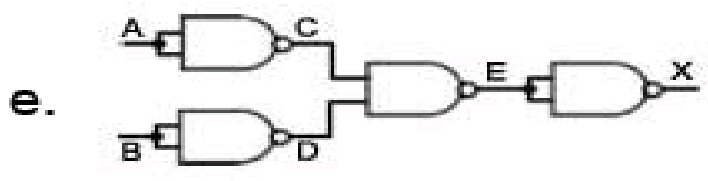
AND

A	B	C	X
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1



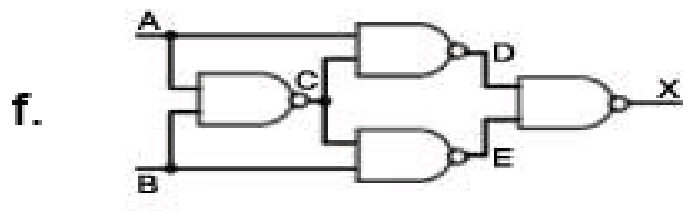
OR

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



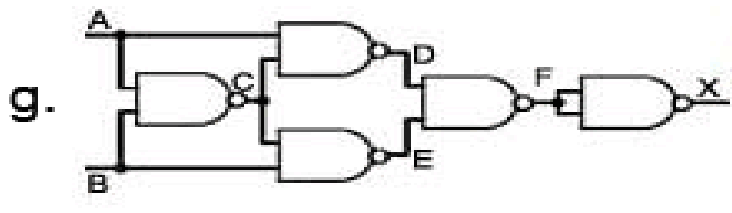
NOR

A	B	C	D	E	X
0	0	1	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	1	0



XOR

A	B	C	D	E	X
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0



XNOR

A	B	C	D	E	F	X
0	0	1	1	1	0	1
0	1	1	1	0	1	0
1	0	1	0	1	1	0
1	1	0	1	1	0	1

ASSIGNMENT:1

- Verify the Truth Table of XNOR GATE
- Show that $(A \cdot \bar{B} + C) \cdot (A+B) \cdot C = (A+\bar{B}) \cdot C$

In previous topic, we have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.

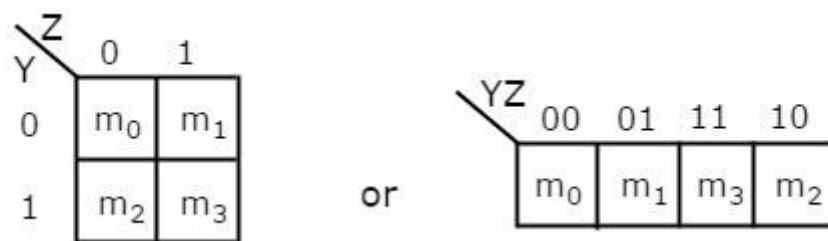
To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 5 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

2 Variable K-Map

The number of cells in 2 variable K-map is four, since the number of variables is two let Y and Z here. The following figure shows 2 variable K-Map.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are $\{(m_0, m_1), (m_2, m_3), (m_0, m_2) \text{ and } (m_1, m_3)\}$.

3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three let X, Y and Z here.. The following figure shows 3 variable K-Map.

		YZ			
		00	01	11	10
X	0	m ₀	m ₁	m ₃	m ₂
	1	m ₄	m ₅	m ₇	m ₆

- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m₀, m₁, m₃, m₂), (m₄, m₅, m₇, m₆), (m₀, m₁, m₄, m₅), (m₁, m₃, m₅, m₇), (m₃, m₂, m₇, m₆) and (m₂, m₀, m₆, m₄)}.
- The possible combinations of grouping 2 adjacent min terms are {(m₀, m₁), (m₁, m₃), (m₃, m₂), (m₂, m₀), (m₄, m₅), (m₅, m₇), (m₇, m₆), (m₆, m₄), (m₀, m₄), (m₁, m₅), (m₃, m₇) and (m₂, m₆)}.
- If x= 0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows 4 variable K-Map.

		YZ			
		00	01	11	10
WX	00	m ₀	m ₁	m ₃	m ₂
	01	m ₄	m ₅	m ₇	m ₆
	11	m ₁₂	m ₁₃	m ₁₅	m ₁₄
	10	m ₈	m ₉	m ₁₁	m ₁₀

- There is only one possibility of grouping 16 adjacent min terms.
- Let R1, R2, R3 and R4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C1, C2, C3 and C4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are $\{(R1, R2), (R2, R3), (R3, R4), (R4, R1), (C1, C2), (C2, C3), (C3, C4), (C4, C1)\}$.
- If $w=0$, then 4 variable K-map becomes 3 variable K-map.

5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variables is 5. The following figure shows 5 variable K-Map.

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

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

- There is only one possibility of grouping 32 adjacent min terms.
- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m_0 to m_{15} and m_{16} to m_{31} .
- If $v=0$, then 5 variable K-map becomes 4 variable K-map.
- In the above all K-maps, we used exclusively the min terms notation. Similarly, you can use exclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is in standard **sum of products** form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in standard **product of sums** form after simplifying the K-map.

Follow these **rules for simplifying K-maps** in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.
- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as prime implicant. The **prime implicant** is said to be **essential prime implicant**, if atleast single '1' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note 1 – If outputs are not defined for some combination of inputs, then those output values will be represented with **don't care** symbol 'x'. That means, we can consider them as either '**0**' or '**1**'.

Note 2 – If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1'.

Example

Let us simplify the following Boolean function, $f(W,X,Y,Z) = WX'Y' + WY + W'YZ'$ using K-map.

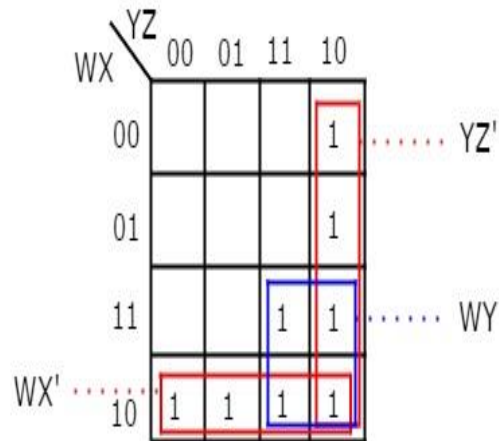
The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require 4 variable K-map. The 4 variable K-map with ones corresponding to the given product terms is shown in the following figure.

		YZ			
		00	01	11	10
WX	00				1
	01				1
	11			1	1
	10	1	1	1	1

Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, **$WX'Y'$** .
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, **WY** .
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, **$W'YZ'$** .

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping **4 adjacent** ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of **2 adjacent ones**. The 4 variable K-map with these three groupings is shown in the following figure.



Here, we got three prime implicants WX' , WY & YZ' . All these prime implicants are essential because of following reasons.

- Two ones (**m8 & m9**) of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.
- Single one (**m15**) of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.
- Two ones (**m2 & m6**) of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the simplified Boolean function is

$$f = WX' + WY + YZ'$$

Rules for Simplifying K-maps

Follow these rules for simplifying K-maps in order to get standard product of sums form.

- Select the respective K-map based on the number of variables present in the Boolean function.

- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one sum term. It is known as prime implicant. The **prime implicant** is said to be **essential prime implicant**, if atleast single '0' is not covered with any other groupings but only that grouping covers.
- **Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.**

Note – If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

ADDER (HALF/FULL Adder)

What is ADDER?

In electronics an adder is digital circuit that perform addition of numbers. In modern computer adder reside in the arithmetic logic unit (ALU).

Adders are important not only in the computer but also in many types of digital systems in which the numeric data are processed.

Types of adder:

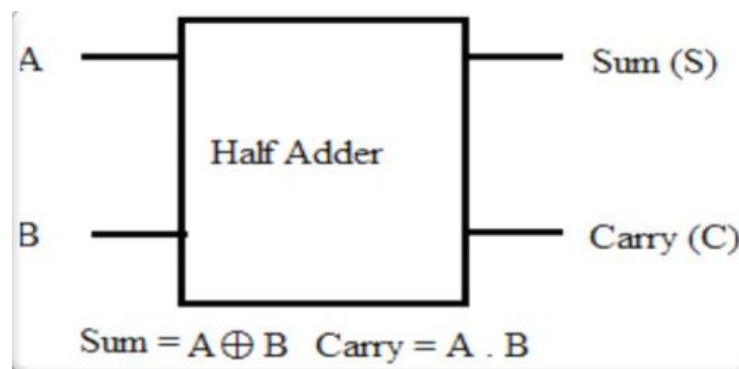
- Half adder
- Full adder

HALF ADDER :

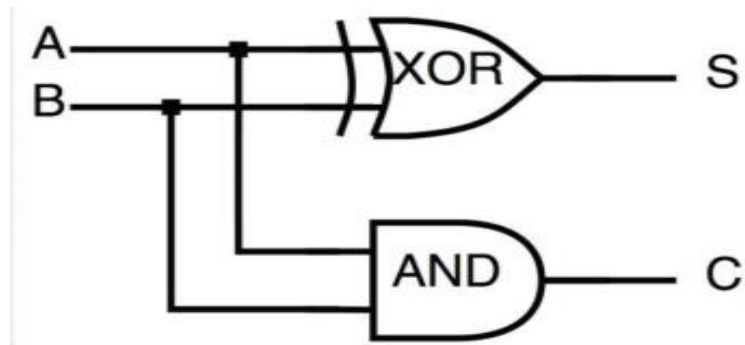
The half adder accepts **two** binary digits on its **inputs** and produce **two** binary digits **outputs**, a sum bit and a carry bit.

The half adder is an example of a simple, functional digital circuit built from **two logic gates**. The half adder adds **inputs** as one-bit binary numbers (A and B). The **output** is the **sum** of the two bits (S) and the **carry** (C).

The block diagram of half adder as shown in figure below:



Note that **how** the same two inputs are directed to two different gates.

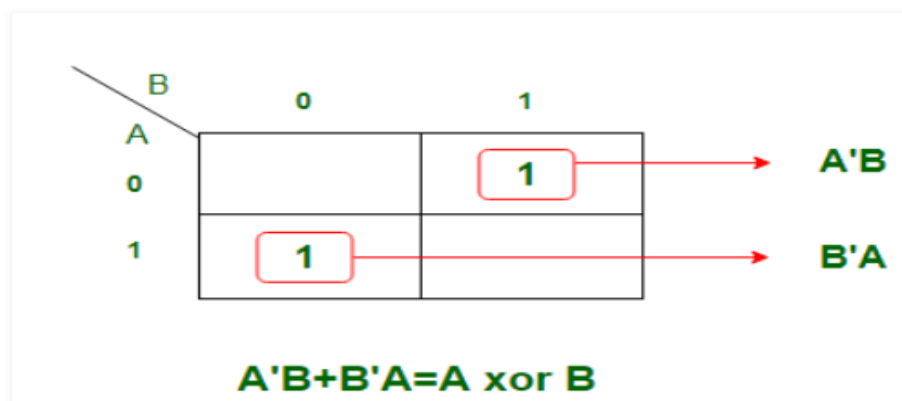


The inputs to the XOR gate are also the inputs to the AND gate. The input "wires" to the XOR gate are tied to the input wires of the AND gate; thus, when voltage is applied to the A input of the XOR gate, the A input to the AND gate receives the same voltage.

