



LMN – Digital Electronics

Last Updated : 29 Jan, 2025

Digital electronics deals with systems that use digital signals, represented as 0s and 1s, to process information. It is the backbone of modern devices like computers, smartphones, and calculators. Unlike analog electronics, which works with continuous signals, digital electronics offers higher accuracy, reliability, and noise resistance, making it essential in today's technology-driven world.

Logic Gates

Logic gates are the basic building blocks of digital circuits. They perform logical operations on one or more binary inputs to produce a single binary output. The output depends on the type of logic gate and the combination of inputs.

Types of Logic Gates

Given Below are the different types of Logic Gates :

- AND Gate(.)
- OR Gate(+)
- NOT Gate(')
- XOR Gate
- NAND Gate
- NOR Gate
- XNOR Gate
- Buffer Gate
- Universal Logic Gates

1. AND Gate(.): The AND gate gives an output of 1 when if both the two inputs are 1, it gives 0 otherwise. For n-input gate if all the inputs are 1 then 1 otherwise 0.The

AND gate operation is similar to the standard multiplication of 1s and 0s. The (.) dot represents the AND operation.

2. OR Gate(+): The OR gate gives an output of 1 if either of the two inputs are 1, it gives 0 otherwise. For n-input gate if all the inputs are 0 then 0 otherwise 1. The OR Operation is represented by the +.

3. NOT Gate('): The NOT gate gives an output of 1 if the input is 0 and vice-versa. It is also known as Inverters. In Boolean algebra NOT operation is represented by bar over the variable such as A^{\sim} .

4. XOR Gate: The XOR gate gives an output of 1 if either both inputs are different, it gives 0 if they are same. For n-input gate if the number of input 1 are odd then it gives 1 otherwise 0. For a two-input XOR gate it means that the output is true only if exactly one of the inputs is true.









5. NAND Gate: The NAND gate (negated AND) gives an output of 0 if both inputs are 1, it gives 1 otherwise. For n-input gate if all inputs are 1 then it gives 0 otherwise 1. The Term "NAND" can be said as "Not AND".

6. NOR Gate: The NOR gate (negated OR) gives an output of 1 only if both inputs are 0, it gives 0 otherwise. For n-input gate if all inputs are 0 then it gives 1 otherwise 0. The "NOR" can be said as "NOT OR".

7. XNOR Gate: The XNOR gate (negated XOR) gives an output of 1 if both inputs are same and 0 if they are different. For n-input gate if the number of input 1 are even then it gives 1 otherwise 0 or if the number of input 0 is even then the output is 1, otherwise 0'. The "XNOR" can be said as "Exclusive NOR".

8. Buffer Gate: A Buffer Gate is a digital logic gate that amplifies a signal. It doesn't change the logic state (it outputs exactly what is input), but its primary purpose is to ensure that the signal can drive more circuits without weakening. Essentially, a buffer isolates the input from the output, making the output more robust.

9. Universal Logic Gates: Out of the eight logic gates discussed above, NAND and NOR are also known as **universal gates** since they can be used to implement any digital circuit without using any other gate. This means that every gate can be created by NAND or NOR gates only.

Name	AND	OR	Inverter	Buffer	NAND	NOR	Exclusive-OR (XOR)	Exclusive-NOR or equivalence																																																																																																						
Graphic Symbol																																																																																																														
Algebraic Function	$F = x \cdot y$	$F = x + y$	$F = x^1$	$F = x$	$F = (xy)^1$	$F = (x+y)^1$	$F = xy^1 + x^1y = x \oplus y$	$F = xy + x^1y^1 = (x \oplus y)^1$																																																																																																						
Truth Table	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																																																																																																												
0	0	0																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	1																																																																																																												
x	y	F																																																																																																												
0	0	0																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	1																																																																																																												
x	F																																																																																																													
0	1																																																																																																													
1	0																																																																																																													
x	F																																																																																																													
0	0																																																																																																													
1	1																																																																																																													
x	y	F																																																																																																												
0	0	1																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	0																																																																																																												
x	y	F																																																																																																												
0	0	1																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	0																																																																																																												
x	y	F																																																																																																												
0	0	0																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	0																																																																																																												
x	y	F																																																																																																												
0	0	1																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	1																																																																																																												

read more about – [Logic Gates](#)

Minimization of Boolean Function

Minimization of a Boolean function involves reducing the number of terms, literals, and gates required to represent a Boolean expression. The goal is to create a simpler and more efficient digital circuit without altering its functionality.

Techniques for Minimizing Boolean Functions

1. Algebraic Simplification:

$$F(A, B) = AB + A \text{ simplifies to } F(A, B) = A$$

2. Karnaugh Map (K-Map) Method:

- A visual method for simplifying Boolean expressions with 2, 3, 4, or more variables.
- Steps:
 1. Construct a K-Map with cells representing truth table outputs.
 2. Group adjacent 1s into power-of-two groups (1, 2, 4, 8, etc.).
 3. Derive a simplified expression by writing terms for each group.

