1 to 16 Decoder

The 4 to 16 decoder is the type of decoder which has 4 input lines and 16 (2^{14}) output lines. The functional block diagram of the 4 to 16 decoder is shown in Figure-6.

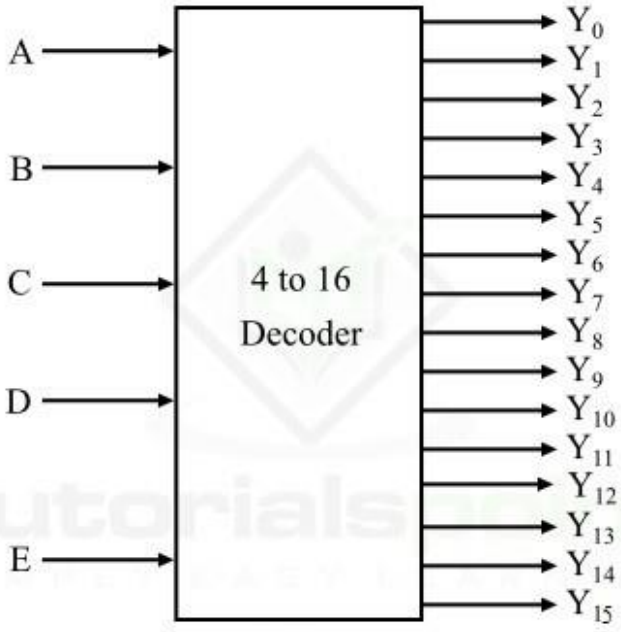


Figure 6 - 4 to 16 Decoder

When this decoder is enabled with the help of enable input E, it's one of the sixteen outputs will be active for each combination of inputs. The operation of the 4-line to 16-line decoder can be analyzed with the help of its function table which is given below.

Inputs					Output
E	**A**	**B**	**C**	**D**	
0	X	X	X	X	0
1	0	0	0	0	Y_0
1	0	0	0	1	Y_1

1	0	0	1	0	Y_2
1	0	0	1	1	Y_3
1	0	1	0	0	Y_4
1	0	1	0	1	Y_5
1	0	1	1	0	Y_6
1	0	1	1	1	Y_7
1	1	0	0	0	Y_8
1	1	0	0	1	Y_9
1	1	0	1	0	Y_{10}
1	1	0	1	1	Y_{11}
1	1	1	0	0	Y_{12}
1	1	1	0	1	Y_{13}
1	1	1	1	0	Y_{14}
1	1	1	1	1	Y_{15}

Binary Addition

In binary arithmetic, the process of adding two binary numbers is called binary addition. Where, the binary numbers consist of only 0 and 1. In the binary addition, a carry is generated when the sum is greater than 1.

Rules of Binary Addition

The addition of two binary numbers is performed according to these rules of binary arithmetic –

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 (\text{Sum} = 0 \,\&\, \text{Carry} = 1)$$

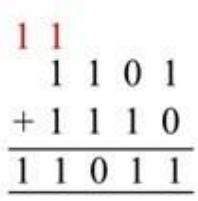
Let us consider some examples to understand the binary addition.

Example 1

Add two binary numbers, 1101 and 1110.

Solution

The binary addition of the given binary numbers is described below −



Explanation

Add 1 (rightmost bit of first number) and 0 (rightmost bit of the second number). It gives 1 + 0 = 1 (thus, write down 1 as sum bit).

Add 0 (second rightmost bit of first number) and 1 (second rightmost bit of the second number). It gives 0 + 1 = 1 (write down 1 as sum bit).

Add 1 (third rightmost bit of first number) and 1 (third rightmost bit of second number). It gives 1 + 1 = 10 (write down 0 as sum and 1 as carry).

Add 1 (leftmost bit of the first number), 1 (leftmost bit of second number) and 1 (carry). It gives 1 + 1 + 1 = 11 (write down 1 as sum and 1 as carry).

Write the end around carry 1 in the sum.

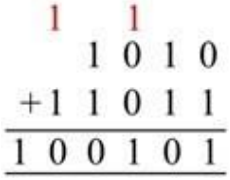
Thus, the result is 11011.

Example 2

Add 1010 and 11011.

Solution

The binary addition of given numbers is explained below −



Explanation

Add 0 (rightmost bit of first number) and 1 (rightmost bit of second number). It gives 0 + 1 = 1 (write down 1 as sum).

Add 1 (second rightmost bit of first number) and 1 (second rightmost bit of second number). It gives 1 + 1 = 10 (write down 0 as sum and 1 as carry).

Add 0 (third rightmost bit of first number), 0 (third rightmost bit of second number), and 1 (carry). It gives 0 + 0 + 1 = 1 (write down 1 as sum).

Add 1 (leftmost bit of first number) and 1 (second leftmost bit of second number). It gives 1 + 1 = 10 (write down 0 as sum and 1 as carry).

Add 1 (leftmost bit of second number) and 1 carry. It gives 1 + 1 = 10 (write down 0 as sum and 1 as the end around carry).

Hence, the sum of 1010 and 11011 is 100101.

Binary Subtraction

In binary arithmetic, binary subtraction is a mathematical operation used to find the difference between two binary numbers. In binary subtraction, each bit of the binary numbers is subtracted, starting from the rightmost bit.

Also, a borrow bit can be taken from higher bits if require.

Rules of Binary Subtraction

The binary subtraction is performed as per the following rules of binary arithmetic −

$$0\ 0\ =\ 0$$

$$1\ 0\ =\ 1$$

$$0\ 1\ =\ 1\ (\text{borrow 1 from the next higher bit})$$

$$1\ 1\ =\ 0$$

Let us see some examples to understand the binary subtraction.

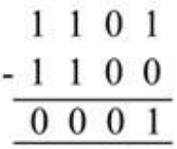
Example 1

Subtract 1100 from 1101.

Solution

The subtraction of given binary numbers is given below –

1101 1100 = 0001



Explanation

Subtract 0 (rightmost bit of second number) from 1 (rightmost bit of first number). It gives 1 0 = 1 (write down 1 as difference).

Subtract 0 (second rightmost bit of second number) from 0 (second rightmost bit of first number). It gives 0 0 = 0 as result.

Subtract 1 (third rightmost bit of second number) from 1 (third rightmost bit of first number). It gives 1 1 = 0 as result.

Subtract 1 (leftmost bit of second number) from 1 (leftmost bit of first number). It gives 1 1 = 0 as result.

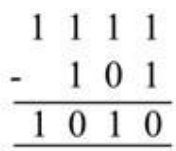
Thus, the difference of 1101 and 1100 is 0001.

Example 2

Subtract 101 from 1111.

Solution

The subtraction of given binary numbers is explained below –



Explanation

Subtract rightmost bits: 1 1 = 0

Subtract second rightmost bits: 1 1 = 1

Subtract third rightmost bits: 1 1 = 0

Subtract leftmost bits: 1 0 = 1

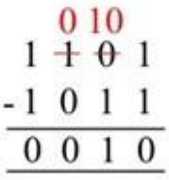
Thus, the result is 1010.

Example 3

Subtract 1011 from 1101.

Solution

The binary subtraction of 1101 and 1011 is given below −



Explanation

Subtract rightmost bits: 1 1 = 0.

Subtract second rightmost bits: 0 1 = 1. A borrow 1 is taken from the next higher bit.

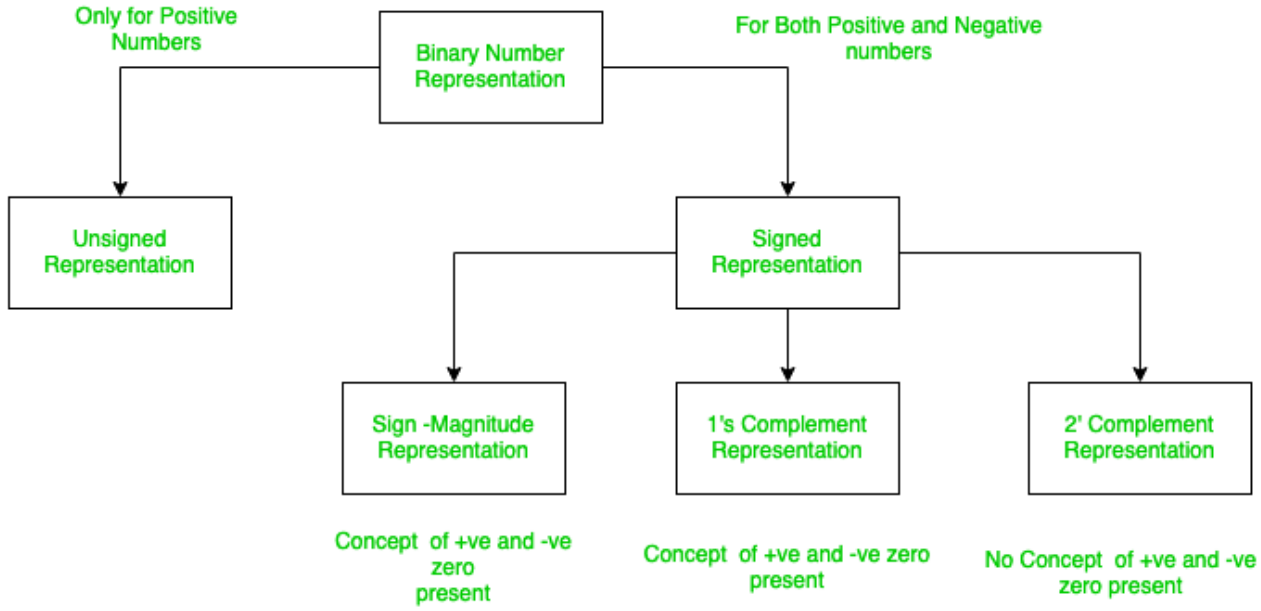
Subtract third rightmost bits: 0 0 = 0. The 1 borrow is given to previous bit.

Subtract leftmost bits: 1 1 = 0. Thus, the difference of 1101 and 1011 is 0010.

Unsigned and Signed Numbers Representation in Binary Number System

The binary number system uses only two digits, 0 and 1, to represent all data in computing and digital electronics. Understanding unsigned and signed numbers is important for efficient data handling and accurate computations in these fields.

- The binary system forms the foundation of all digital systems, enabling devices to process and store data.
- Unsigned numbers represent only positive values while signed numbers handle both positive and negative values using methods like two’s complement.
- Mastering these concepts is essential for programming, error-free calculations and optimizing system performance.
- Its real-world applications include computer arithmetic, embedded systems and digital signal processing.



Unsigned Numbers

Unsigned numbers are numeric values that represent only non-negative quantities (zero and positive values). In the binary system, unsigned numbers are represented using only the magnitude of the value, with each bit contributing to its total. The smallest unsigned number is always zero (all bits set to 0), while the maximum value depends on the number of bits used.

Range of Unsigned Numbers in Binary System:

The range depends directly on the bit length. For an n-bit number:

- Minimum value: 0
- Maximum value: 2n - 1

Range of unsigned numbers: 0 to 2n-1

Example:

- 4-bit Number: Ranges 0 to 15.
- 8-bit Number: Ranges 0 to 255.
- 16-bit Number: Ranges 0 to 65,535.

Use Cases of Unsigned Binary Representation:

Unsigned binary numbers serve fundamental roles across computing systems where negative values aren't needed.

- Every byte in RAM gets a unique unsigned address. A 32-bit system can access 4GB of memory (0 to 4,294,967,295).
- RGB color values (0-255) use 8-bit unsigned numbers per channel. Image dimensions and pixel counts also rely on unsigned ranges.
- Microprocessors use unsigned values for status flags and control signals where only positive states exist.