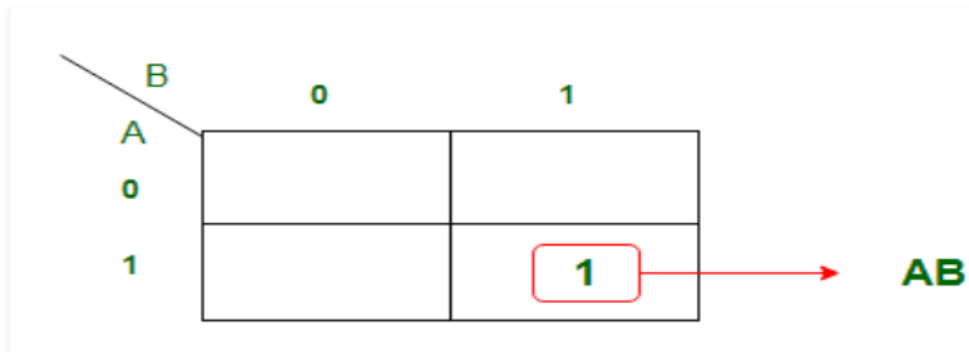
HALF ADDER Truth Table

A	B	Sum	Carry-Out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-map for output variable Sum 'S' :



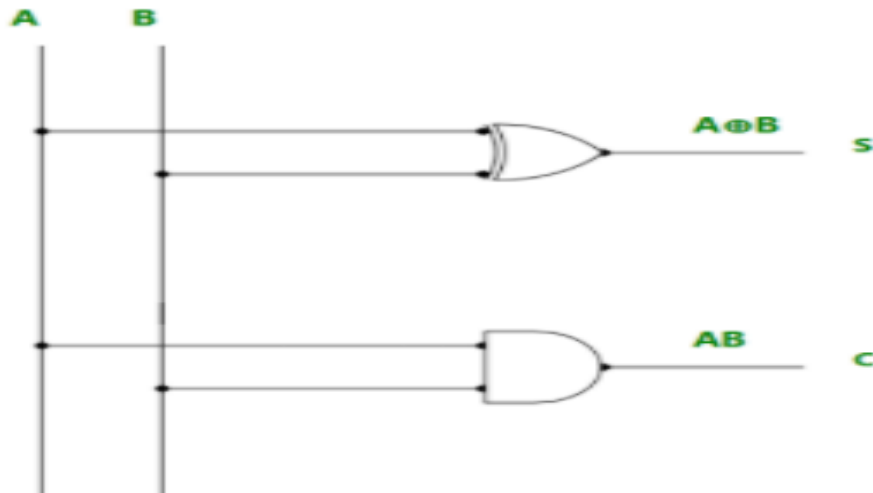
K-map for output variable Carry 'C' :



$$S = A \oplus B \text{ (Exclusive OR)}$$

$$C = A \cdot B \text{ (AND)}$$

Using the Boolean Expression, we can draw logic diagram as follows



Limitations:

Adding of Carry is not possible in Half adder.

FULL ADDER :

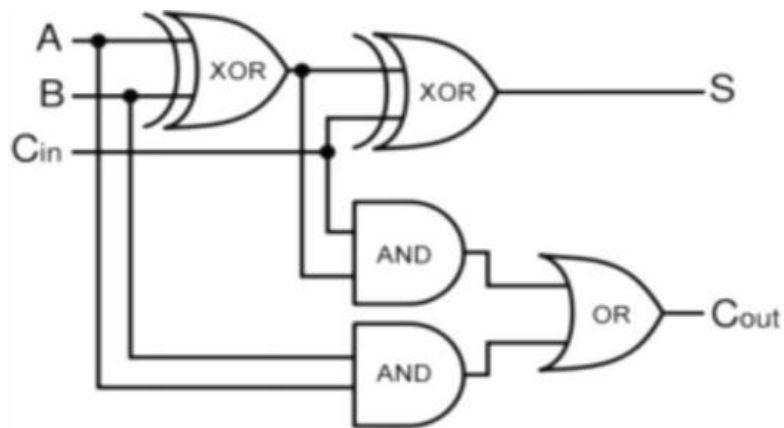
The full adder accepts two inputs bits and an input carry and generates a sum output and an output carry.

The full-adder circuit adds **three** one-bit binary numbers (**Cin, A ,B**) and outputs **two** one-bit binary numbers, a sum (**S**) and a carry (**Cout**). The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.

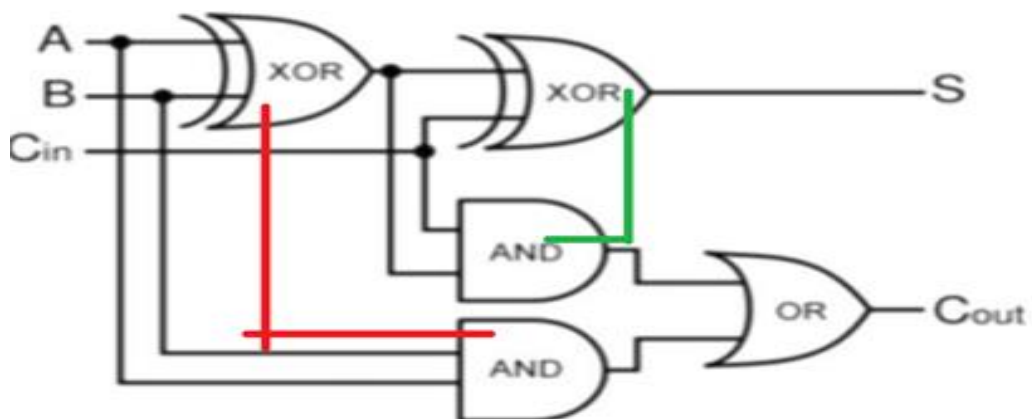
The block diagram of full adder as shown in figure below:



The Circuit diagram of full adder as shown in figure below:



If you look closely, you'll see the full adder is simply **two half** adders joined by an OR as given in figure below.



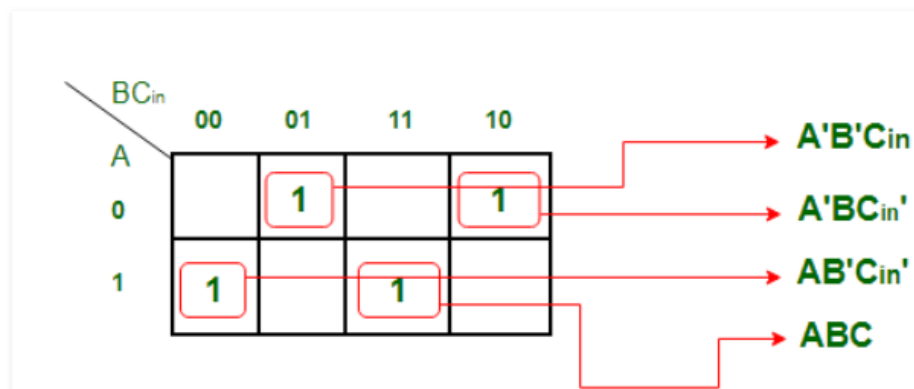
So We can **implement** a **full adder** circuit with the help of **two half adder** circuits. The **first half** adder will be used to add A and B to produce a **partial**

Sum. The **second half** adder logic can be used to add **Cin** to the Sum produced by the first half adder to get the final **S output**. If any of the half adder logic produces a carry, there will be an output carry. Thus, **Cout** will be an OR function of the half-adder Carry outputs.

FULL ADDER Truth Table

A	B	Carry-In	Sum	Carry-Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-map Simplification for output variable Sum 'S' :



The equation obtained is,

$$S = A'B'C_{in} + AB'C_{in}' + ABC + A'BC_{in}'$$

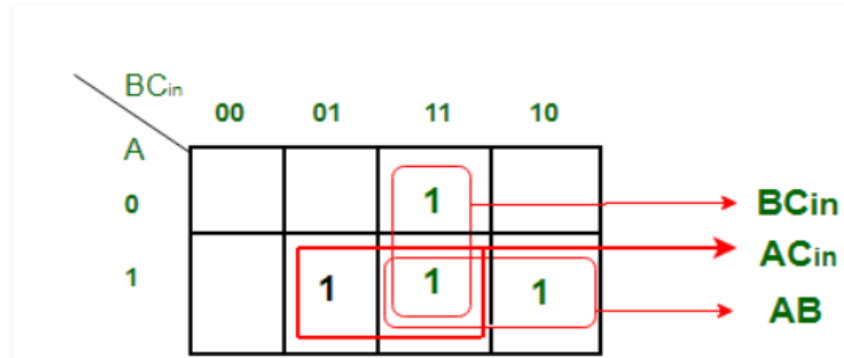
The equation can be simplified as,

$$S = B'(A'C_{in} + AC_{in}') + B(AC + A'C_{in}')$$

$$S = B'(A \text{ xor } C_{in}) + B(A \text{ xor } C_{in})'$$

$$S = A \text{ xor } B \text{ xor } C_{in}$$

K-map Simplification for output variable 'C_{out}'