Representation of Boolean Functions

Any Boolean expression can be expressed in two forms:

- Sum of Product form (SOP)
- Product of Sum form (POS)

1. SOP Form

The SOP expression usually takes the forms of two or more variables OR together.

$$Y = ABC + AB + AC$$

$$Y = AB + BC$$

SOP forms are used to write logical expression for the output becoming logic '1'.

Below is the example:

Input (3-variables)			Output (Y)
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Therefore, Notation of SOP expression is:

$$f(A, B, C) = \sum m(3, 5, 6, 7)$$

$$Y = m3 + m5 + m6 + m7$$

$$\text{Also, } Y = ABC + ABC + ABC + ABC$$

2. POS Form

The POS expression usually takes the form of two or more OR variables within parentheses, ANDed with two or more such terms.

$$\text{Example: } Y = (A + B + C)(BC + D)$$

Each individual term in standard POS form is called maxterm.

POS forms are used to write logical expression for output becoming logic '0'.

$$\text{we get } f(A, B, C) = \pi M(0, 1, 2, 4)$$

$$Y = M_0, M_1, M_2, M_4$$

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

read more about – [K-Map](#)

Don't Care Condition

- Some outputs of a Boolean function are irrelevant or unused; these are treated as “don't care” conditions (denoted as X).
- Can be grouped as 1s or 0s in K-Maps or Quine-McCluskey for better simplification.

Example: Minimize the following function in SOP minimal form using K-Maps

$$f = m(1, 5, 6, 11, 12, 13, 14) + d(4)$$

The SOP K-map for the given expression is:

CD \ AB	00	01	11	10
00		1		
01	X	1		1
11	1	1		1
10			1	

Therefore, SOP minimal is,

$$f = BC' + BD' + A'C'D + AB'CD$$

read more about – [Don't Care Condition](#)

Important Laws in Boolean Algebra

Here are the **De Morgan's Laws**, **Absorption Law**, **Domination Law**, and **Consensus Law** in Boolean Algebra:

1. De Morgan's Laws

De Morgan's Theorems describe how **AND** and **OR** operations interact with **NOT**:

- $(A \cdot B)' = A' + B'$
- $(A + B)' = A' \cdot B'$

2. Absorption Law

The **Absorption** rule simplifies expressions by eliminating redundant terms:

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

3. Domination Law (Zero and One Law)

This law defines the **dominant** elements in Boolean algebra:

- $A + 1 = 1$
- $A \cdot 0 = 0$

4. Consensus Law

This law eliminates redundant terms in Boolean expressions:

- $A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C$
- $(A + B) \cdot (A' + C) \cdot (B + C) = (A + B) \cdot (A' + C)$

Duality Theorem

The **Duality Theorem** states that every Boolean algebra expression remains valid if we:

1. Swap AND (\cdot) with OR ($+$).
2. Swap 0 with 1 (and vice versa).

For example:

- The dual of $A + (B \cdot C) = (A + B) \cdot (A + C)$ is $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$.

Combinational Circuits

Combinational circuits are a type of digital circuit in which the output depends only on the current inputs, not on past inputs or previous states. These circuits are built using logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR.

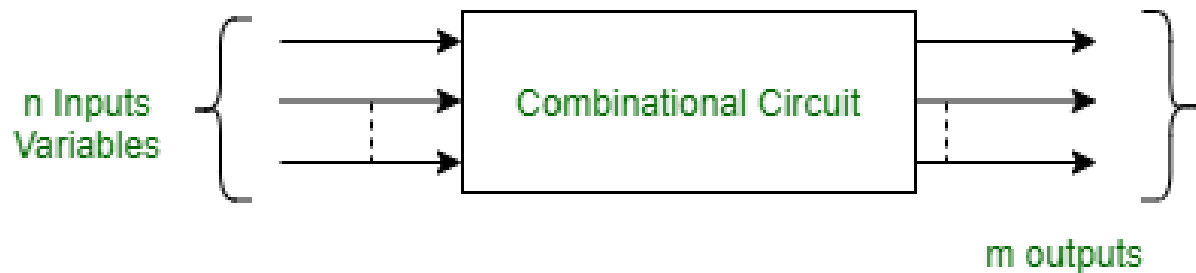


Figure - Block diagram of Combinational circuit

Combinational Circuit

Basic Examples of Combinational Circuits are:

1. Adders: The most basic arithmetic operation is the addition of two binary digits. A combinational circuit that performs the addition of two 1-bit numbers is called as [half adder](#), and the logic circuit that adds three 1-bit numbers is called as [full adder](#).

- **Half Adder:** Adds two single-bit binary numbers (A and B).

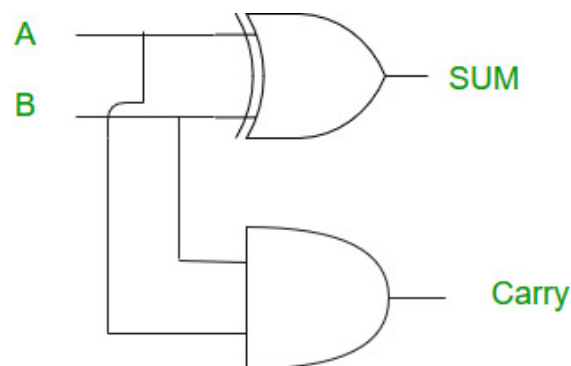
Inputs: A, B

Outputs: Sum (S), Carry (C).