- Packet sizes, port numbers and IP header fields often use unsigned integers to prevent negative interpretations.
- Sensor readings (like temperature or light intensity) and timer counts utilize unsigned formats when negative measurements are impossible.

Sign-Magnitude Representation

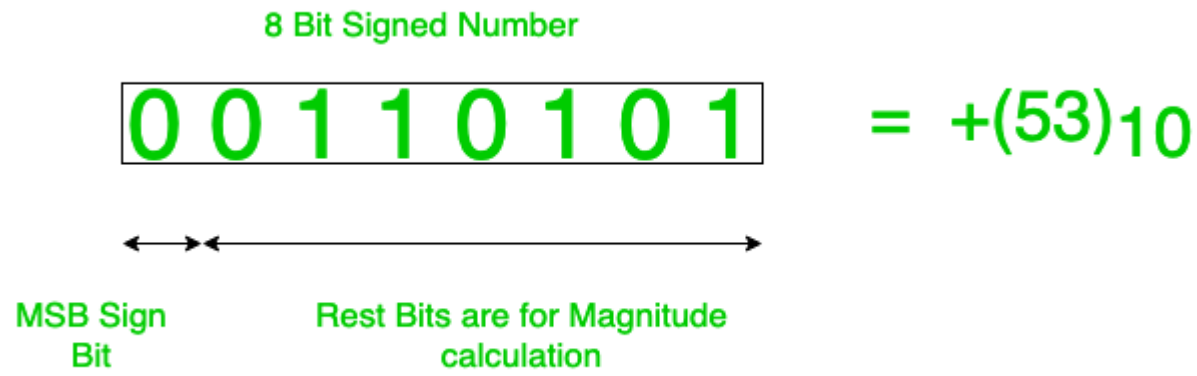
This method uses the leftmost bit as a sign flag (0 for positive, 1 for negative) while remaining bits store the absolute value. It creates two zero representations (+0 and -0) and complicates arithmetic operations due to separate sign handling.

Range of Sign-Magnitude Representation

-2n-1 - 1 to 2n-1 - 1

Example: In 8-bit system

- *Decimal number +25 is represented as 00011001 (Sign bit 0, magnitude 25)*
- *Decimal number -25 is represented as 10011001 (Sign bit 1, magnitude 25)*



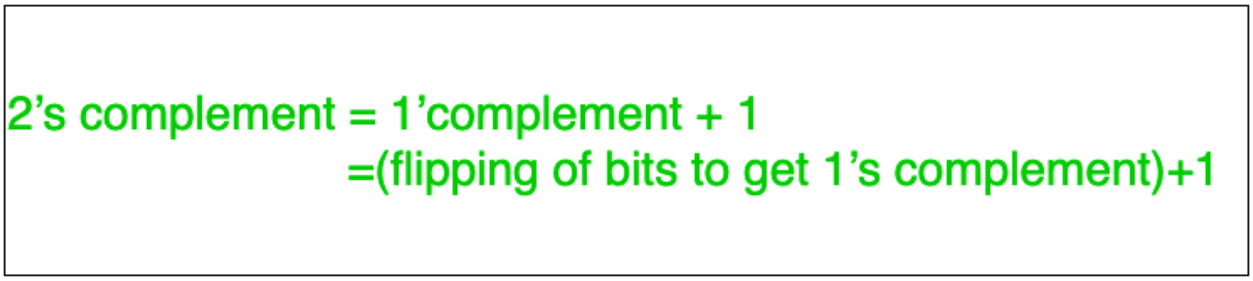
Sign - Magnitude Representation

Why Sign-Magnitude Fails in Practice

- Wastes a representable value by encoding both +0 (0000) and -0 (1000), creating unnecessary complexity in comparisons and arithmetic.
- Basic calculations fail because the sign bit requires separate handling. Adding positive and negative versions of the same number doesn't produce zero.
- Requires extra circuitry to manage sign bits during calculations, slowing down processors compared to complement systems.
- Effectively loses one bit of precision since the sign bit doesn't contribute to magnitude, unlike modern systems that use all bits for value representation.

2's Complement Representation

2's complement forms negative numbers by inverting all bits of the positive value and adding 1. The leftmost bit serves as the sign indicator while enabling single zero representation. It simplifies hardware design by allowing identical addition and subtraction operations.



2's Complement

Range of 2's Complement Representation

-2n-1 to 2n-1 - 1

Example: In 8-bit system

- *Decimal number +20 is represented as 00010100*
- *Decimal number -20 is represented as 11101100 (Invert bits: 11101011, then add 1)*

Limitations of 2's Complement

- In a fixed-bit system, 2's complement has a limited range. For an n-bit system, it represents values from -2(n-1) to 2(n-1) - 1, restricting the number of values that can be processed.
- The range for positive numbers is smaller than negative numbers. Negative values range from -2(n-1) to -1, while positive values range from 0 to 2(n-1) - 1, creating an imbalance.
- Overflow occurs when arithmetic operations exceed the representable range, leading to incorrect results.
- Operations like addition and subtraction become complex with sign extension, especially for numbers of different sizes.
- The zero in 2's complement has only one form, while negative numbers have a mirrored representation (e.g., -1 is 111...1), complicating algorithms like multiplication.

Arithmetic Operations of 2's Complement Number System

We all know how to perform arithmetic operations of binary number systems. However, computer system generally stores numbers in 2's complement format. So it becomes necessary for us to know how arithmetic operations are done in 2's complement.

Addition:

In 2's complement, we perform addition the same as an addition for unsigned binary numbers. The only difference is here we discard the out carry i.e carry from MSB bit as long as the range lies within the accepted range for 2's complement representation.

For eg: Consider a number of bits(n) = 4. So range of numbers will be -2^{n-1} \space to \space 2^{n-1} , i.e. -8 to 7.

```
x=2, y=3

addition: 0010

          0011
```

```
        0101

---> +5 in 4 bits

x=-2,y=-3

addition: 1110

        1101

        1011

---> -5 in 4 bits

(discarding the out carry 1)
```

In the above examples, there was no overflow as the answers +5 and -5 lie in the range of -8 to 7.

Consider the example,

```
x=4, y=5

addition: 0100

        0101

        1001

---> -7 (not in range of -8 to 7)
```

Hence overflow occurs in the above example as -7 does not exist in the 4bits range.

For more details on overflow, you can refer to the following article Overflow in Arithmetic Addition in Binary Number System.

Subtraction

Consider if we have to find the subtraction x-y, we perform the following steps:

1. Take 2's complement of y
2. Add x and 2's complement of y

for eg:

```
x=3, y=2 for 4 bits 2's complement representation



3---> 0011

Take 2's complement of 2----> 0010

1101 + 1 ---> 1110
```

Adding 0011 and 1110

0011

1110

0001

---> +1 (discard carry 1 from MSB)

Multiplication:

If operands x and y which need to be multiplied are of n and m digits respectively then the result will be n+m digits. To get correct results we extend digits of both operands to n+m digits

For eg: x=13, y=-6

13 requires 5(m) bits for representation. and -6 requires 4 bits(n) so we will make the bits of 13 and -6 to be equal to m+n = 5+4 = 9

13 ------> 000001101

-6 ------> 111111010

(2's complement of 6)

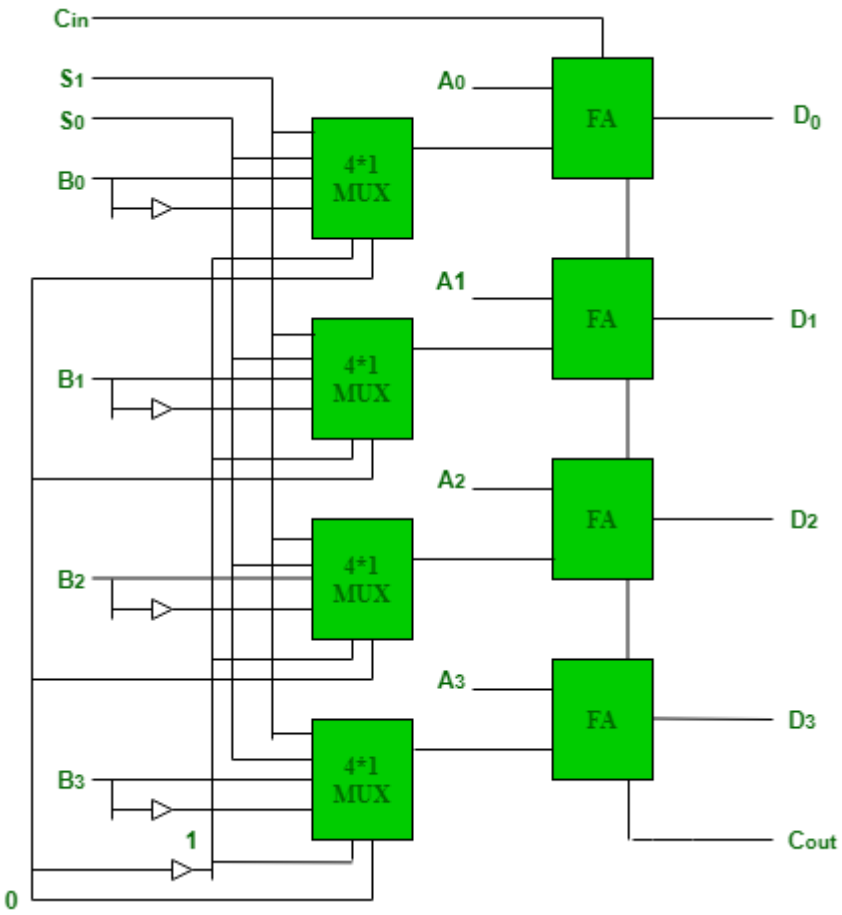
We then take 2's complement of 110110010 which is 001001110 which is 78. And so 110110010 is -78.

Arithmetic Circuit Building blocks

Arithmetic circuits are fundamental blocks in digital systems and are used for arithmetic operations such as addition, subtraction, multiplication and division**. Arithmetic circuits** can perform seven different arithmetic operations using a single composite circuit.
It uses a full adder (FA) to perform these operations. A multiplexer (MUX) is used to provide different inputs to the circuit in order to obtain different arithmetic operations as outputs.

4-Bit Arithmetic Circuit

Consider the following 4-bit Arithmetic circuit with inputs A and B. It can perform seven different arithmetic operations by varying the inputs of the multiplexer and the carry (C0).



Truth Table for the above Arithmetic Circuit

S0	S1	C0	MUX Output	Full Adder Output
0	0	0	B	A + B
0	0	1	B	A + B + 1
0	1	0	B'	A + B'
0	1	1	B'	A + B' + 1 = A - B
1	0	0	0	A
1	0	1	0	A + 1
1	1	0	1	A - 1
1	1	1	1	A - 1 + 1 = A

Hence, the different operations for the inputs A and B are -

1. A + B (adder)
2. A + B + 1
3. A + B'
4. A - B (subtracter)
5. A
6. A + 1 (incrementor)
7. A - 1 (decremental)

Adder Circuit

Adder circuits are basic arithmetic circuits which are used for binary addition. They come in various types based on their complexity and function. They come in various types based on their complexity and function:

1. Half Adder

A Half adder is the basic type of the adder circuit, which is used in many digital designs. It performs addition of two one-bit binary numbers and generates an output in terms of sum and carry. The half adder consists of two logic gates: an XOR gate for the sum and an AND gate for the carry.

2. Full Adder

A Full adder is an improvement of the half adder since it performs addition of three binary numbers (two inputs and a carry from the previous addition). It produces a sum and a carry; it is made of two half adders and an OR gate for the carry.

3. Ripple Carry Adder

This kind of adder is made up of more than one full adder cascaded together in a serial manner. The carry output of each full adder is passed to the next; the circuit is simple but carries propagation delay makes it relatively slow.