The equation obtained is,

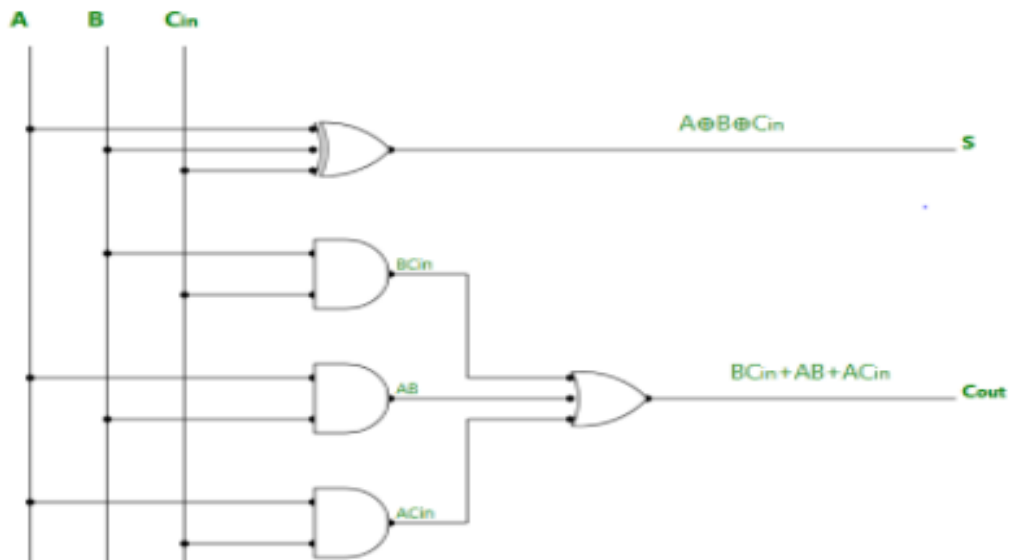
$$C_{out} = BC_{in} + AB + AC_{in}$$

Thus,

$$S = A \oplus B \oplus C_{in}$$

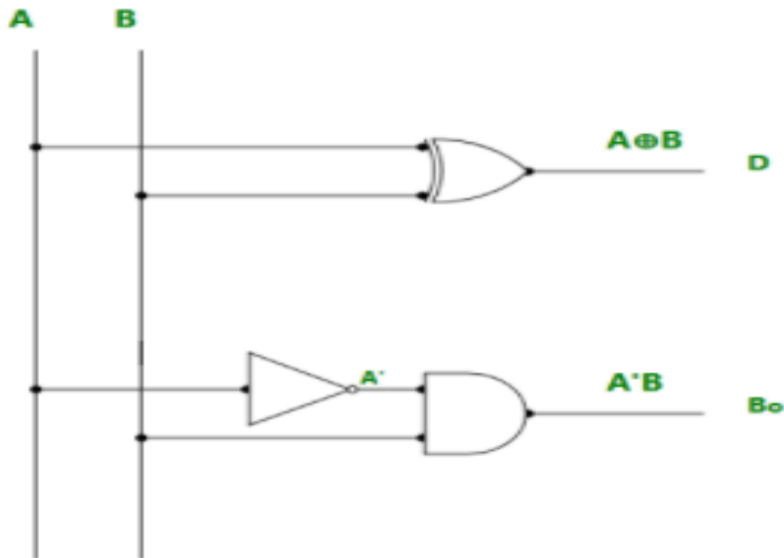
$$C = AB + C_{in} (A \oplus B)$$

Using the Boolean Expression, we can draw logic diagram as follows

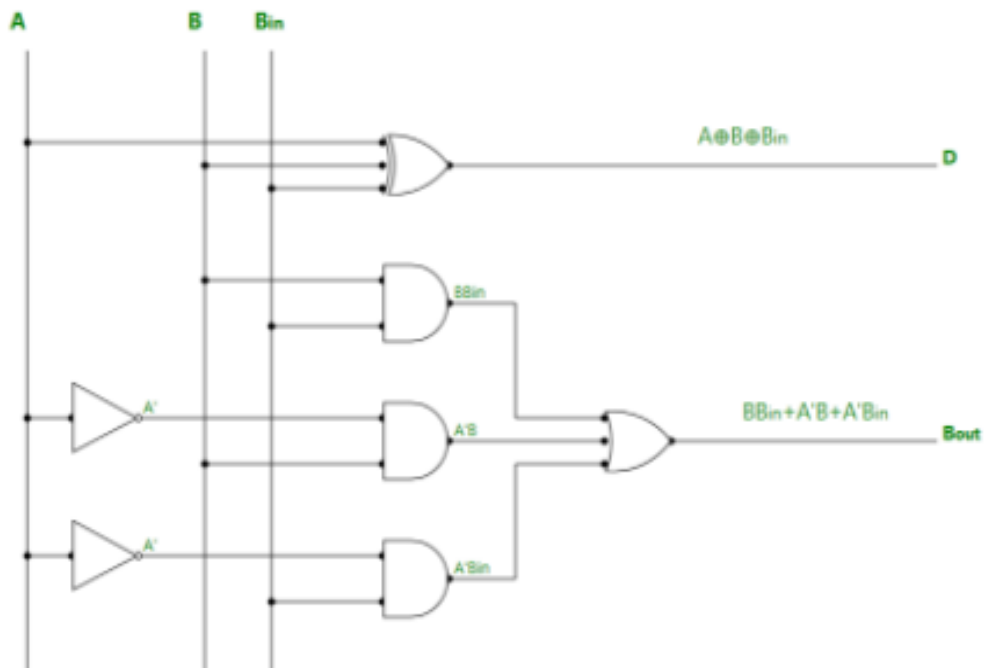


ASSIGNMENT:2

- Verify the Logic Diagram of Half Subtractor as:



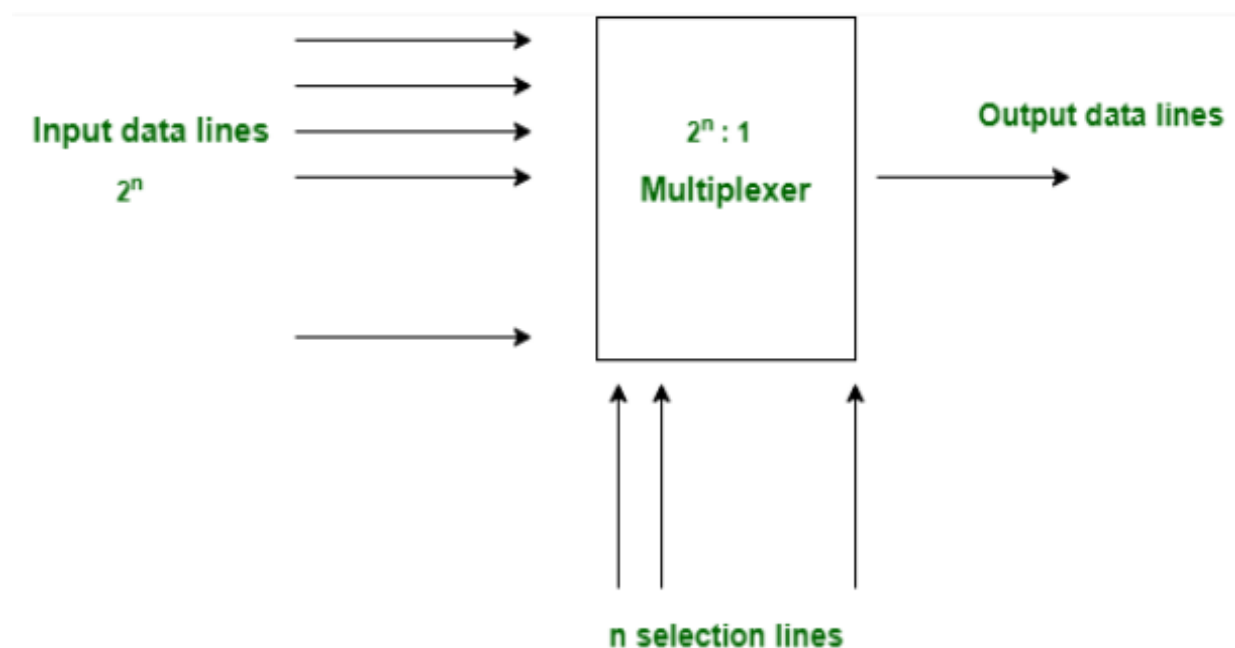
- Verify the Logic Diagram of Full Subtractor as:



Multiplexer

The multiplexer is a device that has **multiple inputs and single line output**. The select lines determine which input is connected to the output, and also increase the amount of data that can be sent over a network within a certain time. It is also called a **data selector**. In multiplexer we have **2^n input lines** and **1 output lines** where n is the number of selection lines.

The **single-pole multi-position switch** is a simple example of a non-electronic circuit of the multiplexer, and it is widely used in many electronic circuits. The multiplexer is used to perform high-speed switching and is constructed by electronic components.



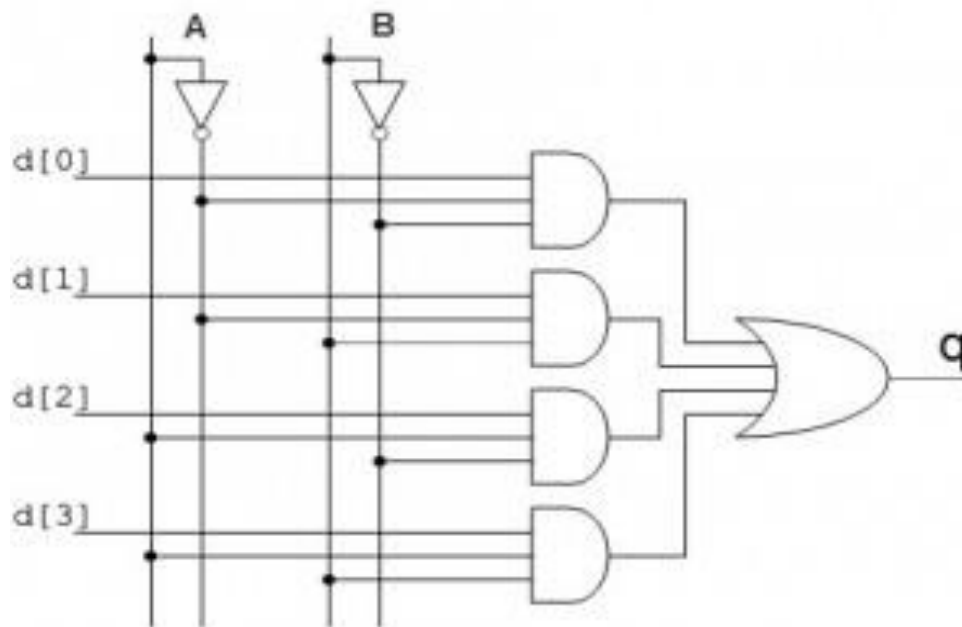
Multiplexer Types:

Multiplexers are classified into four types:

- 2-1 multiplexer (1select line)
- 4-1 multiplexer (2 select lines)
- 8-1 multiplexer(3 select lines)
- 16-1 multiplexer (4 select lines)

4-to-1 Multiplexer

The 4X1 multiplexer comprises **4-input bits, 1- output bit, and 2- control bits**. The four input bits are namely D0, D1, D2, and D3, respectively; only one of the input bits is transmitted to the output. The o/p 'q' depends on the value of control input AB. The control bit AB decides which of the i/p data bit should transmit the output. The following figure shows the 4X1 multiplexer circuit diagram using AND gates. For example, when the control bits AB =00, then the higher AND gates are allowed while remaining AND gates are restricted. Thus, data input D0 is transmitted to the output 'q'