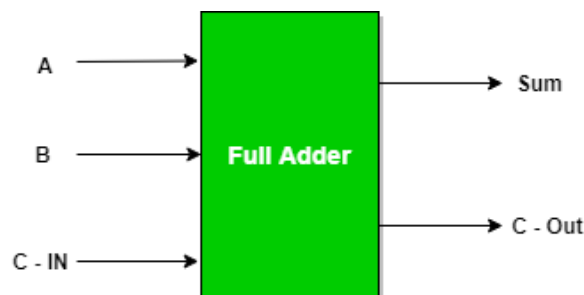
Logic:

$$S = A \oplus B$$

$$C = A \cdot B$$



- **Full Adder:** Adds three single-bit numbers (A, B, and Carry-in).



A Full Adder adds three inputs: two operands (A, B) and a carry-in (Cin) and produces a Sum (S) and Carry-out (Cout).

- Sum (S) = $A \oplus B \oplus C_{in}$
- Carry-out (Cout) = $(A \cdot B) + (B \cdot C_{in}) + (A \cdot C_{in})$

2. Subtractors:

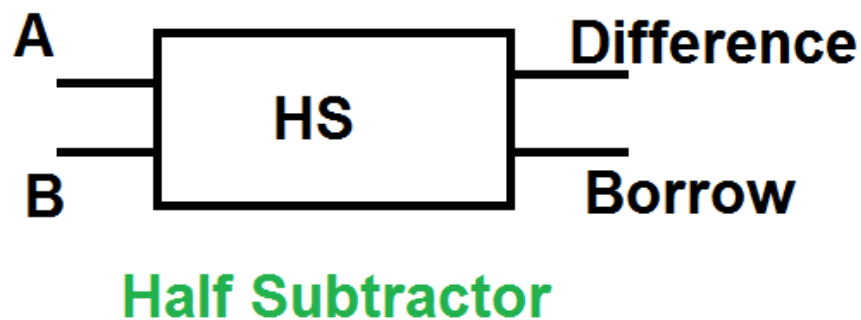
- **Half Subtractor:** A half subtractor is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. [Half Subtractor](#) subtracts one binary digit from another to produce a DIFFERENCE output and a BORROW output.

Outputs: Difference, Borrow.

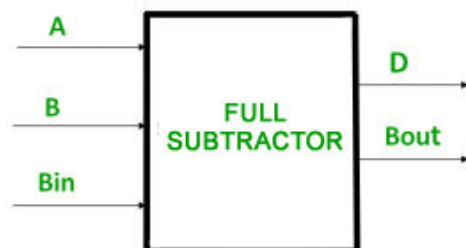
- A **Half Subtractor** subtracts two bits ($A - B$) and gives a **Difference (D)** and **Borrow (Bout)**.

$$\text{Difference (D)} = A \oplus B$$

$$\text{Borrow (Bout)} = A' \cdot B$$



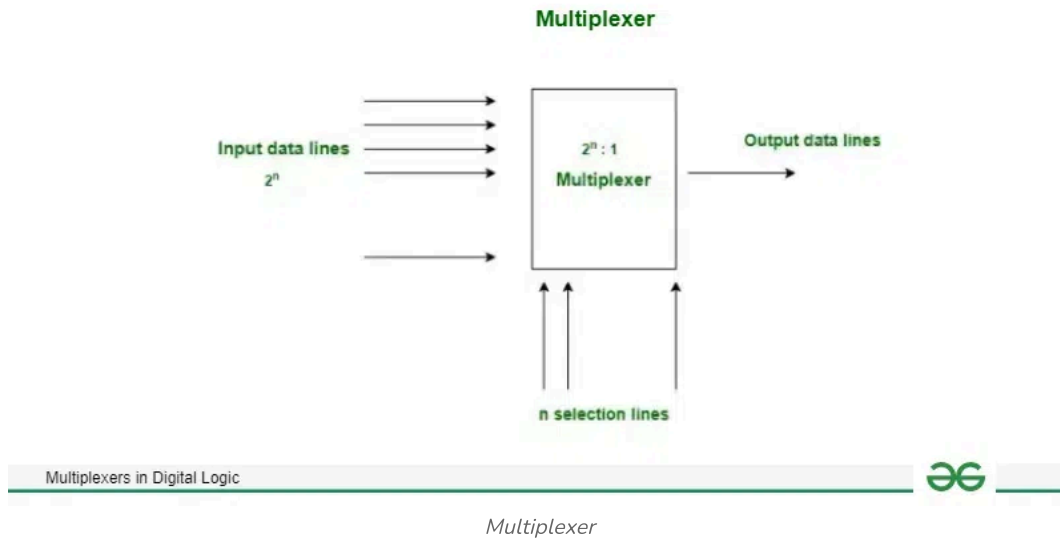
- **Full Subtractor:** Subtracts three single-bit numbers, including Borrow-in.