4. Carry Look-Ahead Adder

Due to the delay issue in ripple carry adders, carry look-ahead adders are built in order to produce carry signals at a faster rate due to increased complexity. They enhance the speed of addition to a very large extent and thus are suitable where high speed processors are required.

Subtractor Circuit

Subtractor circuits perform binary subtraction and come in two primary types: Subtractor circuits perform binary subtraction and come in two primary types

1. Half Subtractor

The half subtractor is used to carry out subtraction with two single bit binary numbers. It produces a difference and a borrow. As in the half adder, the difference is produced using an XOR gate while the borrow is produced using an AND gate with an inverted input.

2. Full Subtractor

A full subtractor extends the half subtractor to handle three inputs: Two binary numbers and borrow from the foregoing stage. And it produces both a difference and a borrow and the circuit diagram consists of two half subtractors and an OR gate for the borrow transfer.

Multiplier Circuit

Arithmetic circuits include the multiplier circuits which are used to multiply binary numbers. They can be classified into various types based on complexity and speed. They can be classified into various types based on complexity and speed:

1. Array Multiplier

An array multiplier employs several adders in an array structure to multiply two binary numbers. They are obtained with the help of AND gates and added subsequent to by using adders.

2. Booth Multiplier

Booth's algorithm is used in multiplication of binary numbers that are positive and also negative values. It minimizes the number of partial products and therefore the multiplication is faster than that of the array multiplier.

3. Wallace Tree Multiplier

It is a kind of multiplier that uses a tree like structure to minimize on the number of adders needed in the computation of the product of two binary numbers. The Wallace tree multiplier is faster than the other array multipliers because the number of carry propagate is minimized.

Divider Circuit

Divider circuits perform binary division, and they come in two types Divider circuits perform binary division, and they come in two types:

1. Restoring Divider

The restoring division algorithm works by subtracting the divisor from the remainder and then taking the remainder and continuing the process until the division process is complete. It is less complex than non-restoring division but takes more time than the latter one.

2. Non-Restoring Divider

The new algorithm outperforms the restoring division method in that there is no need to restore the remainder at each iteration thus making division faster. Non-restoring dividers are suited best for high speed operation.

Arithmetic circuits are circuits that are used in performing arithmetic operations such as addition subtraction multiplication and division.

Adders and Subtractors

In digital electronics, **adders** and **subtractors** both are the combinational logic circuits (a combinational logic circuit is one whose output depends only on the present inputs, but not on the past outputs) that can add or subtract numbers, more specifically binary numbers. Adders and subtractors are the crucial parts of arithmetic logic circuits in processing devices like microprocessors or microcontrollers. In this article, we will discuss adders and subtractors in detail.

What is an Adder?

We have different types of digital devices like computers, calculators that can perform a variety of processing functions like addition, subtraction, multiplication, division, etc. The most basic arithmetic operation that the ALU (arithmetic logic unit) of a computer performs is the **addition** of two or more binary numbers. To perform the operation of addition, a combinational logic circuit, named **Adder** is used.

Adders are classified into two types namely ?

- Half Adder Full
- Adder

A **half-adder** is a combinational logic circuit that performs the addition of only two bits (binary digits). Whereas, a **full-adder** is a combination circuit that performs three bits (binary digits), where two are the significant bits and one is a carry from previous execution.

What is a Half-Adder?

A combinational logic circuit which is designed to add two binary digits is called as a **half adder**. The half adder provides the output along with a carry value (if any). The half adder circuit is designed by connecting an EX-OR gate and one AND gate. It has two input terminals and two output terminals for sum and carry. The block diagram and circuit diagram of a half adder are shown in Figure-1.

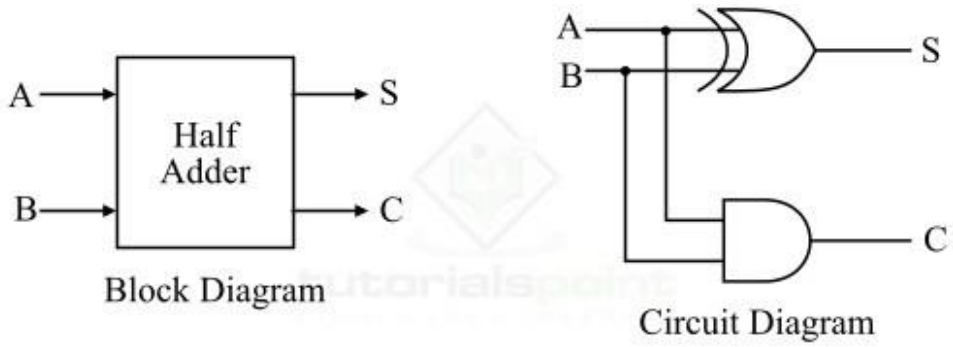


Figure 1 - Half Adder

In the case of a half adder, the output of the EX-OR gate is the sum of two bits and the output of the AND gate is the carry. Although, the carry obtained in one addition will not be forwarded in the next addition because of this it is known as half adder.

Truth Table of Half Adder

The following is the truth table of the half-adder ?

Inputs		Outputs	
A	**B**	**S (Sum)**	**C (Carry)**
0	0	0	0

0	1	1	0
1	0	1	0
1	1	0	1

Characteristic Equations of Half-Adder

The characteristic equations of half adder, i.e., equations of sum (S) and carry (C) are obtained according to the rules of binary addition. These equations are given below ?

The sum (S) of the half-adder is the XOR of A and B. Thus,

$$\text{Sum, } S = A \oplus B = AB' + A'B$$

The carry (C) of the half-adder is the AND of A and B. Therefore,

$$\text{Carry, } C = A \cdot B$$

What is a Full Adder?

A combinational logic circuit that can add two binary digits (bits) and a carry bit, and produces a sum bit and a carry bit as output is known as a **full-adder**.

In other words, a combinational circuit which is designed to add three binary digits and produces two outputs (sum and carry) is known as a full adder. Thus, a full adder circuit adds three binary digits, where two are the inputs and one is the carry forwarded from the previous addition. The block diagram and circuit diagram of the full adder are shown in Figure-2.

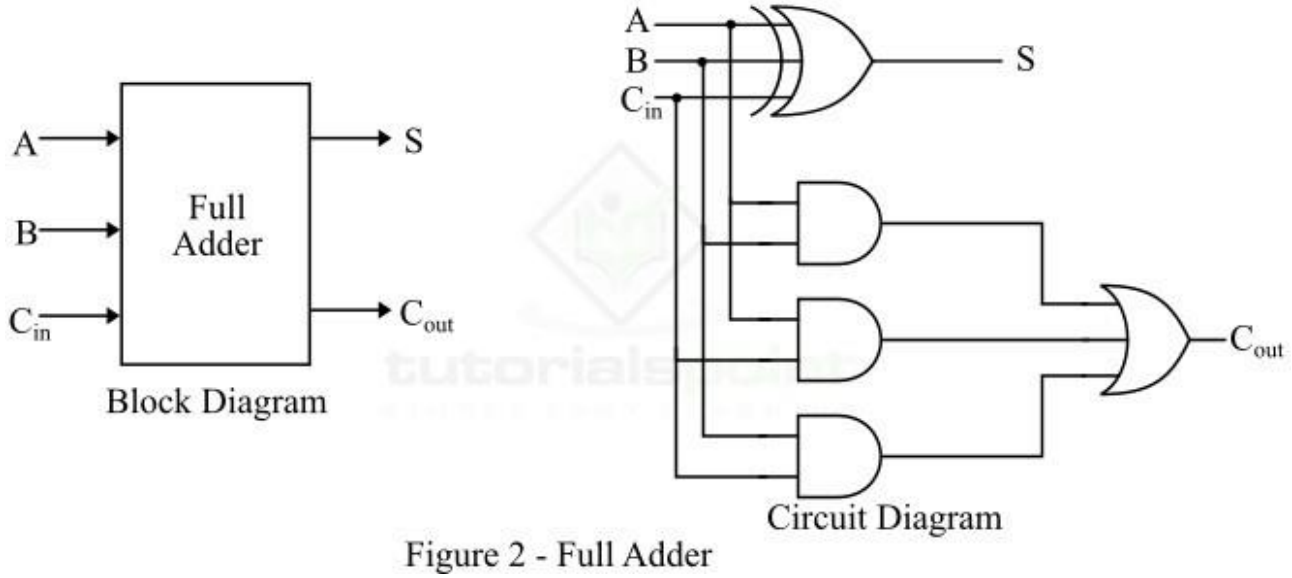


Figure 2 - Full Adder

Hence, the circuit of the full adder consists of one EX-OR gate, three AND gates and one OR gate, which are connected together as shown in the full adder circuit in Figure-2.

Truth Table of Full Adder

The following is the truth table of the full-adder circuit ?

Inputs			Outputs	
A	**B**	**C_{in}**	**S (Sum)**	**C (Carry)**
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

Hence, from the truth table, it is clear that the sum output of the full adder is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. While the carry output has a carry of 1 if two or three inputs are equal to 1.

Characteristic Equations of Full Adder

The characteristic equations of the full adder, i.e. equations of sum (S) and carry output (C_{out}) are obtained according to the rules of binary addition. These equations are given below ?

The sum (S) of the full-adder is the XOR of A, B, and Cin. Therefore,

$$\text{Sum, } S = A \oplus B \oplus C_{in} = A'B'C_{in} + A'BC_{in}' + AB'C' + ABC_{in}$$

Let us discuss the half-subtractor and full-subtractor in detail.

What is a Half-Subtractor?

A **half-subtractor** is a combinational logic circuit that have two inputs and two outputs (i.e. difference and borrow). The half subtractor produces the difference between the two binary bits at the input and also produces a borrow output (if any). In the subtraction (A- B), A is

called as **Minuend bit** and B is called as Subtrahend bit. The block diagram and logic circuit diagram of the half subtractor is shown in Figure-3.

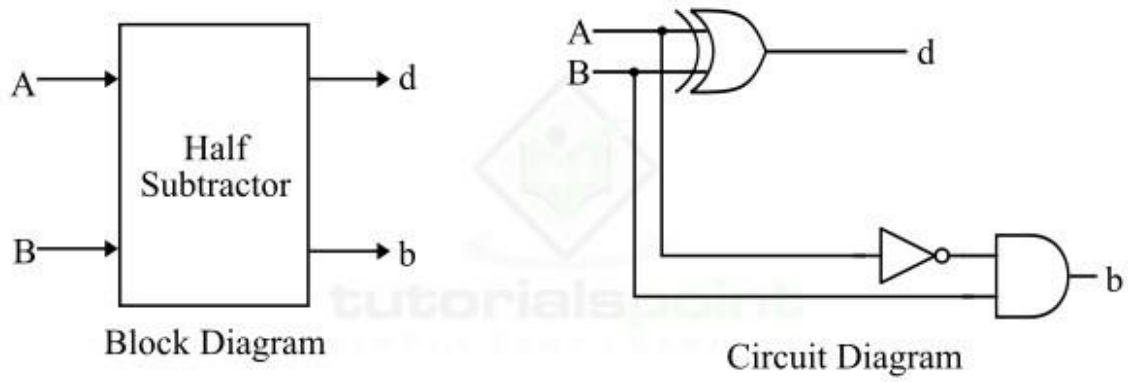


Figure 3 - Half Subtractor

Hence, from the logic circuit diagram, it is clear that a half subtractor can be realized using an XOR gate together with a NOT gate and an AND gate.

Truth Table of Half Subtractor

The following is the truth table the half-subtractor ?

Inputs		Outputs	
A	**B**	**d (Difference)**	**b (Borrow)**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Characteristic Equation of Half Subtractor

The characteristic equations of the half subtractor, i.e. equations of the difference (d) and borrow output (b) are obtained by following the rules of binary subtraction. These equations are given below ?