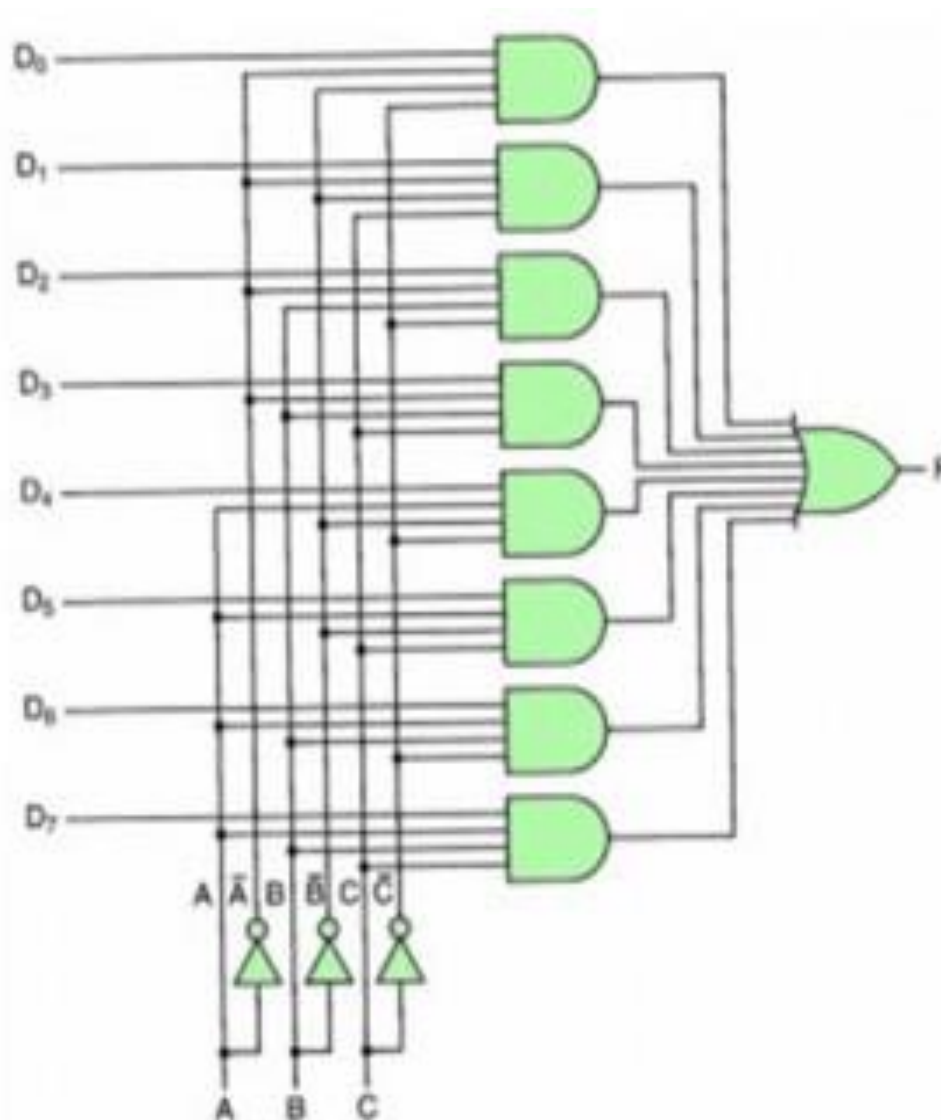


If the control input is changed to 11, then all gates are restricted except the bottom AND gate. In this case, D3 is transmitted to the output, and $q = D3$. If the control input is changed to AB =10, all gates are disabled except the top AND gate. In this case, D0 is transmitted to the output, and $q = D0$. The best example of a 4X1 multiplexer is IC 74153. In this IC, the o/p is the same as the i/p. Another example of a **4X1 multiplexer is IC 45352**. In this IC, the o/p is the complement of the i/p

8-1 Multiplexer Circuit

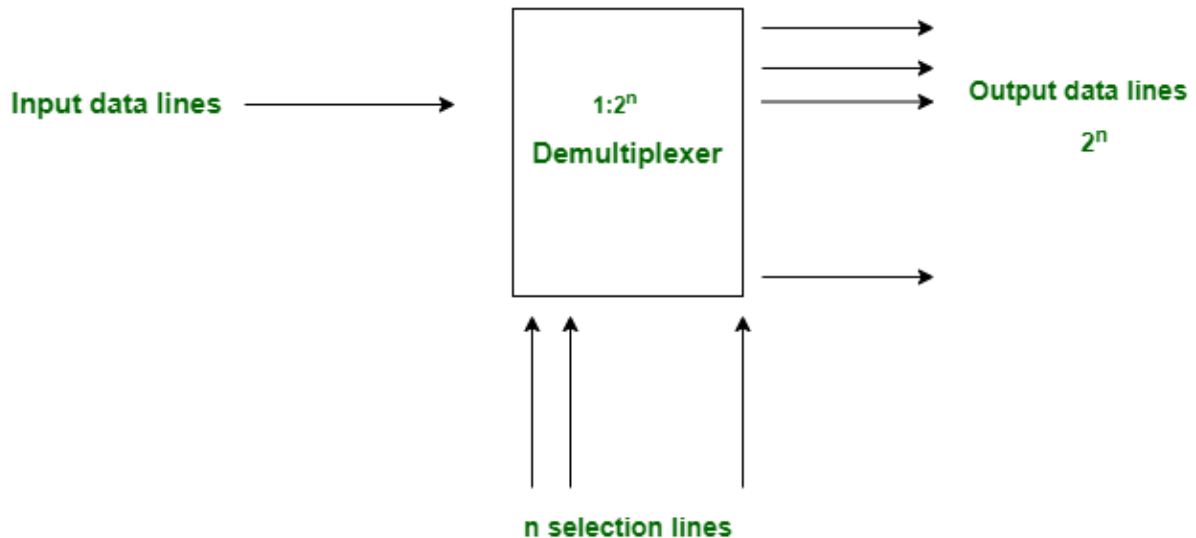
For the combination of a selection input, the data line is connected to the output line. The circuit shown below is an 8*1 multiplexer. The 8-to-1 multiplexer requires **8 AND gates, one OR gate, and 3 selection lines**. As an input, the combination of selection inputs is giving to the AND gate with the corresponding input data lines.

In a similar fashion, all the AND gates are given connection. In this 8*1 multiplexer, for any selection line input, one AND gate gives a value of 1 and the remaining all AND gates give 0. And, finally, by using OR gates, all the AND gates are added; and, this will be equal to the selected value.



Demultiplexer

Demultiplexer is a data distributor which takes a **single input and gives several outputs**. In demultiplexer we have **1 input and 2^n output lines** where n is the selection line.



The main difference between a multiplexer and a de-multiplexer is that a multiplexer takes two or more signals and encodes them on a wire, whereas a de-multiplexer does reverse to what the multiplexer does.

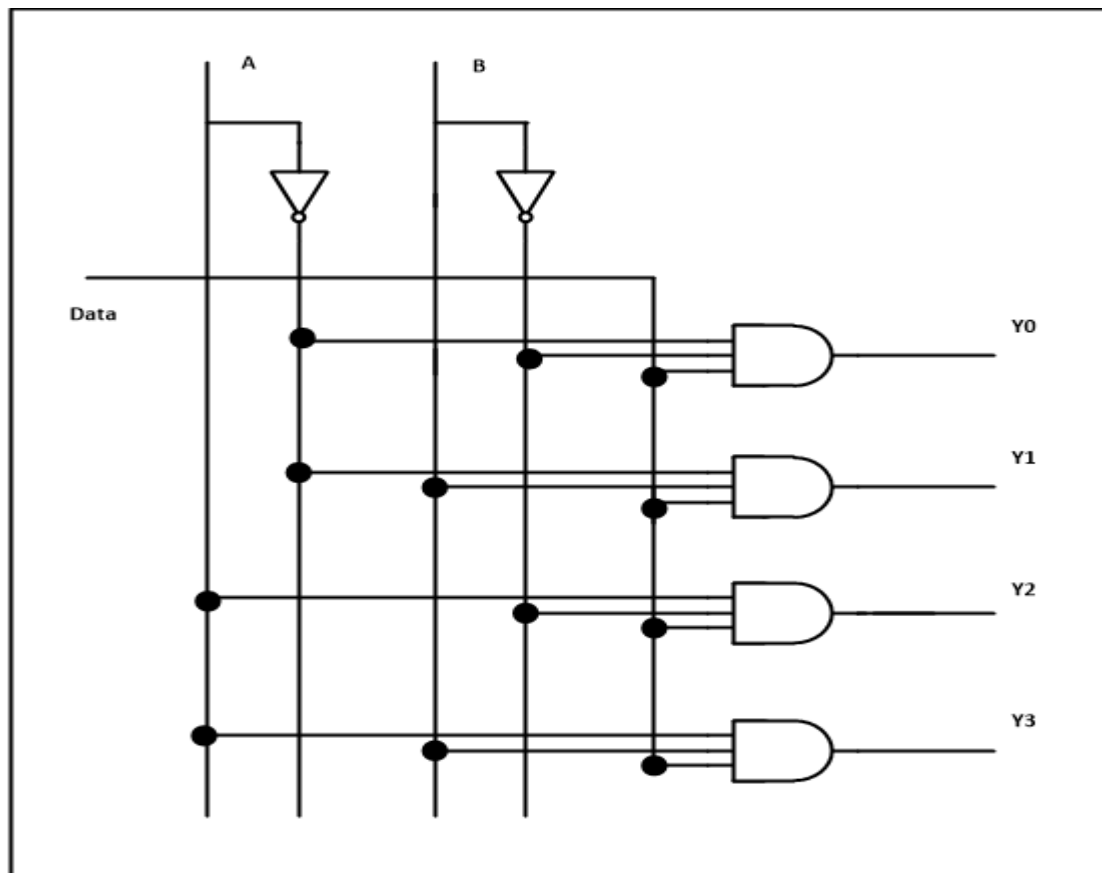
Types of Demultiplexer

Demultiplexers are classified into four types

- 1-2 demultiplexer (1 select line)
- 1-4 demultiplexer (2 select lines)
- 1-8 demultiplexer (3 select lines)
- 1-16 demultiplexer (4 select lines)

1-4 Demultiplexer

The 1-to-4 demultiplexer comprises **1- input bit, 4-output bits, and control bits**. The 1X4 demultiplexer circuit diagram is shown below.



The i/p bit is considered as Data **D**. This data bit is **transmitted** to the data bit of the **o/p lines**, which **depends on the AB** value and the control i/p.

When the control i/p $AB = 01$, the upper second AND gate is permitted while the remaining AND gates are restricted. Thus, only data bit **D** is transmitted to the output, and $Y1 = \text{Data}$.

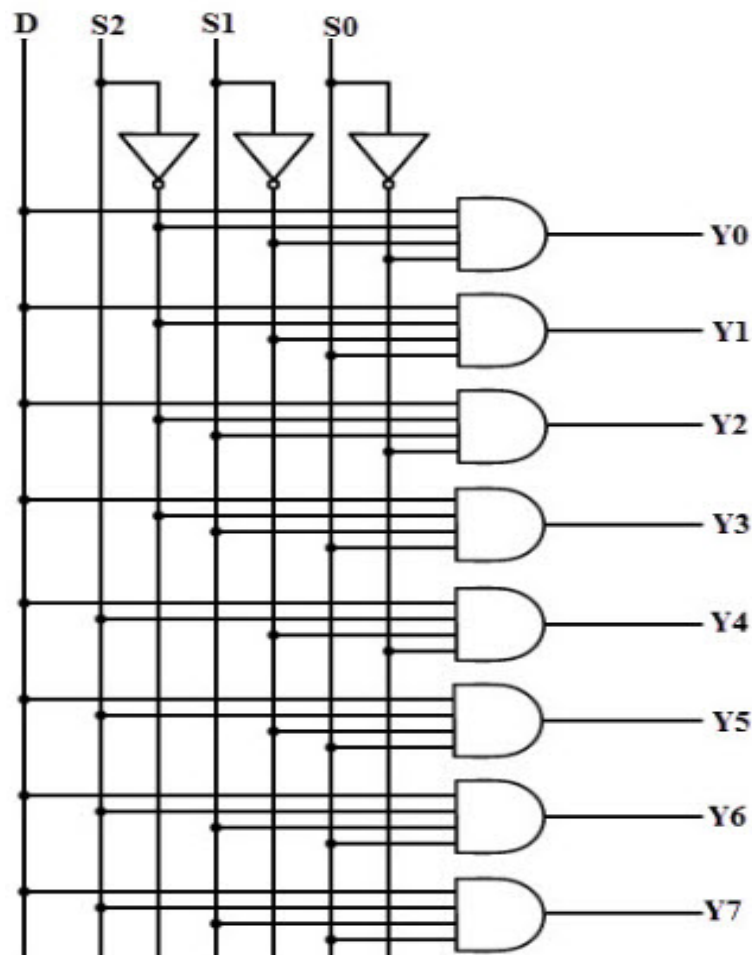
If the data bit **D** is low, the output **Y1** is low. IF data bit **D** is high, the output **Y1** is high. The value of the output **Y1** depends upon the value of data bit **D**, the remaining outputs are in a low state.