A Full Subtractor subtracts two operands (A, B) and a Borrow-in (Bin), producing a Difference (D) and Borrow-out (Bout).

- Difference (D) = $A \oplus B \oplus B_{in}$
- Borrow-out (Bout) = $A' \cdot (B \oplus B_{in}) + (B \cdot B_{in})$

3. Multiplexers (MUX): A multiplexer is a combinational circuit that has many data inputs and a single output, depending on control or select inputs. For N input lines, $\log_2(N)$ selection lines are required, or equivalently, for 2^n input lines, n selection lines are needed. [Multiplexers](#) are also known as “N-to-1 selectors,” parallel-to-serial converters, many-to-one circuits, and universal logic circuits.



Implementation of Higher-Order MUX Using Lower-Order MUX

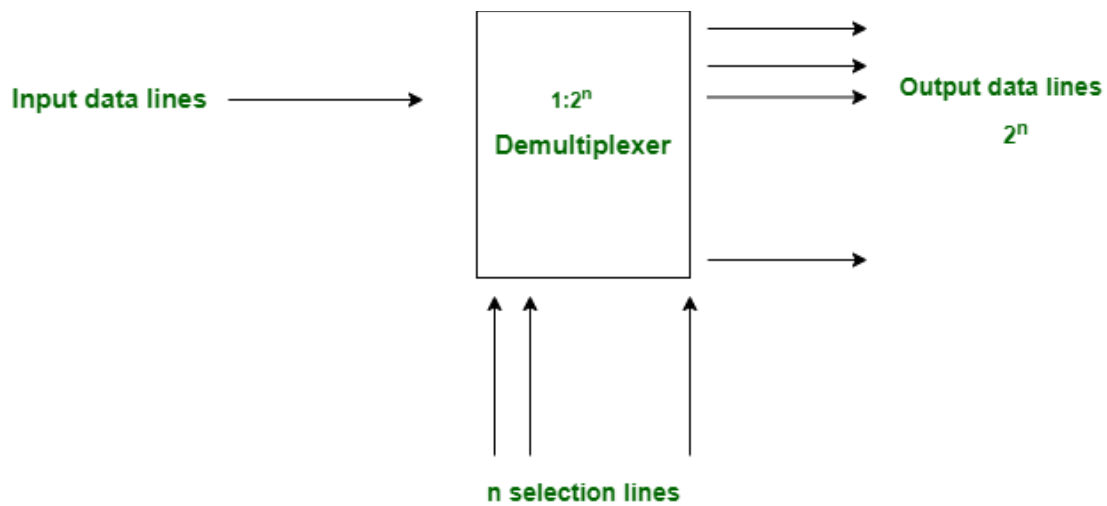
A **higher-order multiplexer (MUX)** can be built using multiple **lower-order MUXes** to handle more input lines. This is done by **breaking the selection process** into smaller steps using additional MUXes.

For example, an **8:1 MUX** can be implemented using **two 4:1 MUXes** and **one 2:1 MUX**:

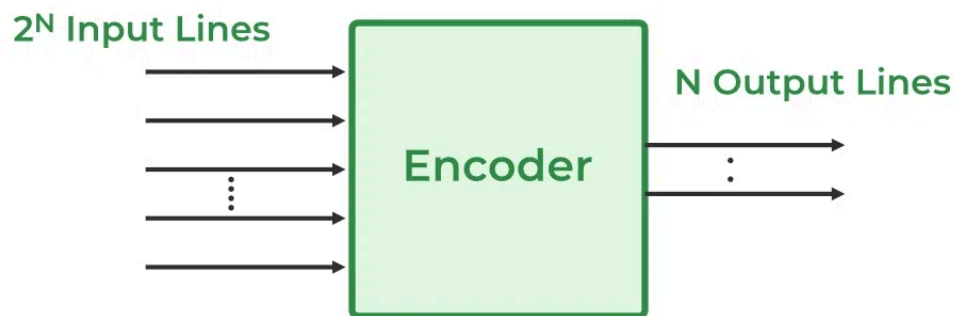
- The two **4:1 MUXes** take inputs and provide two outputs.
- The **2:1 MUX** selects one of these outputs based on the highest selection bit.

This method reduces complexity and allows modular circuit design.

4. Demultiplexers (DEMUX): The [DEMUX](#) is a digital information processor. It takes input from one source and also converts the data to transmit towards various sources. The demultiplexer has one data input line. The demultiplexer has several control lines (also known as select lines). These lines determine to which output the input data should be sent. The number of control lines determines the number of output lines.



5. Encoders: An [Encoder](#) is a combinational circuit that performs the reverse operation of a Decoder. It has a maximum of 2^n input lines and 'n' output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits.