The difference (d) of the half subtractor is the XOR of A and B. Therefore,

$$\text{Difference, } d = A \oplus B = A'B + AB'$$

The borrow (b) of the half subtractor is the AND of A' (compliment of A) and B. Therefore,

$$\text{Borrow, } b = A'B$$

What is a Full-Subtractor?

A **full-subtractor** is a combinational circuit that has three inputs A, B, bin and two outputs d and b. Where, A is the minuend, B is subtrahend, bin is borrow produced by the previous stage, d is the difference output and b is the borrow output.

Since, the half subtractor can only be used to find the difference of LSBs (Least Significant Bits) of two binary numbers. Thus, if there is any borrow during the subtraction of the LSBs, it will affect the subtraction of the next bits of numbers. To overcome this problem of the half subtractor, a full subtractor is realized. The block diagram and circuit diagram of a fullsubtractor is shown in Figure-4.

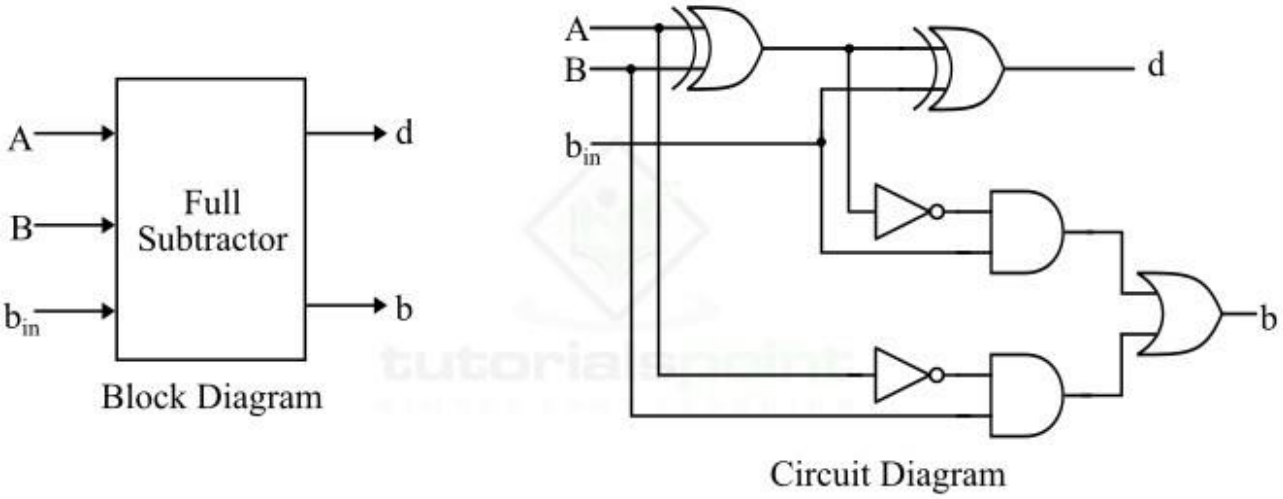


Figure 4 - Full Subtractor

Therefore, we can realize the full-subtractor using two XOR gates, two NOT gates, two AND gates, and one OR gate.

Truth Table of Full-Subtractor

The following is the truth table of the full-subtractor ?

Inputs			Outputs	
A	**B**	**b_{in}**	**d (Difference)**	**b (Borrow)**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

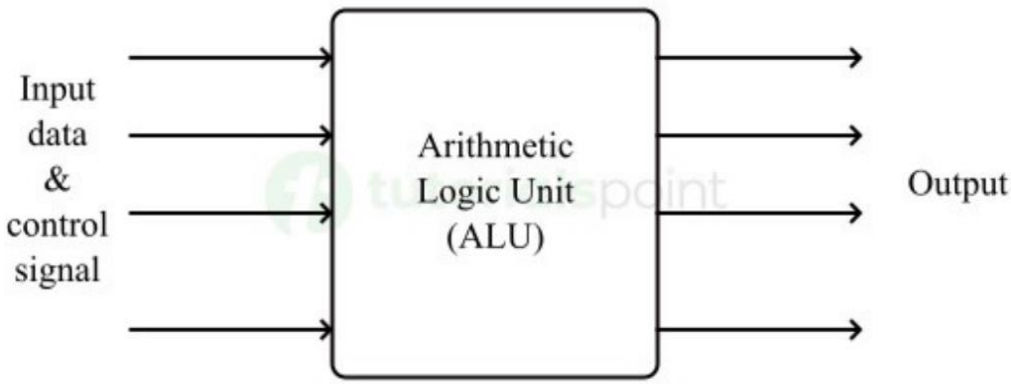
Arithmetic Logic Unit

The **Arithmetic Logic Unit (ALU)** is the fundamental component in a computing system like a computer. It is basically the actual data processing element within the central processing unit (CPU) in a computing system. It performs all the arithmetic and logical operations and forms the backbone of modern computer technology.

In this chapter, we will explain the working of the arithmetic logic unit, along with its main components, their functions, and the importance of the ALU in the field of digital system designs.

What is Arithmetic Logic Unit?

Arithmetic Logic Unit abbreviated as ALU is considered as the engine or heart of every central processing unit (CPU). ALU is basically a combination logic circuit that can perform arithmetic and logical operation on digital data (data in binary format). It can also execute instructions given to a computing system like a digital computer.



Within the complex architecture of a digital computing system, the arithmetic logic unit or ALU plays an important role as it executes and processes all the instructions, performs calculations, manipulates binary data, and performs various decision-making operations.

The development of arithmetic logic unit begins with the need for efficient, high speed, and accurate data processing and computation. With the advancement in electronics technologies, ALU has become a highly sophisticated digital data processing device that can handle a large number of complex instructions and computational tasks.

Todays ALU provides high accuracy, precision, and significantly fast processing speed in computing operations.

Features of Arithmetic Logic Unit

Here are some key features of arithmetic logic unit –

- The ALU can perform all arithmetic and logic operations such as addition,

subtraction, multiplication, division, logical comparisons, etc.

- It can also perform bitwise and mathematical operations on binary numbers.
- It contains two segments namely, AU (**arithmetic unit**) and LU (**logic unit**) to perform arithmetic operations and logical operations respectively.
- It is the computational powerhouse within a central processing unit (CPU).
- ALU is the part of every CPU where actual data processing takes place.
- ALU is responsible for interpreting the code instructions based on which operations to be performed on the input data.
- Once the data processing is completed, the ALU sends the outcomes to the memory unit or output unit.

Main Components of Arithmetic Logic Unit

The arithmetic logic unit consists of various functional parts that are responsible for performing specific operations like addition, subtraction, multiplication, division, comparison, and more. Some of the key components of the arithmetic logic unit are explained below −

Arithmetic Unit

The main components used in the arithmetic unit (AU) segment of the arithmetic logic unit are as follows −

Adder

The adder or binary adder is one of the important components of the arithmetic logic unit. It performs the addition of two or more binary numbers. To accomplish this operation, it performs a series of logical and arithmetic operations. Some common types of adders used in the arithmetic ogic unit are half-adder, full-adder, parallel adder, and ripple carry adder. Each type of adder is designed and optimized to perform a specific computing operation.

Subtractor

The subtractor is another digital combinational circuit designed to perform subtraction of binary numbers. In most arithmetic logic unit, the subtractor uses 2s complement arithmetic to perform subtraction on binary numbers.

Multiplier and Divider

In more complex and advanced arithmetic logic units, dedicated multiplier and divider circuits are also implemented to perform multiplication and division on binary numbers. These circuits use advanced processing techniques like iterative or parallel processing to

accomplish these operations.

Logic Unit

The logic unit (LU) of the ALU comprises the components responsible for performing Boolean or comparison operations. The following are some main components of the logic unit of an ALU –

Logic Gates

The logic gates like AND, OR, NOT, NAND, NOR, XOR, and XNOR are the key components of logic unit. These are standard logic circuits that can manipulate input data based on some predefined logical instructions and generate a desired output.

Each logic gate can perform a specific logical operation. However, different types of logic gates can be connected together in a specific manner to perform complex logical operations.

Type of Logic Gate

The brief overview of each type of logic gate is explained here –

- **AND Gate** – It performs the Boolean multiplication on input binary data. Its output is logic 1 or true, only when all its inputs are logic 1 or true.
- **OR Gate** – The OR gate performs the Boolean addition of input binary data. It generates a logic 1 or true output, if any of its inputs is logic 1 or true.
- **NOR Gate** – The NOT gate performs the inversion operation. It gives a logic 1 or true output when its input is logic 0 or false and vice-versa.
- **NAND Gate** – The NAND gate performs the NOTed AND operation and produces a logic 1 or true output when both inputs or any of the inputs is logic 0 or false.

  NOR Gate – The NOR gate performs the NOTed OR operation and generates a logic 1 or true output when all its inputs are logic 0 or false.

  XOR Gate – The XOR gate performs the exclusive OR operation and produces a logic 1 or true output when its both inputs are dissimilar. Hence, it is used as inequality detector.
- **XNOR Gate** – The XNOR gate performs the exclusive NOR operation and gives a logic 1 or true output when both its inputs are similar. Thus, it is used as an equality detector.

This is all about structure and components of the arithmetic logic unit. Let us now understand what functions an ALU can perform.

Functions of Arithmetic Logic Unit

The arithmetic logic unit can perform a wide range of functions and operations in digital computing systems. Some important functions that an arithmetic logic unit perform are explained below –

Arithmetic Operations

The arithmetic operations are one of the primary functions that the arithmetic logic unit performs. This category of operations includes addition, subtraction, multiplication, and division of binary numbers. All these operations form the basis of mathematical computations that the arithmetic logic unit can perform.

Logical Operations

The arithmetic logic unit can also perform various logical operations such as AND operation, OR operation, NOT operation, etc. These logical operations form the basis of decision making and data manipulation processes.

Comparison Operations

The arithmetic logic unit also facilitates to perform various comparison operations such as equal to, not equal to, less than, greater than, etc. These comparison operations are essential in decision making processes.

Shift Operations

The arithmetic logic unit can also perform shift operations on binary numbers such as left shift and right shift. These operations are important in multiplication and division operations. The shift operations can manipulate binary data at bit level and hence optimize the arithmetic calculations.

Working of Arithmetic Logic Unit

The working of the arithmetic logic unit depends on a combination of input data and control signals. In other words, the arithmetic logic unit receives the input data and control signals and then interpret these data and signals to perform specific operations.

Let us understand the working of the arithmetic logic unit in detail by breaking it down in sub-components.

Receiving Input Data and Control Signals

The arithmetic logic unit receives the input data from the user and a set of control

signals that specifies the operation to be performed. The data is received through the input data path while the control signals are received from the control unit.

Execution of Operation

Once the arithmetic logic unit received the input data and control signals, it selects an appropriate functional component among arithmetic unit, logic unit, comparison unit, or shift unit to perform the specific operation. Once the operation completes, the ALU sends the results to the memory unit for storage or output unit.

Significance of Arithmetic Logic Unit

In the field of digital electronics and computing technology, the arithmetic logic unit plays an important role because of the following reasons –

- It can perform the arithmetic, logical, and comparison operations with very high accuracy, precision, and efficiency.
- It can also perform complex data processing and decision-making operations.
- ALU can execute complex processing tasks at a very high speed that results in better performance and higher efficiency.

Binary Multiplication

In binary arithmetic, binary multiplication is the process of multiplying two binary numbers and obtain their product.

In binary multiplication, we multiply each bit of one binary number by each bit of another binary number and then add the partial products to obtain the final product.