If the control input changes to $AB = 10$, then all the gates are restricted except the third AND gate from the top. Then, data bit **D** is transmitted only to the output **Y2**; and, $Y2 = \text{Data}$. . The best example of 1X4 demultiplexer is IC 74155.

8-1 Multiplexer Circuit

For the combination of a selection input, the data line is connected to the output line. The circuit shown below is an 8*1 multiplexer. The 8-to-1 multiplexer **requires 8 AND gates, one OR gate, and 3 selection lines**. As an input, the combination of selection inputs is giving to the AND gate with the corresponding input data lines.

In a similar fashion, all the AND gates are given connection. In this 8*1 multiplexer, for any selection line input, one AND gate gives a value of 1 and the remaining all AND gates give 0. And, finally, by using OR gates, all the AND gates are added; and, this will be equal to the selected value.



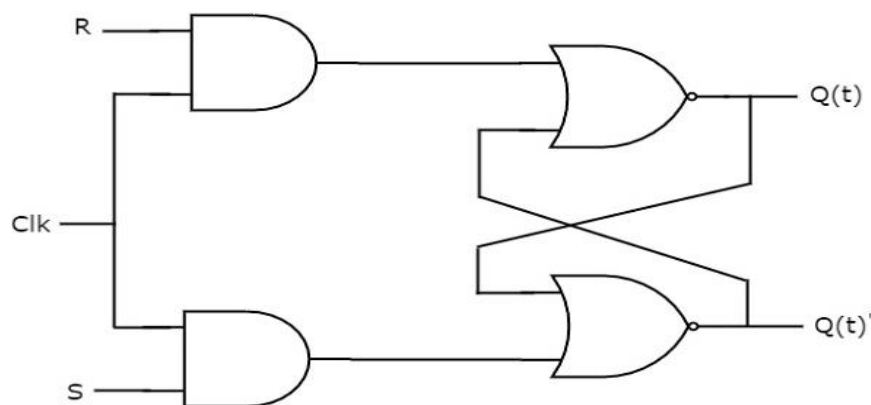
Flip-flop

A flip flop is an electronic circuit with **two stable states** that can be used to **store binary data**. The **stored data** can be **changed** by applying **varying inputs**. Flip-flops are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Flip-flops are used as data storage elements. It is the basic storage element in sequential logic.

- SR Flip-Flop
- D Flip-Flop
- JK Flip-Flop
- T Flip-Flop

SR Flip-Flop

This simple flip flop circuit has a set **input (S)** and a reset **input (R)**. In this circuit when you Set “S” as active the output $Q(t)$ would be high and $Q(t)'$ will be low. Once the outputs are established, the wiring of the circuit is maintained until “S” or “R” go high, or power is turned off. The two **outputs are the inverse of each other**.



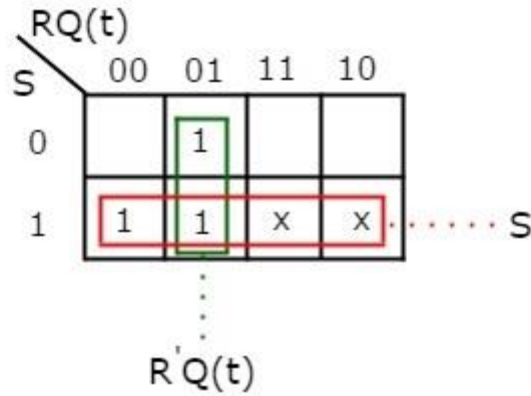
The following table shows the **State Table** of SR flip-flop.

S	R	Q	Q'
0	0	0	1
0	1	0	1
1	0	1	0
1	1	∞	∞

Here, **Q(t) & Q(t+1)** are **present state & next state respectively**. So, SR flip-flop can be used for one of these **three functions** such as **Hold, Reset & Set** based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of SR flip-flop.

Present Inputs		Present State	Next State
S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

By using three variable K-Map, we can get the **simplified expression for next state, Q(t+1)**. The three variable K-Map for next state, Q(t+1) is shown in the following figure.

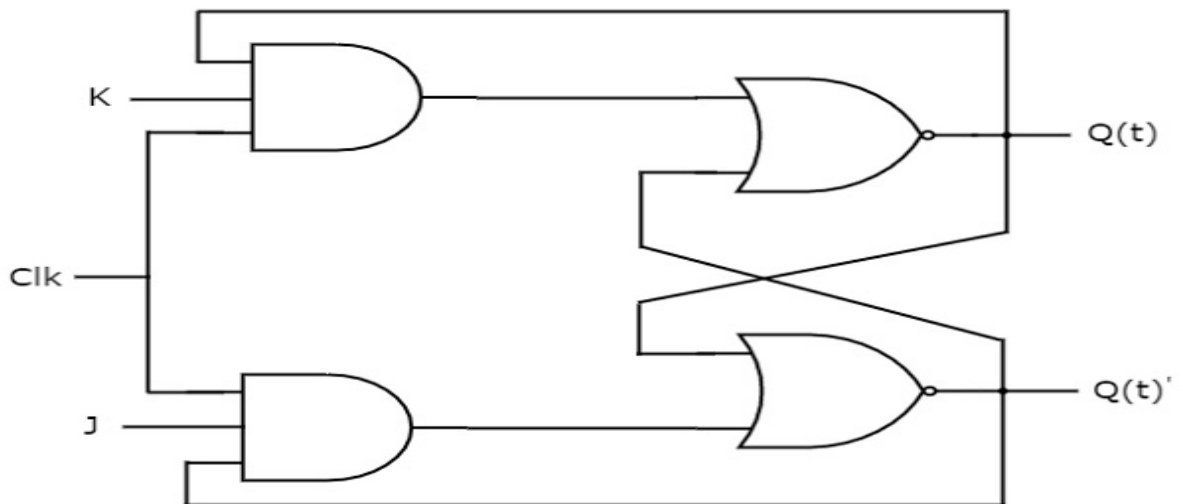


The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the simplified expression for next state $Q(t+1)$ is

$$Q(t+1) = S + R'Q(t)$$

JK Flip-flop

Due to the **undefined state in the SR flip flop**, another flip flop is required in electronics. The JK flip flop is an improvement on the SR flip flop where **S=R=1** is not a problem.



This circuit has two inputs J & K and two outputs $Q(t)$ & $Q(t)'$. The operation of JK flip-flop is similar to SR flip-flop. Here, we considered the inputs of SR flip-flop as **S = JQ(t)'** and **R = KQ(t)** in order to utilize the modified SR flip-flop for 4 combinations of inputs.

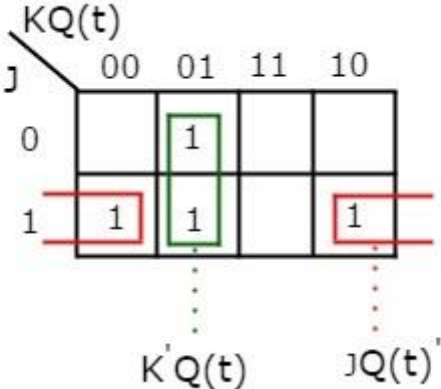
The following table shows the **State Table** of JK flip-flop.

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q(t)'

Here, Q(t) & Q(t+1) are present state & next state respectively. So, JK flip-flop can be used for one of these four functions such as **Hold, Reset, Set & Complement of present state based on the input conditions**, when positive transition of clock signal is applied. The following table shows the characteristic table of JK flip-flop.

Present Inputs		Present State	Next State
J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

By using three variable K-Map, we can get the **simplified expression** for next state, $Q(t+1)$. Three variable K-Map for next state, $Q(t+1)$ is shown in the following figure.

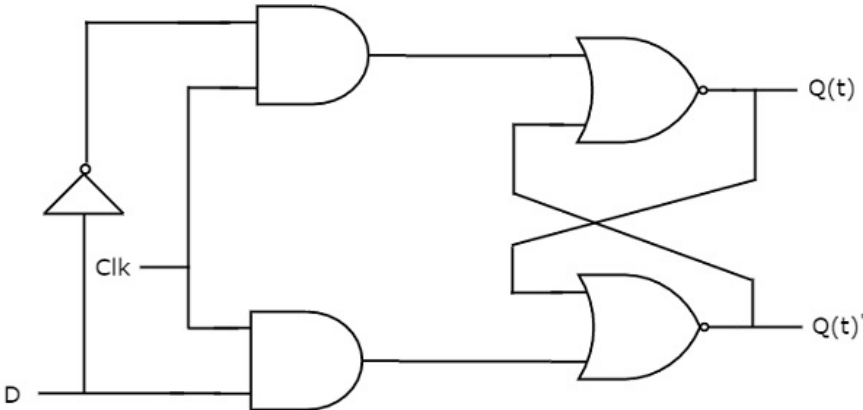


The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the simplified expression for next state $Q(t+1)$ is

$$Q(t+1) = JQ(t)' + K'Q(t)$$

D Flip-Flop

D flip-flop operates with only positive clock transitions or negative clock transitions. That means, the output of **D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal**. The circuit diagram of D flip-flop is shown in the following figure.



This circuit has single input D and two outputs $Q(t)$ & $Q(t)'$. The outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the state table of D flip-flop.

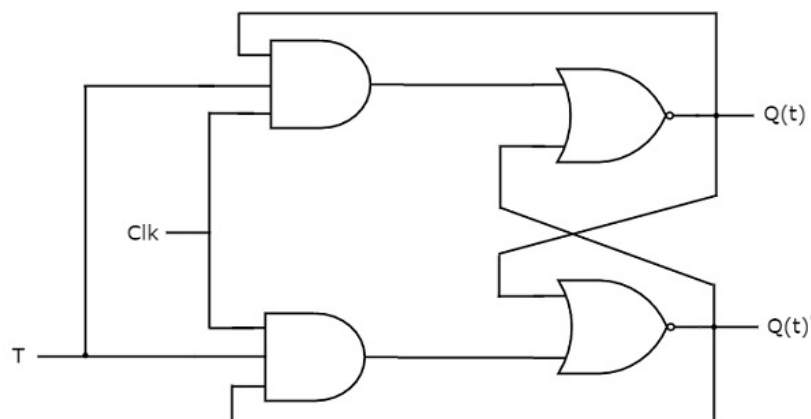
D	Q(t + 1)
0	0
1	1

Therefore, D flip-flop always **hold** the information, which is available on **data input**, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation as

$$Q(t+1) = D$$

T Flip-Flop

T flip-flop is the **simplified version of JK flip-flop**. It is obtained by connecting the same input 'T' to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions. The circuit diagram of T flip-flop is shown in the following figure.