Encoder

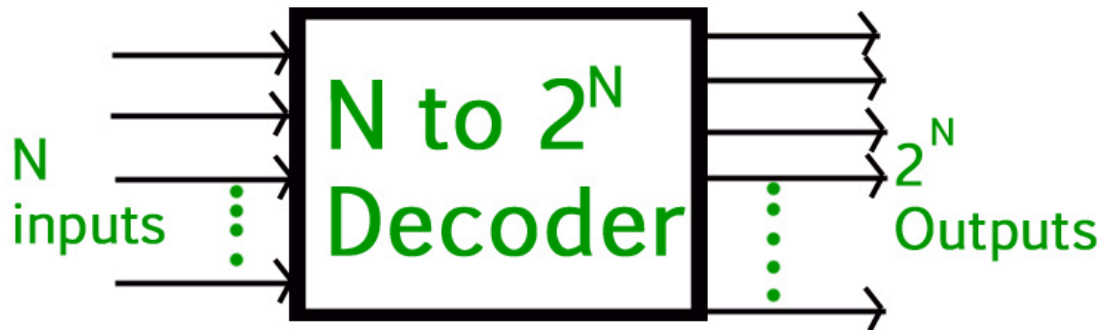
Priority Encoder

A priority encoder is a digital circuit that converts multiple input lines into a binary code based on priority. If multiple inputs are active, the highest-priority input (the one with the highest index) is encoded in the output.

Key Features:

- **Assigns priority** to inputs when more than one is active.
- **Ignores lower-priority inputs** when a higher-priority input is present.
- **Commonly used in interrupt controllers** and other selection-based applications.

6. Decoders: A decoder is a combinational circuit that converts an n -bit binary input data into 2^n output lines, such that each output line will be activated for only one of the possible combinations of inputs. [Decoders](#) are usually represented as n -to- 2^n line decoders, where n is the number of input lines and 2^n is the number of maximum possible output lines.



7. Comparators: The comparator is a combinational logic circuit. It compares the magnitude of two n -bit numbers and provides the relative result as the output. Let A and B are the two n -bit inputs. The comparator has three outputs namely $A > B$, $A = B$ and $A < B$. Depending upon the result of comparison, one of these outputs will go high.



Sequential Logic Circuits

Sequential circuits are digital circuits that store and use previous state information to determine their next state. They are commonly used in digital systems to implement state machines, timers, counters, and memory elements and are essential components in digital systems design.

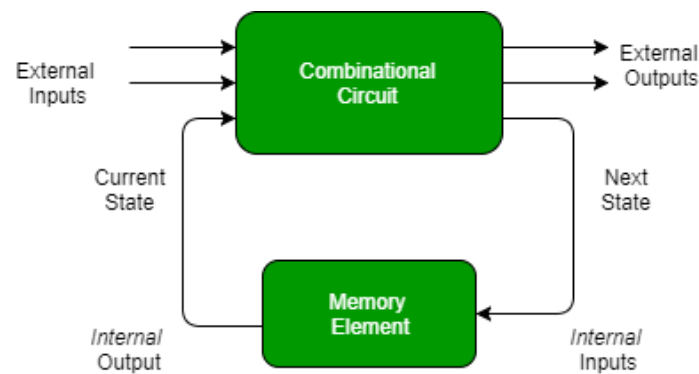


Figure: Sequential Circuit

Types of Sequential Circuits

There are two types of [sequential circuits](#):

1. Asynchronous Sequential Circuits: These circuits **do not use a clock signal** but uses the pulses of the inputs. These circuits are **faster** than synchronous sequential circuits because there is clock pulse and change their state immediately when there is a change in the input signal.

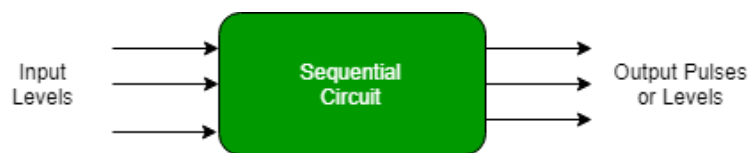


Figure: Asynchronous Sequential Circuit

2. Synchronous Sequential Circuits: These circuits **uses clock signal** and level inputs (or pulsed) (with restrictions on pulse width and circuit propagation). The output pulse is the same duration as the clock pulse for the clocked sequential circuits. Since they wait for the next clock pulse to arrive to perform the next operation, so these circuits are bit **slower** compared to asynchronous.