Rules of Binary Multiplication

The multiplication of two binary numbers is performed as per the following rules of binary arithmetic –

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

It is clear that the binary multiplication is similar to the decimal multiplication. Let us

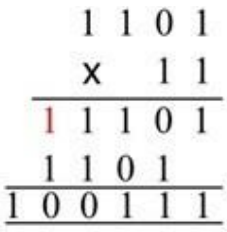
understand the binary multiplication with the help of solved examples.

Example 1

Multiply 1101 and 11.

Solution

The binary multiplication of given numbers is described below –



Explanation

Multiply the rightmost bit of the second number, 1 by each bit of the first number

Now, shift the partial product one position to the left to perform the next multiplication. (1101).

Multiply the leftmost bit of the second number, 1 by each bit of the first number (1101).

Finally, sum up all the partial products to obtain the final product.

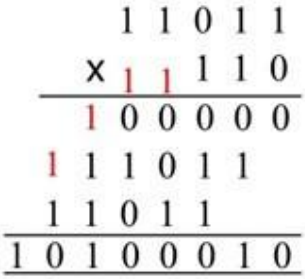
Hence, the product of 1101 and 11 is 100111.

Example 2

Multiply 11011 and 110.

Solution

The multiplication of given binary numbers is demonstrated below –



Explanation

Multiply rightmost bit of the second number (0) by each bit of the first binary number (11011). Shift the partial product one position to the left.

Multiply the second rightmost bit of the second number (1) by each bit of the first binary number (11011).

Again, shift the partial product one position to the left.

Multiply the leftmost bit of the second number (1) by each bit of the first number.

Then, sum up all the partial products to obtain the final product.

Hence, the product of 11011 and 110 is 10100010.

Binary Division

Binary division is one of the basic arithmetic operations used to find the quotient and remainder when dividing one binary number by another.

Rules of Binary Division

The following rules of binary arithmetic are utilized while diving one binary number by another –

$$0 \div 0 = \text{Undefined}$$

$$0 \div 1 = 0 \text{ with Remainder } = 0$$

$$1 \div 0 = \text{Undefined}$$

$$1 \div 1 = 1 \text{ with Remainder } = 0$$

Binary Division Procedure

- Start dividing from the leftmost bits of the dividend by the divisor.
- Multiply the quotient obtained by the divisor and subtract from the dividend.
- Bring down the next bits of the dividend and repeat the division process until all the bits of given divided are used.

Let us consider some solved examples to understand the binary division.

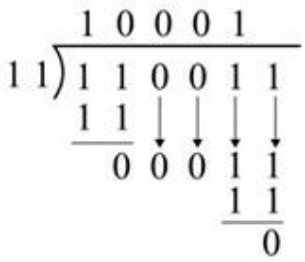
Example 1

Divide 110011 by 11.

Solution

The division of the given binary numbers is explained below −

110011 11 = 10001



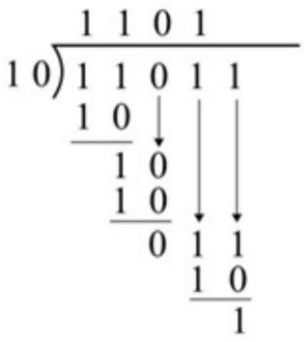
In this example of binary division, the quotient obtained is 10001 and the remainder is 0.

Example 2

Divide 11011 by 10.

Solution

The binary division of 11011 by 10 is explained below − 11011 10 = 1101



Arithmetic Circuits Using HDL

1. Introduction

- Arithmetic circuits are **combinational circuits** that perform mathematical operations.
- They are the backbone of processors' **Arithmetic Logic Unit (ALU)**.
- Common operations: **Addition, Subtraction, Multiplication, Division**.
- Designed using **HDL (Hardware Description Language)** → Verilog/VHDL.

---

2. Half Adder

Definition:

- A **Half Adder** adds two **single-bit binary numbers** (A and B).
- It has two outputs: **Sum (S)** and **Carry (Cout)**.

Truth Table:

A	B	Sum (S)	Carry (Cout)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Expressions:

- **Sum = A ⊕ B**
- **Carry = A · B**

Block Diagram (explained):

- Inputs: A, B

- XOR gate → gives Sum
- AND gate → gives Carry

HDL (Verilog):

```
module half_adder (input A, B, output Sum, Carry);
  assign Sum = A ^ B;
  assign Carry = A & B;
endmodule
```

---

3. Full Adder

Definition:

- A **Full Adder** adds **three inputs**: A, B, and Carry-in (Cin).
- Outputs: **Sum (S)** and **Carry-out (Cout)**.

Truth Table:

A	B	Cin	Sum	Cout
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

Expressions:

- **Sum = A ⊕ B ⊕ Cin**
- **Cout = (A · B) + (Cin · (A ⊕ B))**

Block Diagram:

- Two Half Adders + One OR gate.

HDL:

```
module full_adder (input A, B, Cin, output Sum, Carry);
  assign Sum = A ^ B ^ Cin;
  assign Carry = (A & B) | (Cin & (A ^ B));
endmodule
```

---

4. Ripple Carry Adder (RCA)

Definition:

- An **n-bit adder** made by cascading **n Full Adders**.
- Carry-out of one FA → Carry-in of the next.

Limitation:

- **Slow** due to carry propagation delay.

HDL (4-bit RCA):

```
module ripple_adder_4bit (input [3:0] A, B, input Cin, output [3:0] Sum, output Cout);
  wire C1, C2, C3;
  full_adder FA0 (A[0], B[0], Cin, Sum[0], C1);
  full_adder FA1 (A[1], B[1], C1,  Sum[1], C2);
  full_adder FA2 (A[2], B[2], C2,  Sum[2], C3);
  full_adder FA3 (A[3], B[3], C3,  Sum[3], Cout);
endmodule
```

---

5. Carry Look-Ahead Adder (CLA)

Concept:

- To overcome RCA delay, use **Generate (G)** and **Propagate (P)** signals.
- **Pi = Ai ⊕ Bi**, **Gi = Ai · Bi**
- Carry generated faster using logic equations.

Advantage:

- Much **faster** than RCA.

---

6. Subtractor Circuits

Half Subtractor:

- Inputs: A, B
- Outputs: **Difference (D), Borrow (Bout)**
- Equations:
  - **D = A ⊕ B**
  - **Bout = A' · B**

Full Subtractor:

- Inputs: A, B, Bin
- Outputs: Difference (D), Borrow-out (Bout)
- Equations:
  - **D = A ⊕ B ⊕ Bin**
  - **Bout = (A' · B) + (Bin · (A ⊕ B)')**

Using Adders (2's complement method):

- Subtraction A – B = A + (2's complement of B).

---

7. Multipliers

- **Array Multiplier**: shift-and-add method.
- **Booth's Multiplier**: efficient for signed binary multiplication.
- Implemented in HDL using loops and adders.

---

8. Dividers

- Implemented using **restoring / non-restoring division algorithms**.
- HDL often uses **iterative subtraction**.

---

9. Arithmetic Logic Unit (ALU)

Definition:

- ALU combines **arithmetic and logic operations**.
- Controlled by **select lines**.

Typical Operations:

- ADD, SUB, AND, OR, XOR, NOT, INC, DEC, SHIFT.

HDL (4-bit ALU):

```
module ALU (
  input [3:0] A, B,
  input [2:0] Sel,
  output reg [3:0] Result,
  output reg Carry
);
always @(*) begin
  case(Sel)
    3'b000: {Carry, Result} = A + B;  // Addition
    3'b001: {Carry, Result} = A - B;  // Subtraction
    3'b010: Result = A & B;           // AND
    3'b011: Result = A | B;           // OR
    3'b100: Result = A ^ B;           // XOR
    3'b101: Result = ~A;              // NOT
    3'b110: Result = A + 1;           // Increment
    3'b111: Result = A - 1;           // Decrement
  endcase
end
endmodule
```

---

10. Advantages of HDL for Arithmetic Circuits

- **Abstract design** (no gate-level wiring needed).
- **Reusable modules** (adders, subtractors).
- **Simulation & testing** possible before hardware.
- **Easily scalable** (4-bit → 8-bit → 32-bit).
- **Synthesis** on FPGA/ASIC.

Clock Waveforms

---

1. Introduction

- A **clock signal** is a special digital waveform used to **synchronize** operations in sequential circuits (flip-flops, counters, registers, processors).
- Acts as a **timing reference** for when data should be sampled, stored, or transferred.
- Clock signals are usually **periodic square waves** alternating between logic HIGH (1) and logic LOW (0).

---

2. Characteristics of Clock Waveforms

1. **Period (T):**
   - Time taken for one complete cycle (High + Low).
   - Measured in seconds.
2. **Frequency (f):**
   - Number of cycles per second.
   - Formula:
     [
     f = \frac{1}{T}
     ]
   - Measured in Hertz (Hz).
3. **Duty Cycle (D):**
   - Percentage of time clock stays HIGH in one cycle.
   - Formula:
     [
     D = \frac{T_{high}}{T} \times 100%
     ]
4. **Rise Time (tr):**
   - Time taken for the clock to rise from logic LOW (0) to logic HIGH (1).
5. **Fall Time (tf):**
   - Time taken for the clock to fall from HIGH (1) to LOW (0).
6. **Clock Skew:**
   - The time difference when the same clock signal reaches different parts of a circuit.

---

3. Types of Clock Waveforms

1) Square Wave Clock

- Most common clock signal.
- HIGH and LOW times are equal (50% duty cycle).
- Used in synchronous digital systems.

---

2) Rectangular Clock (Variable Duty Cycle)

- HIGH and LOW times are not equal.
- Duty cycle may be >50% or <50%.

---

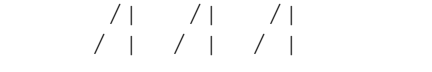
3) Pulse Clock

- A very short HIGH pulse followed by long LOW time.
- Used in triggering flip-flops or control signals.

---

4) Triangular / Sawtooth Waveform (Analog Clocks)

- Rarely used in digital logic but common in communication systems.
- Slope increases and decreases linearly.



---

4. Clock in Sequential Circuits

- **Edge Triggering:**
  - Flip-flops respond to **edges** of clock:
    - **Positive Edge (↑):** Transition from LOW → HIGH.
    - **Negative Edge (↓):** Transition from HIGH → LOW.
- **Level Triggering:**
  - Flip-flops respond while clock is HIGH or LOW (not just at edges).

---

5. HDL Representation of Clock

In **Verilog HDL**, clock signals are often generated in testbenches for simulation.

Example (10 ns period clock):

```
module testbench;
  reg clk;

  initial begin
    clk = 0;
    forever #5 clk = ~clk; // toggles every 5ns → period = 10ns
  end
endmodule
```

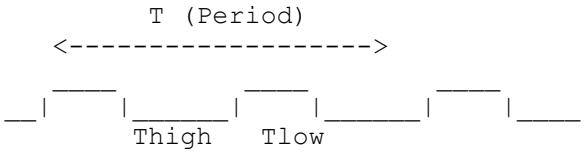
---

6. Applications of Clock Waveforms

- Synchronization in **processors** (CPU clock).
- Driving **counters, registers, flip-flops**.
- Controlling **timing of data transfer** in buses.

- Used in **communication protocols** (I²C, SPI, UART).