This circuit has single input T and two outputs Q(t) & Q(t)'. The operation of T flip-flop is same as that of JK flip-flop. Here, we considered the inputs of JK flip-flop as **J = T and K = T** in order to utilize the modified JK flip-flop for 2 combinations of inputs. So, we eliminated the other two combinations of J & K, for which those two values are complement to each other in T flip-flop.

The following table shows the state table of T flip-flop.

T	Q(t+1)
0	Q(t)
1	Q(t)'

Here, Q(t) & Q(t+1) are present state & next state respectively. So, **T flip-flop can be used for one of these two functions such as Hold, & Complement of present state based on the input conditions**, when positive transition of clock signal is applied. The following table shows the characteristic table of T flip-flop.

Inputs	Present State	Next State
T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

From the above characteristic table, we can directly write the next state equation as

$$Q(t+1) = T'Q(t) + TQ(t)'$$

$$\Rightarrow Q(t+1) = T \oplus Q(t)$$

Shift Registers

One flip-flop can **store one-bit** of information. In order to **store multiple bits** of information, we require **multiple flip-flops**. So the group of flip-flops group of **flip flops connected in series used to store multiple bits of data, is known as register**.

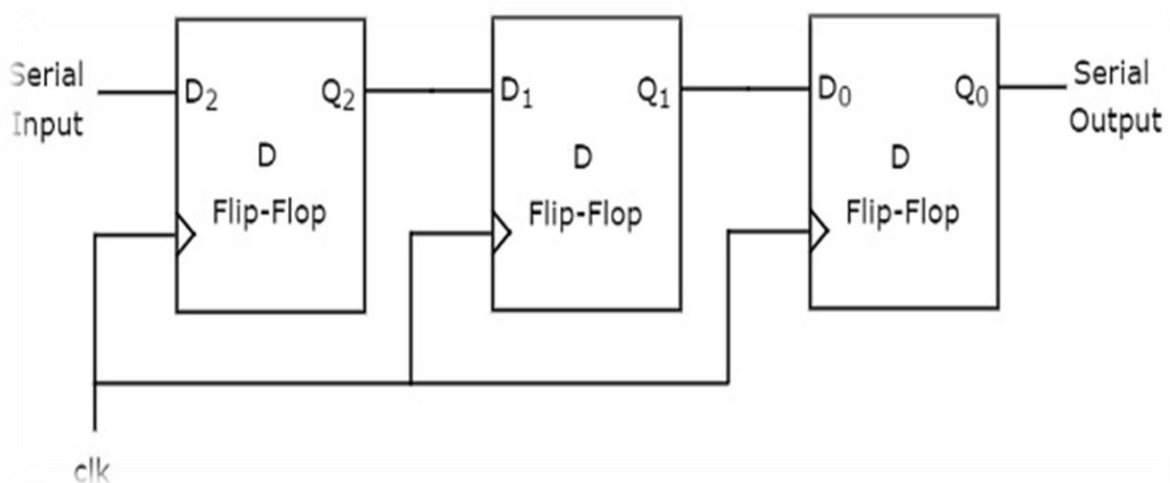
The **bits** stored in such registers can be made to **move** within the registers and **in/out** of the registers by applying **clock pulses**. An 'N' bit shift register contains 'N' flip-flops. The registers which will shift the bits to left are called **Shift left registers**. The registers which will shift the bits to right are called **Shift right registers**.

Four types of shift registers based on applying inputs and accessing of outputs.

- Serial In – Serial Out shift register
- Serial In – Parallel Out shift register
- Parallel In – Serial Out shift register
- Parallel In – Parallel Out shift register

Serial-In Serial-Out Shift Register (SISO)

This block diagram consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the **same clock signal is applied** to each one.



In this shift register, we can send the bits serially from the **input of left most D flip-flop**. Hence, this input is also called as **serial input**.

For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the **output of right most D flip-flop**. Hence, this output is also called as **serial output**.

Example

The working of 3-bit SISO shift register by sending the binary information “**011**” from LSB to MSB serially at the input.

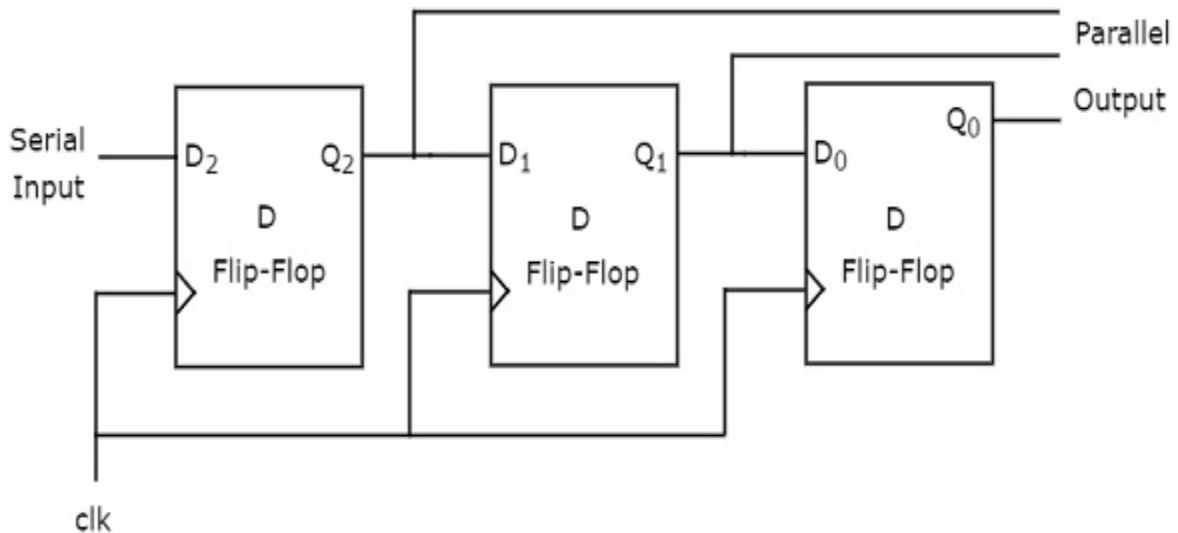
Let initial status of the D flip-flops from left to rightmost is $Q_2Q_1Q_0 = 000$. We can understand the working of 3-bit SISO shift register from the following table.

No of positive edge of Clock	Serial Input	Q_2	Q_1	Q_0
0	-	0	0	0
1	1 at LSB	1	0	0
2	1	1	1	0
3	0 at MSB	0	1	1 at LSB
4	-	-	0	1
5	-	-	-	0 at MSB

Therefore, the **N-bit SISO shift register requires $2N-1$ clock pulses** in order to shift ‘N’ bit information.

Serial-In Parallel-Out Shift Register (SIPO)

The shift register, which allows **serial input** (one bit after the other through a single data line) and produces a **parallel output** is known as Serial-In Parallel-Out shift register.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these **flip-flops are synchronous** with each other since, the same clock signal is applied to each one.

Example

The working of 3-bit SIPO shift register by sending the binary information “**011**” from LSB to MSB serially at the input.

Let initial status of the D flip-flops from left to rightmost is $Q_2Q_1Q_0 = 000$. Here, Q_2 & Q_0 are **MSB & LSB** respectively.

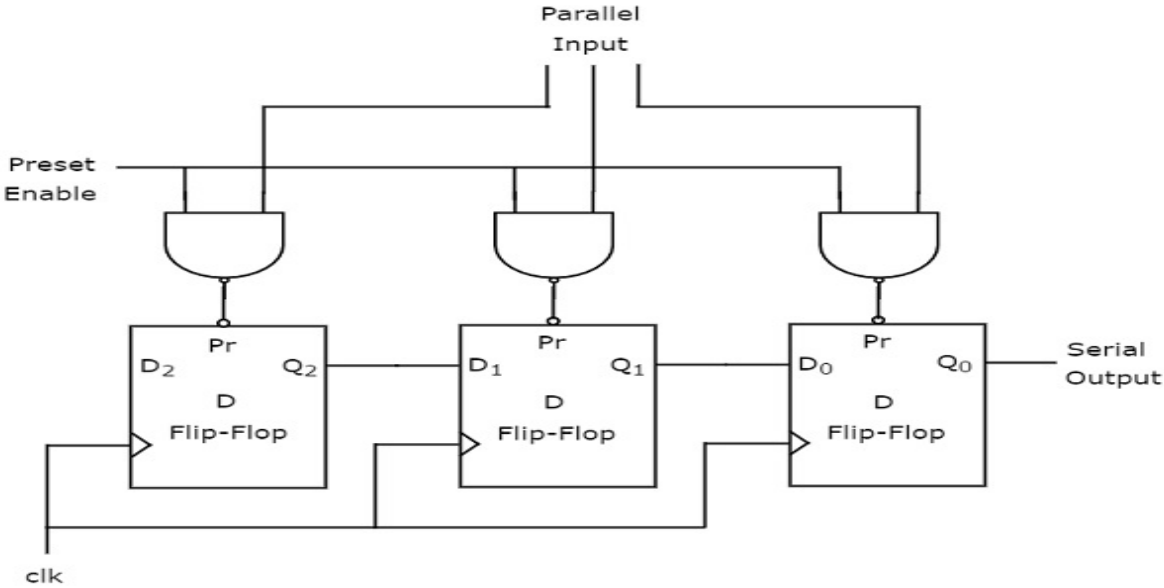
The N-bit SIPO shift register requires **N clock pulses in order to shift ‘N’ bit** information.

We can understand the working of 3-bit SIPO shift register from the following table.

No of positive edge of Clock	Serial Input	Q ₂ MSB	Q ₁	Q ₀ LSB
0	-	0	0	0
1	1 at LSB	1	0	0
2	1	1	1	0
3	0 at MSB	0	1	1

Parallel-In Serial-Out Shift Register (PISO)

The shift register, which allows **parallel input** (data is given separately to each flip flop and in a simultaneous manner) and produces a **serial output** is known as Parallel-In Serial-Out shift register. The block diagram of 3-bit PISO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the parallel inputs to each D flip-flop by making Pre-set Enable to 1. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the serial output from the right most D flip-flop.

Example

Let us see the working of 3-bit PISO shift register by applying the binary information “011” in parallel through pre-set inputs.

Since the pre-set inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be $Q_2Q_1Q_0 = 011$. We can understand the working of 3-bit PISO shift register from the following table.