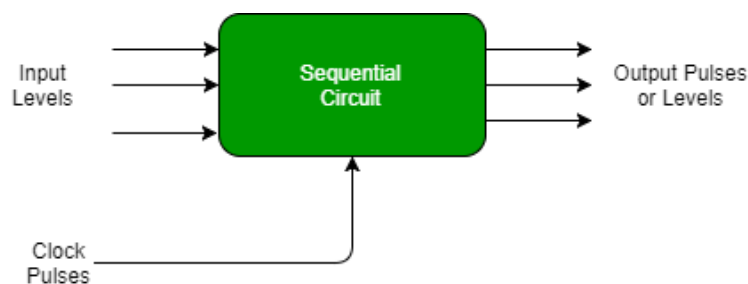


Figure: Synchronous Sequential Circuit

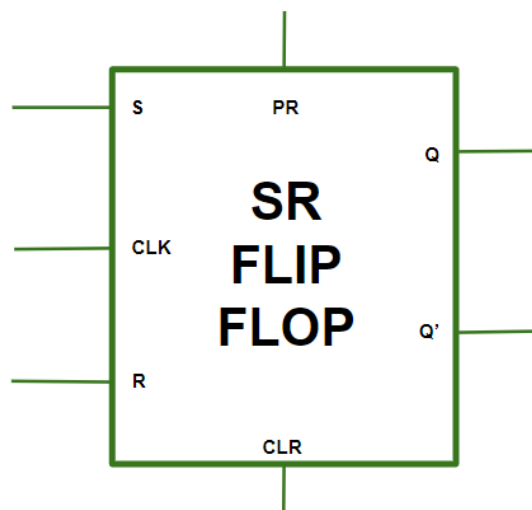
Flip-Flops

The flip-flop is a circuit that maintains a state until directed by input to change the state. A basic flip-flop can be constructed using four-NAND or four-NOR gates. Flip-flop is popularly known as the basic digital memory circuit. It has its two states as logic 1(High) and logic 0(low) states. A flip flop is a sequential circuit which consist of single binary state of information or data.

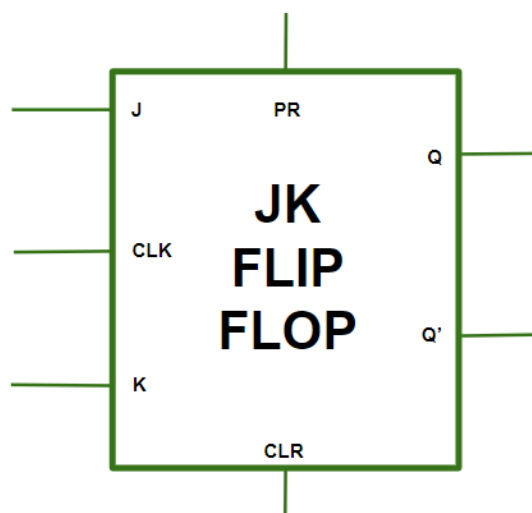
Types of Flip-Flops

Given Below are the Types of Flip-Flop:

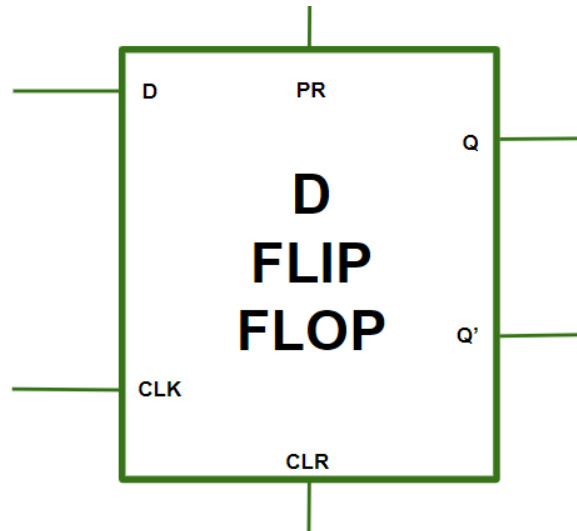
1. S-R Flip-Flop: In the flip flop, with the help of preset and clear when the power is switched ON, the states of the circuit keeps on changing, that is it is uncertain. It may come to set($Q=1$) or reset($Q'=0$) state. In many applications, it is desired to initially set or reset the flip flop that is the initial state of the flip flop that needs to be assigned. This thing is accomplished by the preset(PR) and the clear(CLR).



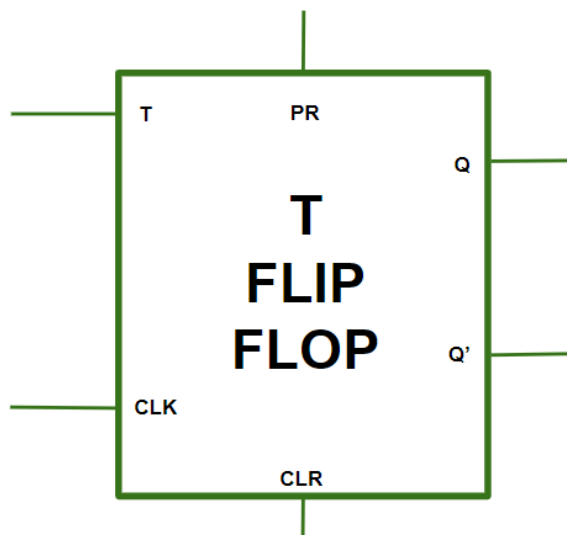
2. J-K Flip-Flop: n JK flip flops, The basic structure of the flip flop which consists of Clock (CLK), Clear (CLR), Preset (PR).



3. D Flip-Flop: The D Flip Flop Consists a single data input(D), a clock input(CLK),and two outputs: Q and Q' (the complement of Q).