1. Clock with Period & Frequency



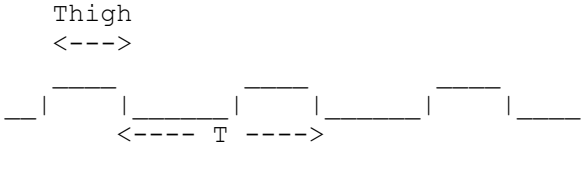

```
f = 1/T
```

- **T = Thigh + Tlow**
- **f = 1/T**

---

2. Duty Cycle Illustration

```
Duty Cycle (D) = (Thigh / T) × 100%
```

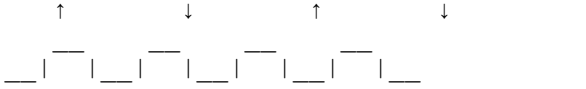


- **Thigh** = Time clock is HIGH
- **T** = Total period

---

3. Positive and Negative Edge

```
Positive Edge (↑) → LOW to HIGH
Negative Edge (↓) → HIGH to LOW
```



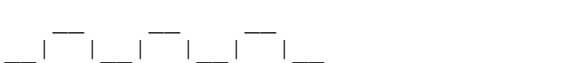
---

4. Level Triggering

```
HIGH-Level Triggering
```

```
Triggered while clock = HIGH


LOW-Level Triggering
```



```
Triggered while clock = LOW
```

---

5. Types of Clock Waveforms

(a) Square Wave (50% Duty Cycle)

(b) Rectangular Wave (≠ 50% Duty Cycle)

(c) Pulse Clock

TTL Clock

1. Introduction to Clock in Digital Logic

- A **clock** is a signal that oscillates between HIGH and LOW states at regular intervals.
- It provides the **timing reference** for synchronous digital systems.
- Without a clock, flip-flops, counters, and sequential circuits cannot function correctly.

---

2. TTL (Transistor–Transistor Logic) Basics

- **TTL Logic** is a type of digital circuit built using **bipolar junction transistors (BJTs)**.
- It uses a supply voltage of **+5V (standard)**.
- Logic levels in TTL:
  - **Logic HIGH (1)** ≈ 2V to 5V (typically 5V)
  - **Logic LOW (0)** ≈ 0V to 0.8V (typically 0V)
- TTL is widely used in digital IC families like **74-series ICs**.

---

3. TTL Clock Signal

- A **TTL Clock** is a square waveform that alternates between 0V (LOW) and +5V (HIGH).
- It has two important parameters:
  - **Time Period (T):** Duration of one complete cycle (HIGH + LOW).
  - **Frequency (f):** Number of cycles per second. (f = \dfrac{1}{T})
  - **Duty Cycle (D):** Ratio of HIGH time to the total time period, expressed as a percentage.

Example:

- If the HIGH period = 5 ms and LOW period = 5 ms → Total T = 10 ms
- Frequency = 100 Hz
- Duty cycle = 50%

---

4. Generation of TTL Clock

TTL clock signals can be generated using:

1. **Crystal Oscillator** – provides a stable frequency.
2. **555 Timer IC in Astable Mode** – generates square wave pulses.
3. **Ring Oscillator or Logic Gates** – simple clock generation.
4. **Microcontrollers / Microprocessors** – built-in clock generators.

---

5. Characteristics of TTL Clock

- Operates on **5V supply**.
- Produces a **clean square waveform**.
- Provides **synchronization** for sequential circuits.
- Frequency can range from a few Hz to several MHz depending on the design.
- Compatible with TTL logic circuits (74xx family).

---

6. Applications of TTL Clock in Digital Logic

- **Flip-Flops:** Edge-triggered devices need clock pulses for storing data.
- **Counters:** Count pulses according to clock cycles.
- **Registers:** Shift or store data with each clock pulse.
- **Microprocessors & Microcontrollers:** Require clock for instruction execution.
- **Timers & Control Circuits:** Use clock for scheduling events.

Summary

- A **TTL Clock** is a periodic square waveform with voltage levels defined by TTL logic (0V = LOW, 5V = HIGH).
- It is the **heartbeat** of digital systems, ensuring all components work in synchrony.
- Essential in **flip-flops, counters, registers, microprocessors, and timing circuits**.

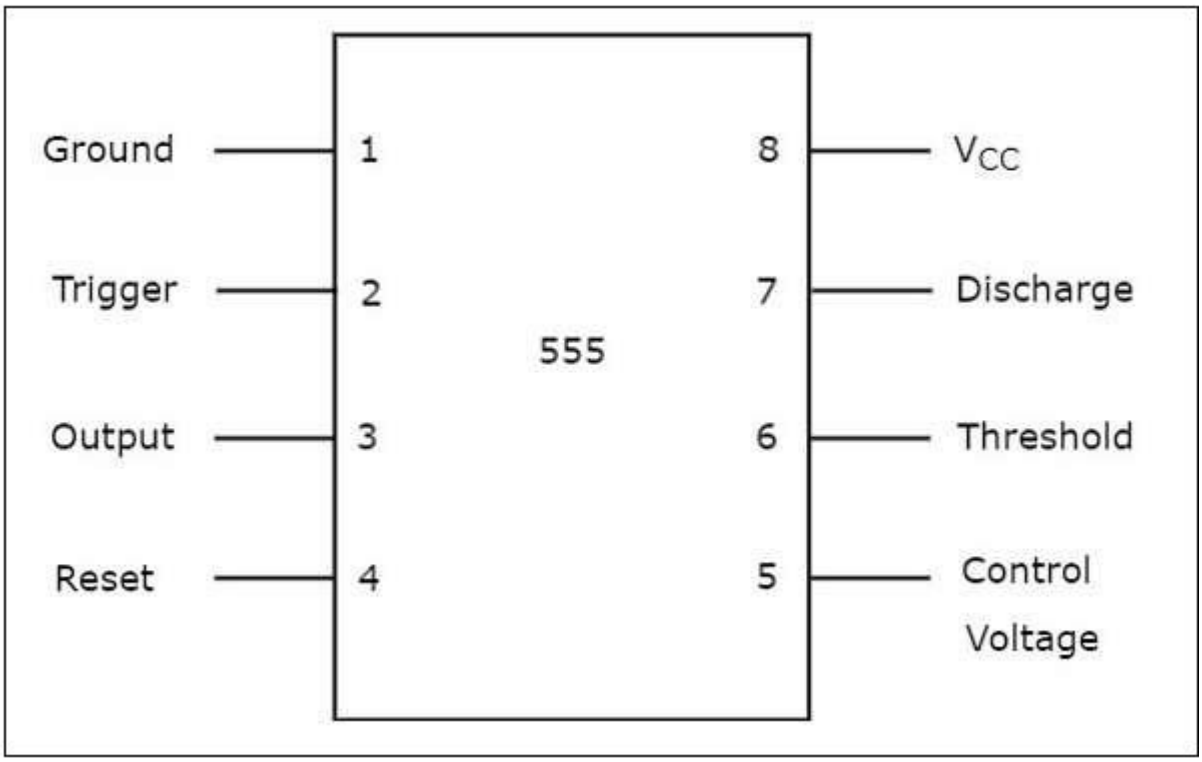
The **555 Timer** IC got its name from the three $5K\Omega$ resistors that are used in its voltage divider network. This IC is useful for generating accurate time delays and oscillations. This chapter explains about 555 Timer in detail.

Pin Diagram and Functional Diagram

In this section, first let us discuss about the pin diagram of 555 Timer IC and then its functional diagram.

Pin Diagram

The 555 Timer IC is an 8 pin mini Dual-Inline Package (DIP). The **pin diagram** of a 555 Timer IC is shown in the following figure −

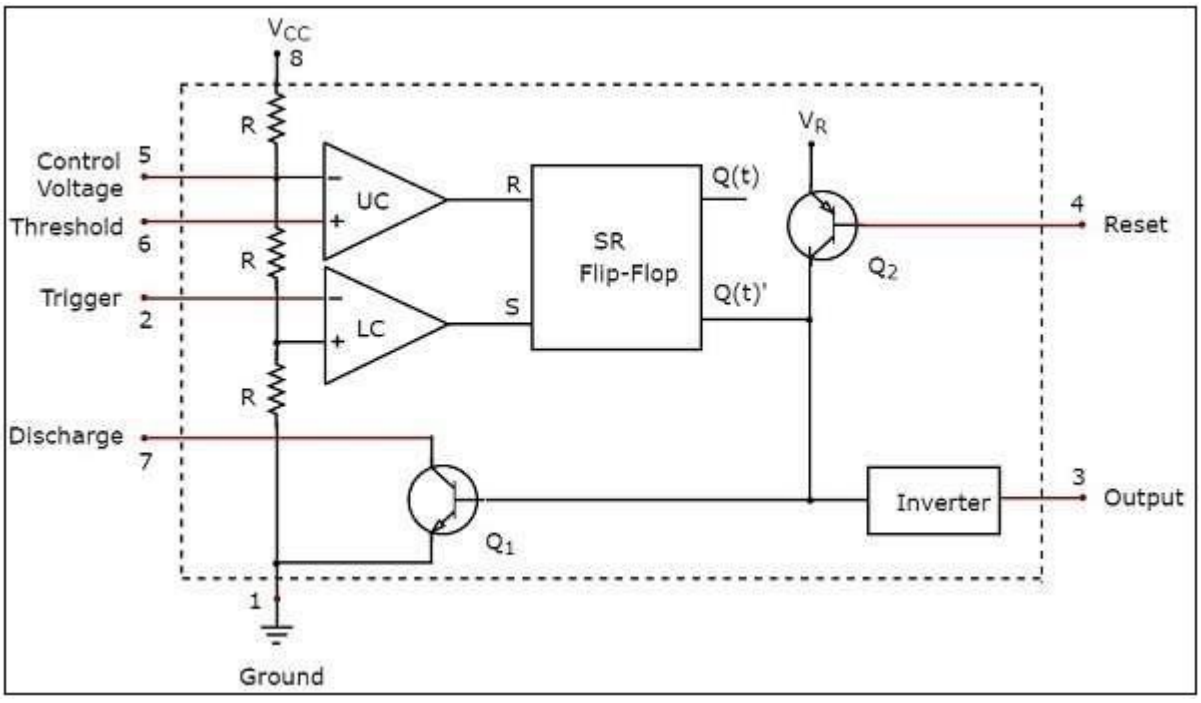


The significance of each pin is self-explanatory from the above diagram. This 555 Timer IC can be operated with a DC supply of +5V to +18V. It is mainly useful for generating **non-sinusoidal** wave forms like square, ramp, pulse & etc

Functional Diagram

The pictorial representation showing the internal details of a 555 Timer is known as functional diagram.

The **functional diagram** of 555 Timer IC is shown in the following figure −



Observe that the functional diagram of 555 Timer contains a voltage divider network, two comparators, one SR flip-flop, two transistors and an inverter. This section discusses about the purpose of each block or component in detail −

Voltage Divider Network

- The voltage divider network consists of a three $5K\Omega$ resistors that are connected in series between the supply voltage V_{cc} and ground.
- This network provides a voltage of $V_{cc}3$ between a point and ground, if there exists only one $5K\Omega$ resistor. Similarly, it provides a voltage of $2V_{cc}3$ between a point and ground, if there exists only two $5K\Omega$ resistors.

Comparator

- The functional diagram of a 555 Timer IC consists of two comparators: an Upper Comparator (UC) and a Lower Comparator (LC).
- Recall that a **comparator** compares the two inputs that are applied to it and produces an output.
- If the voltage present at the non-inverting terminal of an op-amp is greater than the voltage present at its inverting terminal, then the output of comparator will be $+V_{sat}$. This can be considered as **Logic High** ('1') in digital representation.
- If the voltage present at the non-inverting terminal of op-amp is less than or equal to the voltage at its inverting terminal, then the output of comparator will be $-V_{sat}$. This can be considered as **Logic Low** ('0') in digital representation.

SR Flip-Flop

- Recall that a **SR flip-flop** operates with either positive clock transitions or negative clock transitions. It has two inputs: S and R, and two outputs: Q(t) and Q(t). The outputs, Q(t) & Q(t) are complement to each other.
- The following table shows the **state table** of a SR flip-flop

S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
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 (negative) transition of clock signal is applied.
- The outputs of Lower Comparator (LC) and Upper Comparator (UC) are applied as **inputs of SR flip-flop** as shown in the functional diagram of 555 Timer IC.