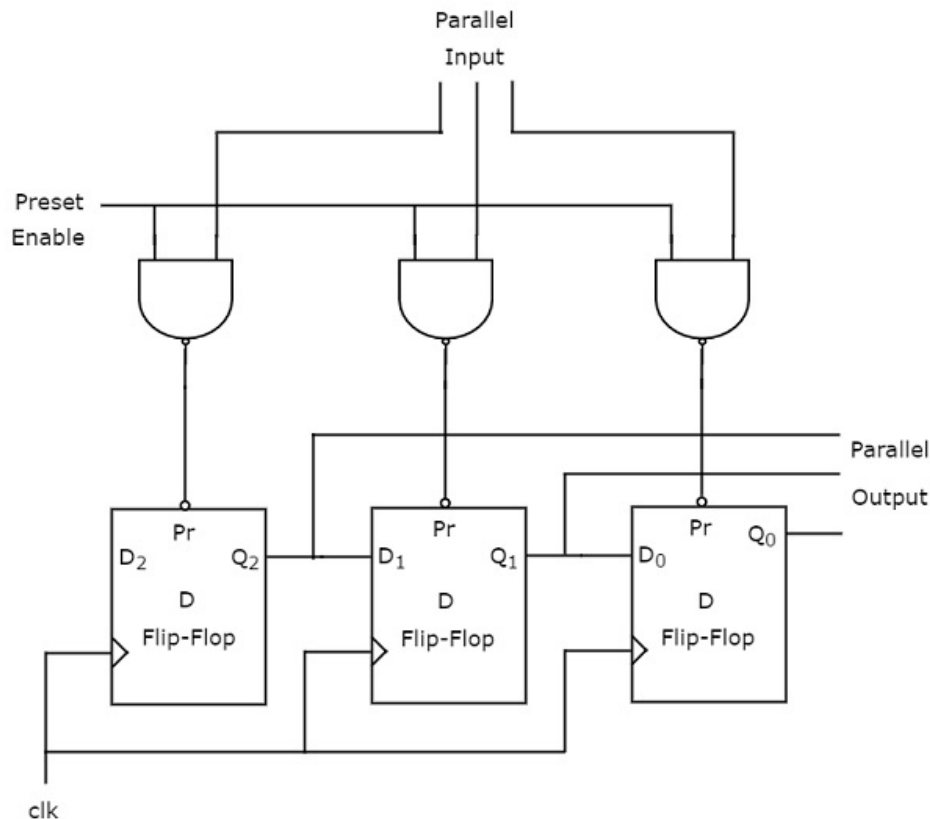
No of positive edge of Clock	Q_2	Q_1	Q_0
0	0	1	1 at LSB
1	-	0	1
2	-	-	0 at MSB

Thus N-bit PISO shift register requires **N-1 clock pulses in order to shift ‘N’ bit** information.

Parallel-In Parallel-Out Shift Register (PIPO)

The shift register, which allows **parallel input** (data is given separately to each flip flop and in a simultaneous manner) and also produces a **parallel output** is known as Parallel-In parallel-Out shift register.

The block diagram of 3-bit PIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the parallel inputs to each D flip-flop by making Pre-set Enable to 1. We can apply the parallel inputs through pre-set or clear. These two are asynchronous inputs. That means, **the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition.** So, we will get the parallel outputs from each D flip-flop.

Example

Let us see the working of 3-bit PIPO shift register by applying the binary information “011” in parallel through pre-set inputs.

Since the pre-set inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be $Q_2Q_1Q_0 = 011$. So, the binary information “011” is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.

Thus the N-bit PIPO shift register **doesn't require any clock pulse** in order to shift 'N' bit information.

ASSIGNMENT:3

- *Find difference between MUX and DE-MUX*
- *Applications of Flip-Flops and Shift Registers*

Counters

Counter is a device which stores and displays the **number of times a particular event** or process has occurred. It is a group of **flip-flops** with a clock signal applied or we can say counters are used in digital electronics for **counting** purpose, they can count specific event happening in the circuit. An **'N' bit binary counter consists of 'N' flip-flops**. If the counter counts from **0 to $2^N - 1$** , then it is called as binary **up counter**. Similarly, if the counter counts down from **$2^N - 1$ to 0**, then it is called as binary **down counter**.

Counters are of two types.

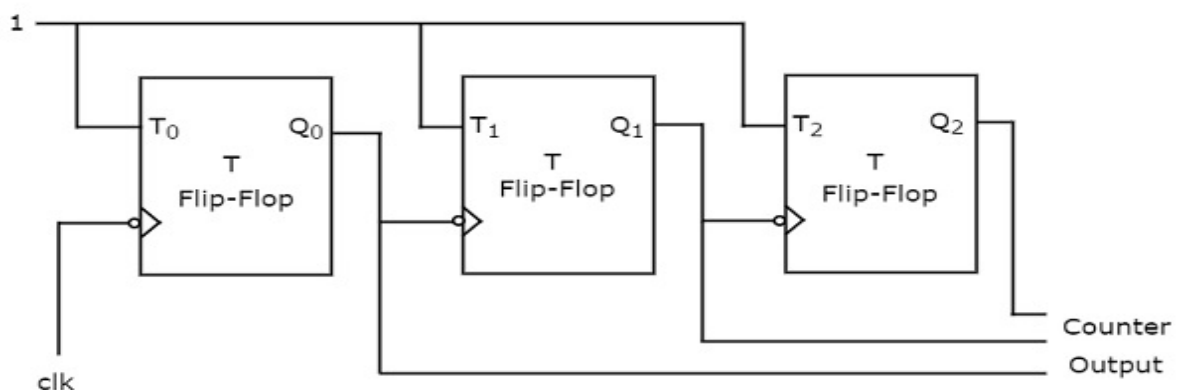
- Asynchronous or ripple counters.
- Synchronous counters.

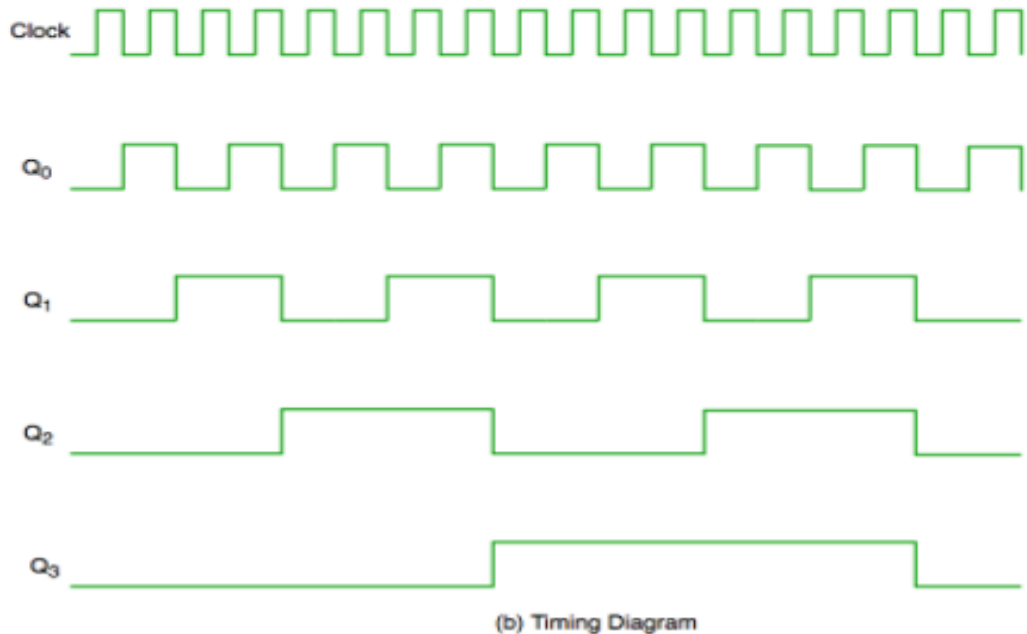
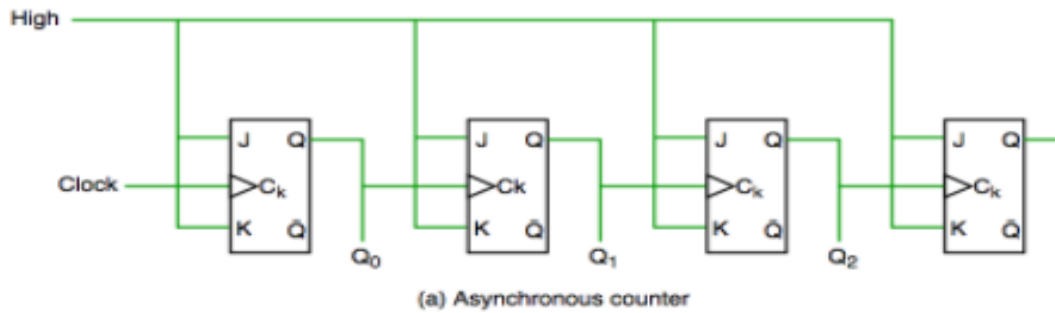
Asynchronous Counter

Asynchronous stands for the absence of synchronization or Asynchronous stands for controlling the operation timing by sending a pulse **only when the previous operation is completed** rather than sending it in regular intervals.

In asynchronous counter **we don't use universal clock**, only **first flip flop is driven by main clock** and the clock input of rest of the following flip flop is driven by output of previous flip flops. Flip-flops are **serially connected** together, and the clock pulse ripples through the counter. Due to the ripple clock pulse, it's often called a **RIPPLE counter**.

An Asynchronous counter can **count $2^n - 1$** possible counting state. The T flip-flop or JK flip-flop are being used to design asynchronous counter. We can understand it by following diagram

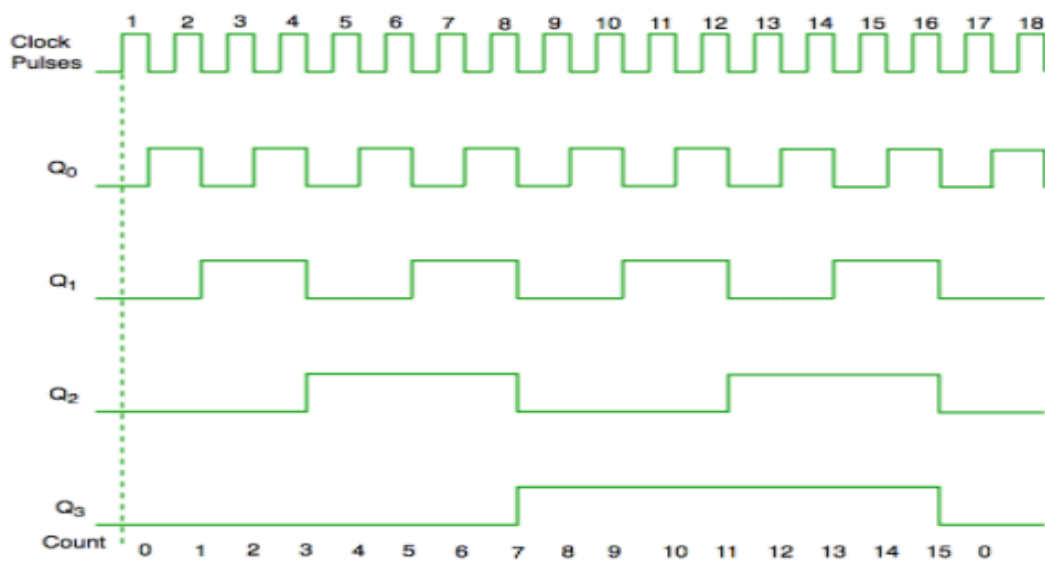
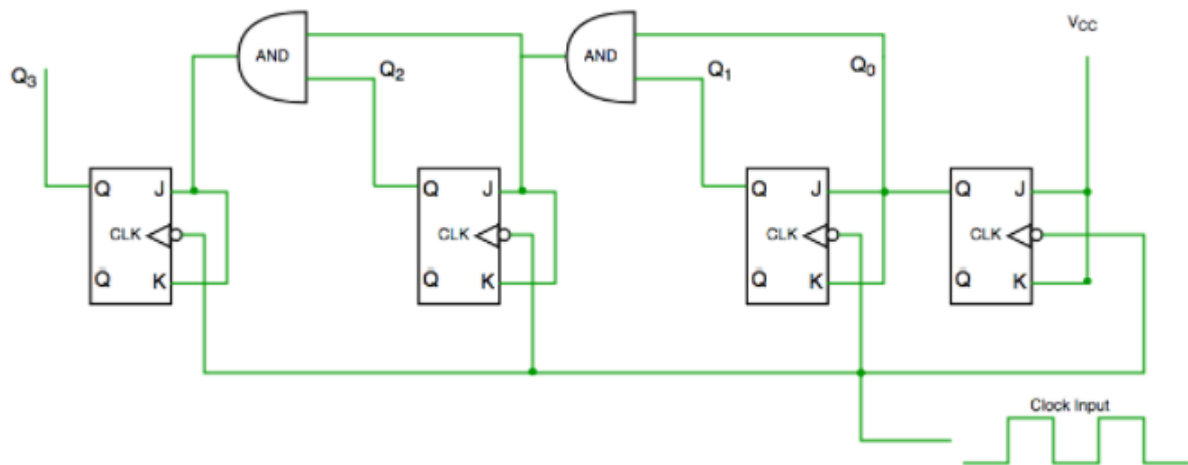




It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered (because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called RIPPLE counter.