4. T Flip-Flop: The T Flip Flop consists of data input (T), a clock input (CLK), and two outputs: Q and Q' (the complement of Q).



read more about – [Flip-Flops](#)

Number System

A number system is a method to represent numbers mathematically. It can use arithmetic operations to represent every number uniquely. To represent a number, it requires a base or radix.

Types of Number System

There are four common types of number systems based on the radix or base of the number :

1. Decimal Number System (Base 10 number system)

2. Binary Number System (Base 2 number system)
3. Octal Number System (Base 8 number system)
4. Hexadecimal Number System (Base 16 number system)

Number System Conversion Methods

A number N in base or radix b can be written as:

$$(N)_b = d_{n-1} d_{n-2} \dots d_1 d_0 . d_{-1} d_{-2} \dots d_{-m}$$

In the above, d_{n-1} to d_0 is the integer part, then follows a radix point, and then d_{-1} to d_{-m} is the fractional part.

d_{n-1} = Most significant bit (MSB)

d_{-m} = Least significant bit (LSB)

Base	Representation
2	Binary
8	Octal
10	Decimal
16	Hexadecimal

1. Decimal to Binary Number System: To convert from decimal to binary, start dividing decimal number by 2, and whatever the remainder getting, writing down from bottom to top, and that will be the binary number representation of the decimal number. And the number contains fractional part, then multiply 2 in the fractional part.

Example: $(10.25)_{10}$

Integer part :

2	10	0
2	5	1
2	2	0
	1	

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

Fractional part

$$\begin{array}{l} \vdots \\ 0.25 \times 2 = 0.50 \\ 0.50 \times 2 = 1.00 \end{array} \downarrow$$

$$(0.25)_{10} = (0.01)_2$$

2. Binary to Decimal Number System: To convert from binary to decimal, start multiplying the exponent of 2 with each digit of the number in decreasing order. If

the number contains fractional part then will divide it by the exponent of 2.

Example:

$$\begin{aligned} &(1010.01)_2 \\ &1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 8 + 0 + 2 + 0 + 0 + 0.25 = 10.25 \\ &(1010.01)_2 = (10.25)_{10} \end{aligned}$$

3. Decimal to Octal Number System: To convert from decimal to octal, start dividing decimal number by 8, and whatever the remainder getting, writing down from bottom to top, and that will be the octal number representation of the decimal number. And the number contains fractional part, then multiply 8 in the fractional part.

Example:

$$\begin{aligned} &(10.25)_{10} \\ &(10)_{10} = (12)_8 \\ &\text{Fractional part:} \\ &0.25 \times 8 = 2.00 \end{aligned}$$

4. Octal to Decimal Number System: To convert from octal to decimal, start multiplying the exponent of 8 with each digit of the number in decreasing order. If the number contains fractional part then will divide it by the exponent of 8.

Example:

$$\begin{aligned} &(12.2)_8 \\ &1 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} = 8 + 2 + 0.25 = 10.25 \\ &(12.2)_8 = (10.25)_{10} \end{aligned}$$

5. Hexadecimal to Binary Number System: To convert from Hexadecimal to Binary, write the 4-bit binary equivalent of hexadecimal.

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

Example: $(3A)_{16} = (00111010)_2$

6. Binary to Hexadecimal Number system: To convert from Binary to Hexadecimal, start grouping the bits in groups of 4 from the right-end and write the equivalent hexadecimal for the 4-bit binary. Add extra 0's on the left to adjust the groups.

Example:

1111011011
 001111011011
 $(001111011011)_2 = (3DB)_{16}$

7. Binary to Octal Number System: To convert from binary to octal, start grouping the bits in groups of 3 from the right end and write the equivalent octal for the 3-bit binary. Add 0's on the left to adjust the groups.

Example:

111101101
 $111\ 101\ 101$
 $(111101101)_2 = (755)_8$

read more about – [Number System](#)

Dreaming of **M.Tech in IIT**? Get AIR under 100 with our [GATE 2026 CSE & DA courses](#)! Get flexible **weekday/weekend** options, **live mentorship**, and **mock tests**. Access exclusive features like **All India Mock Tests**, and Doubt Solving—your GATE success starts now!

[Comment](#)[More info](#)[Advertise with us](#)

Next Article

Digital Logic and Design - GATE CSE
Previous Year Questions

Similar Reads

LMN - Digital Electronics

Digital electronics deals with systems that use digital signals, represented as 0s and 1s, to process information. It is the backbone of modern devices like computers,...

14 min read

Logic Synthesis in Digital Electronics

The process of resolving into component parts or analyzing, interpreting, translating, optimizing (rearranging or rewriting to improve efficiency), and mapping RTL (Registe...

5 min read

Basics of Boolean Algebra in Digital Electronics

In the time middle of the 19th century, mathematician George Boole introduced a new form of mathematical reasoning system which is widely known as Boolean Algebra....

6 min read

Digital Electronics and Computer Organisation

Digital Electronics Half Adder Half Subtractor K-Map Counters Computer Organisation and Architecture Machine Instructions What's difference between 1's Complement an...