Transistors and Inverter

- The functional diagram of a 555 Timer IC consists of one npn transistor Q_1 and one pnp transistor Q_2. The npn transistor Q_1 will be turned ON if its base to emitter voltage is positive and greater than cut-in voltage. Otherwise, it will be turned-OFF.
- The pnp transistor Q_2 is used as **buffer** in order to isolate the reset input from SR flip-flop and npn transistor Q_1.

- The **inverter** used in the functional diagram of a 555 Timer IC not only performs the inverting action but also amplifies the power level.

The 555 Timer IC can be used in mono stable operation in order to produce a pulse at the output. Similarly, it can be used in astable operation in order to produce a square wave at the output.

A Stable

1. Definition

- An **Astable Multivibrator** is a **free-running oscillator** that continuously switches between two states (HIGH and LOW) without any external trigger.
- It does **not have a stable state**, hence called "astable."
- Produces a **continuous square wave output** used as a clock pulse in digital circuits.

---

2. Working Principle

- The circuit uses **capacitor charging and discharging** to toggle output between HIGH and LOW.
- When the capacitor voltage reaches the **upper threshold**, output goes LOW.
- When the capacitor voltage reaches the **lower threshold**, output goes HIGH.
- This cycle repeats indefinitely, generating a square wave.

---

3. Astable 555 Timer Circuit

- The **555 Timer IC** is widely used to build an astable multivibrator.
- **Connections**:
  - **R1** → between Vcc and pin 7 (Discharge)
  - **R2** → between pin 7 (Discharge) and pin 6/2 (Threshold/Trigger)
  - **C** → between pin 6/2 and ground
- **Pin 3** gives the **square wave output**.

---

4. Timing Calculations

Let:

- **R1, R2** = Resistors in ohms (Ω)
- **C** = Capacitor in farads (F)
- **High Time (T_H):**
  [
  T_H = 0.693 \times (R1 + R2) \times C
  ]
- **Low Time (T_L):**
  [
  T_L = 0.693 \times R2 \times C
  ]

- **Total Period (T):**
  [
  T = T_H + T_L = 0.693 \times (R1 + 2R2) \times C
  ]
- **Frequency (f):**
  [
  f = \dfrac{1}{T} = \dfrac{1.44}{(R1 + 2R2) \times C}
  ]
- **Duty Cycle (D):**
  [
  D = \dfrac{T_H}{T} \times 100 = \dfrac{R1 + R2}{R1 + 2R2} \times 100\%
  ]

Note: Duty cycle is usually greater than 50% unless modified using diodes.

---

5. Characteristics of Astable Multivibrator

- No stable state → output continuously oscillates.
- Produces a square waveform.
- Frequency and duty cycle depend on **R1, R2, and C**.
- Can drive TTL and CMOS logic circuits.

---

6. Applications

- **Clock pulse generator** for digital circuits.
- **LED flashers and lamp blinkers.**
- **Tone generators and alarms.**
- **PWM (Pulse Width Modulation) circuits.**
- **Timing and waveform generation in microcontrollers.**

✅ **Summary**

- An **Astable Multivibrator** produces continuous square waves without any external trigger.
- When implemented with a **555 Timer**, it's a versatile clock generator used in digital logic applications.
- Its **frequency and duty cycle** can be easily adjusted using resistors and capacitors.

UNIT IV

RS flip-flop

The RS flip-flop is used to store binary information (i.e. 0 or 1). It consists of two inputs, SET and RESET. In RS flip-flop 'R' Stands for RESET and 'S' stands for SET. The flip-flop keeps its present state even when one or both inputs are deactivated. The flip-flop enters the '0' state when the RESET input is activated, and the '1' state when the SET input is activated.

RS Flip-Flop block diagram

The block diagram of the RS flip-flop is shown above. Since RS flip-flops are used for storing binary information it is used in simple digital applications like data registers and memory cells that require binary storage. Nevertheless, more sophisticated flip-flop designs, such as the D flip-flop or JK flip-flop, are frequently used over the RS flip-flop for their increased reliability and versatility in complex digital systems due to restrictions in handling specific input conditions.

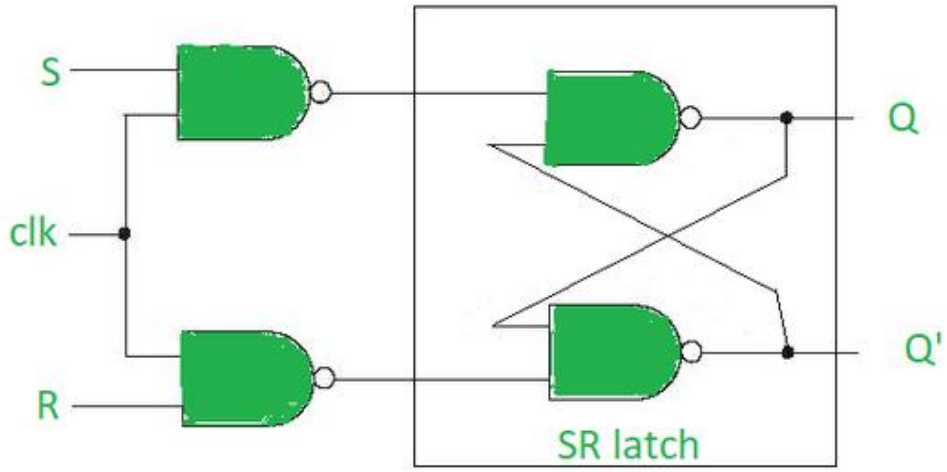
Construction and Working of RS flip flop

RS flip can be constructed using basic logic gates such as NAND gates or NOR gates. Below shown a RS flip-flop constructed using a NAND gate same we can construct a RS flip flop using a NOR gate.

Working of RS flip-flop depends on its inputs.

- In initial state output can in any state SET or RESET.
- In SET , S=1 and R=0.
- Conversely in RESET , S=0 and R=1.
- When R=0 and S=0 it hold the current state
- When R=1 and S=1 it falls under Undefined / Forbidden state

RS flip-flop Using NAND gates



RS flip-flop

The RS flip-flop can be constructed using 4 two input NAND gate which is shown in the above figure.

Truth Table For RS Flip-flop

Sl no.	Clock	S	R	Qn+1
1	0	X	X	Qn
2	1	0	0	No change
3	1	0	1	0
4	1	1	0	1
5	1	1	1	Undefined / Forbidden state

Here, **S** is the Set input, **R** is the reset input,**Qn+1** is the next state and **State** tells in which state it enters

Characteristic Table For RS Flip-flop

The characteristic equation tells us about what will be the next state of flip flop in terms of present state.

Sl no.	Qn	S	R	Qn+1
1	0	0	0	0

Sl no.	Qn	S	R	Qn+1
2	0	0	1	0
3	0	1	0	1
4	0	1	1	X
5	1	0	0	1
6	1	0	1	0
7	1	1	0	1
8	1	1	1	X

Characterstic Equation : Qn+1 = S + Qn . R'

Here, **S** is the Set input, **R** is the reset input, **Qn** is the current state input and **Qn+1** is the next state outputs.

Excitation table For RS Flip-flop

Excitation Table basically tells about the excitation which is required by flip flop to go from current state to next state.

Sl no.	Qn	Qn+1	S	R
1	0	0	0	X
2	0	1	1	0
3	1	0	0	1
4	1	1	X	0

Here, **Qn** is the current state, **Qn+1** is the next state outputs and **S**, **R** are the set and reset inputs respectively.

Applications of RS Flip-flop

RS flipflop are data storage device used to store binary information. It is used is mainly devices requires binary information. Its used in:

1. Asynchronous Counters (RS flipflops are used in building the asynchronous counters).

2. Shift Registers (RS flip-flops can be used to build shift registers).
3. State Machines.
4. Debouncing Circuit (used in stabilizing output of a switch/button).
5. Address Decoding.
6. Clock Synchronization.

Edge-Triggered Flip-Flop

The type of digital circuit which is capable of storing 1-bit of information and responds only when a specific edge of the clock pulse occurs is known as an **edge-triggered flip-flop**.

Therefore, the output state of the edge-triggered flip flop updates only when a specific edge of the clock pulse occurs, i.e. the clock pulse goes from either low to high or high to low states. This flip flop does not respond to a continuous clock pulse.

Edge-triggered flip-flop are used in several digital circuits where the output of the flip flop should be updated when the clock pulse changes its state from 0 to 1 or 1 to 0 as shown in Figure-2.

Figure 2 - Clock Pulse Changes State from 0 to 1 or 1 to 0

Hence, the edge triggered flip flop operates only on the rising or falling edge of the clock pulse.

Types of Edge-Triggered Flip-Flop

Depending on the responding edge of the clock pulse, the edge-triggered flip flops are classified into two types namely,

- Positive Edge-Triggered Flip-Flop
- Negative Edge-Triggered Flip-Flop

Positive Edge-Triggered Flip-Flop

The type of edge-triggered flip-flop whose output changes its state only on the rising edge (edge that goes from low to high) of the clock pulse is called a **positive edge-triggered flip-flop**. The positive edge triggered flip flop is also called a **rising edge-triggered flip-flop**. The block diagram of a positive edge triggered flip flop is shown in Figure-3 below.

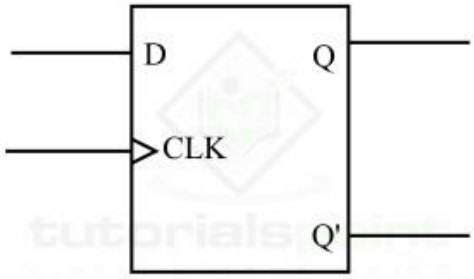


Figure 3 - Positive Edge-Triggered Flip-Flop

In a positive edge triggered flip flop, the inputs are accepted and stored only when the clock pulse goes from low (0) to high (1), i.e. on the rising edge of the clock pulse. This stored value is then available on the outputs.

Negative Edge-Triggered Flip-Flop

The type of edge-triggered flip flop whose output changes its state only on the falling edge (edge that goes from high to low) of the clock pulse is called **a negative edge-triggered flip-flop**. The negative edge triggered flip flop is also known as a **falling edge-triggered flip-flop**. The block diagram of a negative edge triggered flip flop is shown in Figure-4 below.

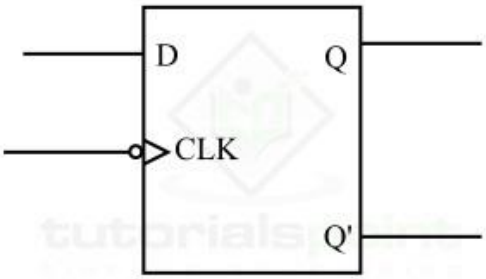


Figure 4 - Negative Edge-Triggered Flip-Flop

In the case of negative edge triggered flip flop, the flip-flop captures and stores the inputs only when the clock pulse goes from high to low, i.e. on falling edge of the clock pulse.

Operation of Edge-Triggered Flip-Flop

The operation of a typical edge-triggered flip-flop is described below

In the edge-triggered flip-flop, the inputs are applied through the input terminals and a clock pulse is connected to the clock input of the flip-flop. The edge triggered flip flop responds according to the applied inputs when the clock pulse goes from either low to high or high to low. When this state transition of clock pulse occurs, the flip-flop captures and stores the input values. These stored input values will be then available on the outputs (Q and Q') of the flip-flop.