Synchronous Counter

Unlike the asynchronous counter, synchronous counter has **one global clock** which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on **higher frequency** than asynchronous counter as it does **not have cumulative delay** because of **same clock** is given to each flip flop. We can understand it by following diagram



From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0, Q3 is dependent on Q2, Q1 and Q0.

Microprocessor and Microcontroller

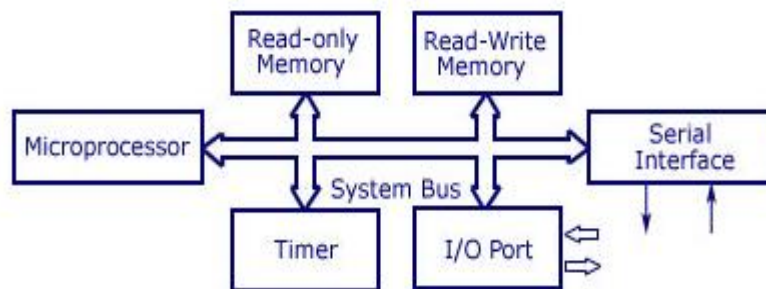
At the basic level, a microprocessor and micro controller exist for performing some operations they are fetching instructions from the memory and executing this instruction (arithmetic or logic operations) and the result of these executions are used to serve to output devices.

Microprocessors in most cases will begin with Intel 8085 and **Microcontrollers** with Intel 8051 from the MCS 51 micro controller family,

Microprocessor

A microprocessor is a controlling unit of a micro-computer wrapped inside a small chip. It performs Arithmetic Logical Unit (ALU) operations and communicates with the other devices connected with it. It is a single Integrated Circuit in which several functions are combined.

Schematic Arrangement of a Microprocessor Based System



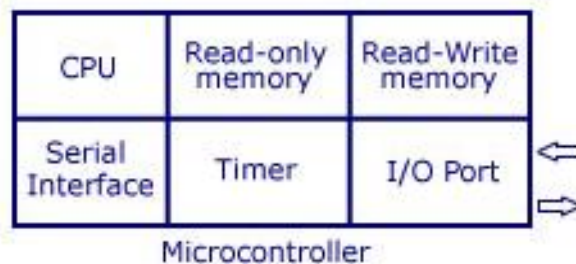
Microprocessor that has many support devices like Read-only memory, Read-Write memory, Serial interface, Timer, Input/Output ports, etc. All these support devices are interfaced to the microprocessor via a system bus. So one point is clear now, all support devices in a microprocessor-based system are external. The system bus is composed of an address bus, data bus, and control bus.

Microcontroller

A microcontroller is a chip optimized to control electronic devices. It is stored in a single integrated circuit which is dedicated to performing a particular task and executes one specific application.

It is specially designed circuits for embedded applications and is widely used in automatically controlled electronic devices. It contains memory, processor, and programmable I/O.

Schematic Internal Architecture
of a Microcontroller

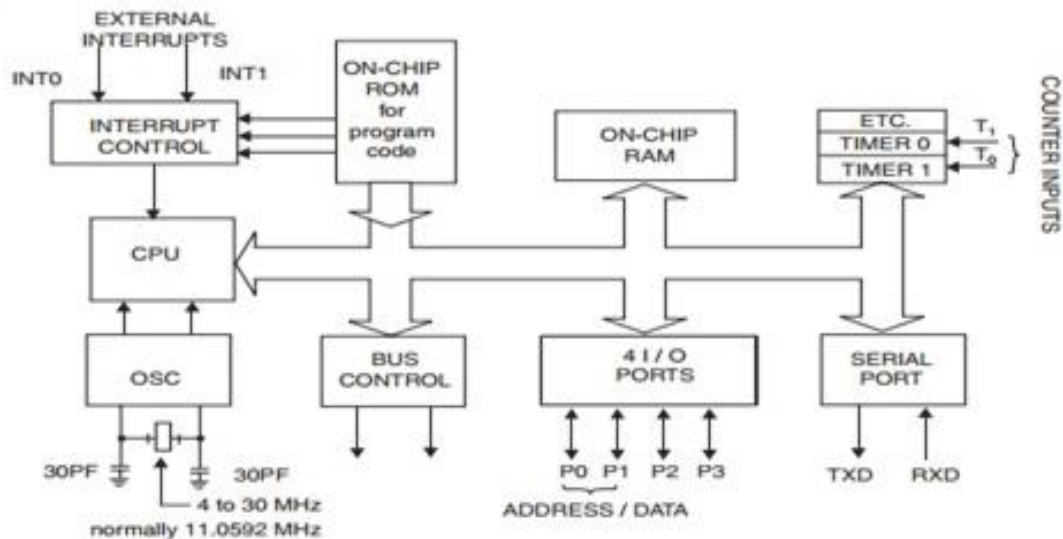


All the support devices like Read-only memory, Read-Write memory, Timer, Serial interface, I/O ports are internal. There is no need for interfacing these support devices and this saves a lot of time for the individual who creates the system

Difference between Microcontroller vs Microprocessor

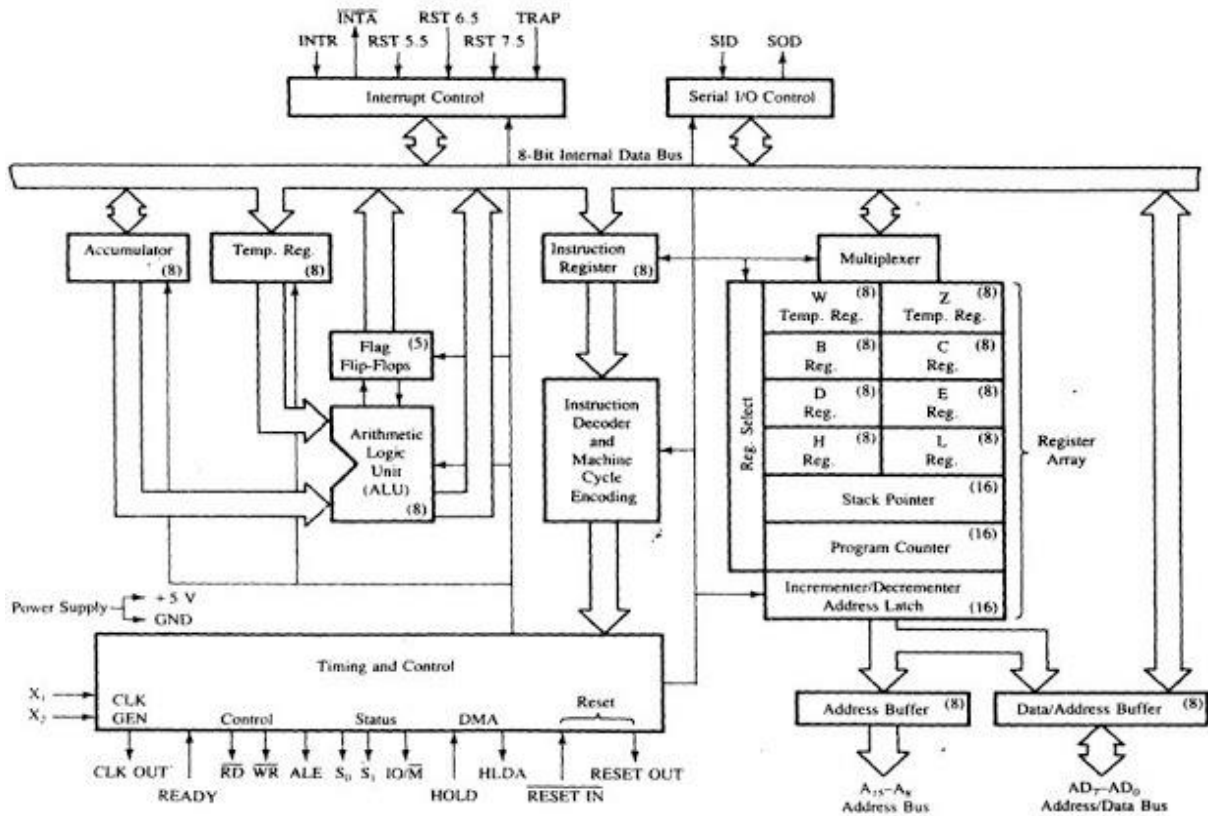
8051 Microcontroller

- It houses 8 bit CPU and 4 KByte ROM used for code or program storage
- 128 Byte RAM to store data or variables used in program
- 32 Input/Output lines with 4 ports (8 lines per port)
- 2 Timers used for putting delay & setting baud rate for data communication.
- 1 Serial Port with one TxD and RxD line used for serial communication with external devices.
- 6 Interrupt Sources and Clock oscillator circuit runs at 12MHz frequency.



8085 Microprocessor

- It is 8 bit size processor developed as single chip using N-MOS.
- It has multiplexed address and data bus on 8 lines AD0 to AD7.
- The maximum clock frequency used in 8085 is 3 MHz.
- It has 40 pins and runs at 5V power supply.
- There are 5 hardware interrupts viz. TRAP, INTR, RST5.5, RST6.5, RST7.5
- It has about 74 programming instructions with 4 addressing modes.
- It does not house memory, but it has 16 address lines which can access 64K bytes (2^{16}) of externally connected memory.
- It has 8 bit lines which can address 256 ($\sim 2^8$) ports connected externally.
- It has two serial lines viz. SID and SOD. These can be connected with serial peripherals.
- It consists of ACC, one flag register, 6 general purpose registers and two special registers (SP-Stack Pointer, PC-Program Counter).



ASSIGNMENT:4

- Find applications of Microcontroller and Microprocessor
- What is Asynchronous up and down counter, Explain?

Designed by: Vipra Bohara

Assistant Professor

JECRC, Sitapura, Jaipur.