1 min read

Block Coding in Digital Electronics

The block coding is a technique through which much reliability is injected into the data that is sent over the network or kept in a device. It does this by adding extra...

8 min read

What is Scrambling in Digital Electronics ?

A computer network is designed to send information from one point to another. Data that we send can either be digital or analog. Also, signals that represent data can als...

5 min read

Operational Amplifier (op-amp) in Digital Electronics

An amplifier is a device that increases the strength of the input signal. It can be Voltage amplifiers, whose input is some voltage and output is also voltage but amplified....

4 min read

Digital Electronics - Radix and Diminished Radix Complements

Prerequisite - Complement of a Number The use of complements is to mainly perform Subtraction. We can easily perform addition in contrast if we would like to implement...

4 min read

Wired Logic in Digital Electronics

Wired logic is a type of digital logic where some logic operations are carried out by directly coupling the outputs of a single or several logic gates. This approach is...

4 min read

What is Register in Digital Electronics ?

A register is a small and temporary storage unit inside a computer's (CPU). It plays a vital role in holding the data required by the CPU for immediate processing and is...

9 min read

Registered Address:

K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305



Advertise with us

Company

About Us
Legal
Privacy Policy
Careers
In Media
Contact Us
GFG Corporate Solution
Placement Training Program

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Python Tutorial

Python Programming Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scraping

Explore

Job-A-Thon Hiring Challenge
Hack-A-Thon
GfG Weekly Contest
Offline Classes (Delhi/NCR)
DSA in JAVA/C++
Master System Design
Master CP
GeeksforGeeks Videos
Geeks Community

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
DSA Interview Questions
Competitive Programming

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
NodeJs
Bootstrap
Tailwind CSS

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering

[OpenCV Tutorial](#)
[Python Interview Question](#)

DevOps

[Git](#)
[AWS](#)
[Docker](#)
[Kubernetes](#)
[Azure](#)
[GCP](#)
[DevOps Roadmap](#)

School Subjects

[Mathematics](#)
[Physics](#)
[Chemistry](#)
[Biology](#)
[Social Science](#)
[English Grammar](#)

Databases

[SQL](#)
[MYSQL](#)
[PostgreSQL](#)
[PL/SQL](#)
[MongoDB](#)

Competitive Exams

[JEE Advanced](#)
[UGC NET](#)
[UPSC](#)
[SSC CGL](#)
[SBI PO](#)
[SBI Clerk](#)
[IBPS PO](#)
[IBPS Clerk](#)

Free Online Tools

[Typing Test](#)
[Image Editor](#)
[Code Formatters](#)
[Code Converters](#)
[Currency Converter](#)
[Random Number Generator](#)
[Random Password Generator](#)

DSA/Placements

[Digital Logic Design](#)
[Engineering Maths](#)

System Design

[High Level Design](#)
[Low Level Design](#)
[UML Diagrams](#)
[Interview Guide](#)
[Design Patterns](#)
[OOAD](#)
[System Design Bootcamp](#)
[Interview Questions](#)

Commerce

[Accountancy](#)
[Business Studies](#)
[Economics](#)
[Management](#)
[HR Management](#)
[Finance](#)
[Income Tax](#)

Preparation Corner

[Company-Wise Recruitment Process](#)
[Resume Templates](#)
[Aptitude Preparation](#)
[Puzzles](#)
[Company-Wise Preparation](#)
[Companies](#)
[Colleges](#)

More Tutorials

[Software Development](#)
[Software Testing](#)
[Product Management](#)
[Project Management](#)
[Linux](#)
[Excel](#)
[All Cheat Sheets](#)
[Recent Articles](#)

Write & Earn

[Write an Article](#)
[Improve an Article](#)
[Pick Topics to Write](#)
[Share your Experiences](#)
[Internships](#)

Development/Testing

DSA - Self Paced Course
DSA in JavaScript - Self Paced Course
DSA in Python - Self Paced
C Programming Course Online - Learn C with Data Structures
Complete Interview Preparation
Master Competitive Programming
Core CS Subject for Interview Preparation
Mastering System Design: LLD to HLD
Tech Interview 101 - From DSA to System Design [LIVE]
DSA to Development [HYBRID]
Placement Preparation Crash Course [LIVE]

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]
Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]
Data Science Training Program - [LIVE]
Mastering Generative AI and ChatGPT
Data Science Course with IBM Certification

Clouds/Devops

DevOps Engineering
AWS Solutions Architect Certification
Salesforce Certified Administrator Course

JavaScript Full Course
React JS Course
React Native Course
Django Web Development Course
Complete Bootstrap Course
Full Stack Development - [LIVE]
JAVA Backend Development - [LIVE]
Complete Software Testing Course [LIVE]
Android Mastery with Kotlin [LIVE]

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course

GATE

GATE CS & IT Test Series - 2025
GATE DA Test Series 2025
GATE CS & IT Course - 2025
GATE DA Course 2025
GATE Rank Predictor