Advantages of Edge-Triggered Flip-Flop

The important advantages of the edge-triggered flip flop are listed as follows

- Edge triggered flip flops have an improved timing behavior as compared to the level triggered flip flops. This is because, the edge triggered flip flop responds only on the transition of clock pulse.
- Edge-triggered flip flops reduces the possibility of glitches that cause errors in the system.
- Edge triggered flip flops consumes relative low power than the level-triggered flip flops.
- Edge-triggered flip flop has relatively less complex circuit design.

- Edge triggered flip flops can be easily integrated in the form of digital ICs.
- Edge triggered flip flops are very useful in digital systems having very high clock speed.

Applications of Edge-Triggered Flip-Flop

Edge triggered flip flops are used in a variety of digital systems. Some common applications of edge triggered flip flops are −

- Edge triggered flip-flops are used in registers to store and transfer binary information among different parts of a digital circuit.
- Edge triggered flip flops are used in digital counters for generating sequences of binary values.
- Edge triggered flip flops are also used for digital signal processing.
- Edge triggered flip flops are used in several digital applications where timing and signal synchronization is required.
- Edge-triggered flip flops are also used to build memory cells of ROM, RAM, etc. to stores binary data in a digital system.

Edge Triggered D Flip Flops

Like in D latch, in D Flip-Flop the basic SR Flip-Flop is used with complemented inputs. The D Flip-Flop is similar to D-latch except clock pulse is used instead of enable input.

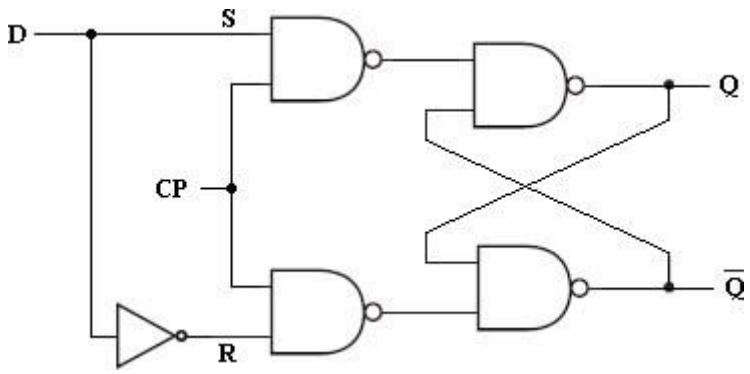


Fig : 3.18 - D Flip-Flop

To eliminate the undesirable condition of the indeterminate state in the RS Flip-Flop is to ensure that inputs S and R are never equal to 1 at the same time. This is done by D Flip-Flop. The D (***delay***) Flip-Flop has one input called delay input and clock pulse input. The D Flip-Flop using SR Flip-Flop is shown below.

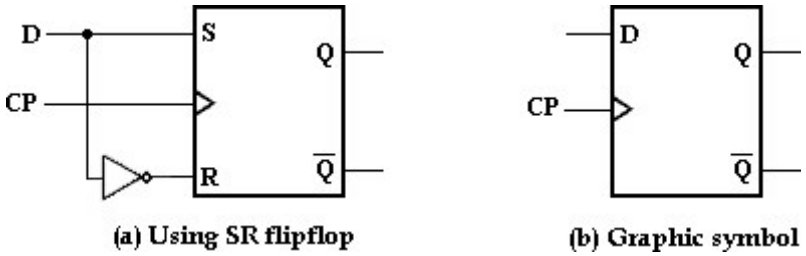


Fig : 3.19 – D Flipflop using SR Flipflop The truth table of D Flip-Flop is given below.

Clock	D	Qn+1	State
1	0	0	Reset
1	1	1	Set
0	x	Qn	No Change

Truth table for D Flip-Flop

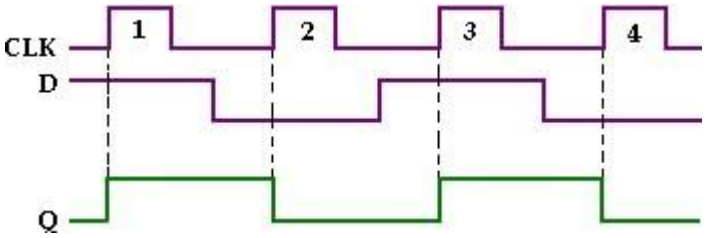


Fig : 3.20 - Input and output waveforms of clocked D Flip-Flop

Looking at the truth table for D Flip-Flop we can realize that Q_{n+1} function follows the D input at the positive going edges of the clock pulses.

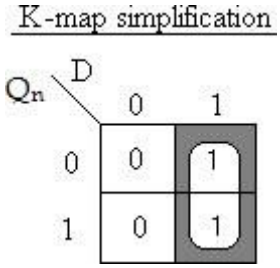
Characteristic table and Characteristic equation:

The characteristic table for D Flip-Flop shows that the next state of the Flip- Flop is

independent of the present state since Q_{n+1} is equal to D. This means that an input pulse will transfer the value of input D into the output of the Flip-Flop independent of the value of the output before the pulse was applied. The characteristic equation is derived from K-map.

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

Characteristic table



Characteristic equation: $\mathbf{Q_{n+1} = D.}$

Edge Triggered JK Flip Flops

JK means Jack Kilby, Texas Instrument (TI) Engineer, who invented IC in 1958. JK Flip-Flop has two inputs J(set) and K(reset). A JK Flip-Flop can be obtained from the clocked SR Flip-Flop by augmenting two AND gates as shown below.

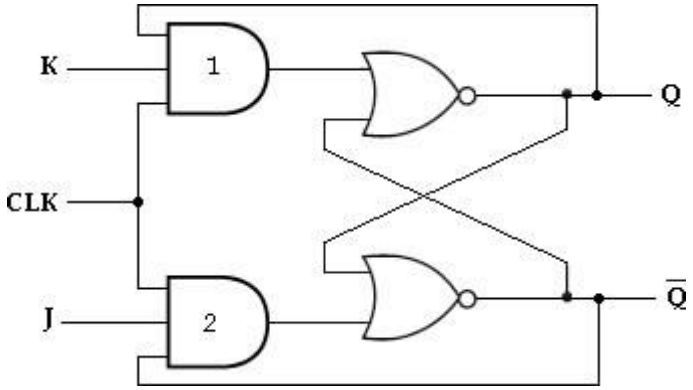


Fig : 3.15 - JK Flip Flop

The data input J and the output Q' are applied o the first AND gate and its output (JQ') is applied to the S input of SR Flip-Flop. Similarly, the data input K and the output Q are applied to the second AND gate and its output (KQ) is applied to the R input of SR Flip-Flop.

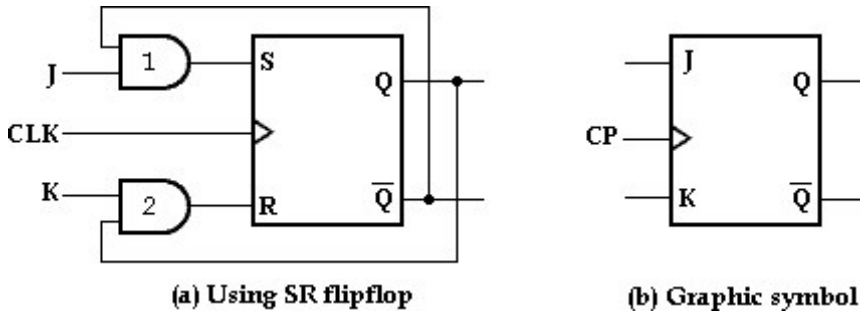


Fig : 3.16 – JK Flipflop using SR Flipflop

J= K= 0

When J=K= 0, both AND gates are disabled. Therefore clock pulse have no effect, hence the Flip-Flop output is same as the previous output.

J= 0, K= 1

When J= 0 and K= 1, AND gate 1 is disabled i.e., S= 0 and R= 1. This condition will reset the Flip-Flop to 0.

J= 1, K= 0

When J= 1 and K= 0, AND gate 2 is disabled i.e., S= 1 and R= 0. Therefore the Flip-Flop will set on the application of a clock pulse.

J= K= 0

When J=K= 1, it is possible to set or reset the Flip-Flop. If Q is High, AND gate 2 passes on a reset pulse to the next clock. When Q is low, AND gate 1 passes on a set pulse to the next clock. Eitherway, Q changes to the complement of the last state i.e., toggle. Toggle means to switch to the opposite state.

The truth table of JK Flip-Flop is given below.

CLK	**Inputs**		**Output**	**State**
	J	**K**	**Qn+1**	
1	0	0	Qn	No Change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	Q_n'	Toggle

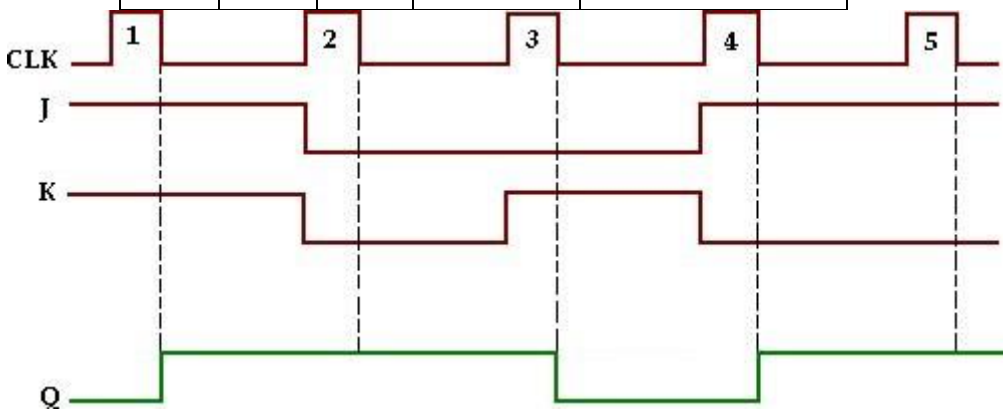


Fig : 3.17 - Input and output waveforms of JK Flip-Flop

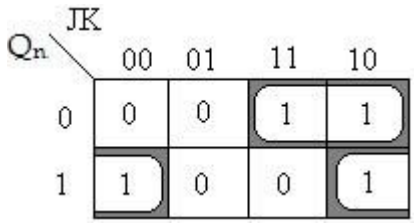
Characteristic table and Characteristic equation:

The characteristic table for JK Flip-Flop is shown in the table below. From the table, K-map for the next state transition (Q_{n+1}) can be drawn and the simplified logic expression which represents the characteristic equation of JK Flip-Flop can be found.

Qn	**J**	**K**	**Qn+1**
0	0	0	0
0	0	1	0
0	1	0	1

0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

K-map Simplification:



Characteristic equation: $\mathbf{Q_{n+1} = JQ' + K'Q.}$

JK Master Slave Flip Flops

A master-slave Flip-Flop is constructed using two separate JK Flip-Flops. The first Flip-Flop is called the master. It is driven by the positive edge of the clock pulse. The second Flip-Flop is called the slave. It is driven by the negative edge of the clock pulse. The logic diagram of a master-slave JK Flip-Flop is shown below.

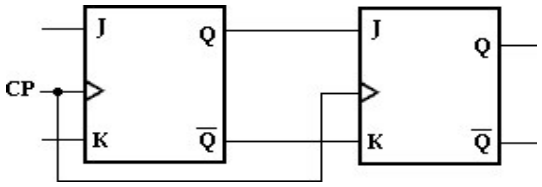


Fig : 3.32 - Logic diagram

When the clock pulse has a positive edge, the master acts according to its J- K inputs, but the slave does not respond, since it requires a negative edge at the clock input.

When the clock input has a negative edge, the slave Flip-Flop copies the master outputs. But the master does not respond since it requires a positive edge at its clock input. The clocked master-slave J-K Flip-Flop using NAND gates is shown below.

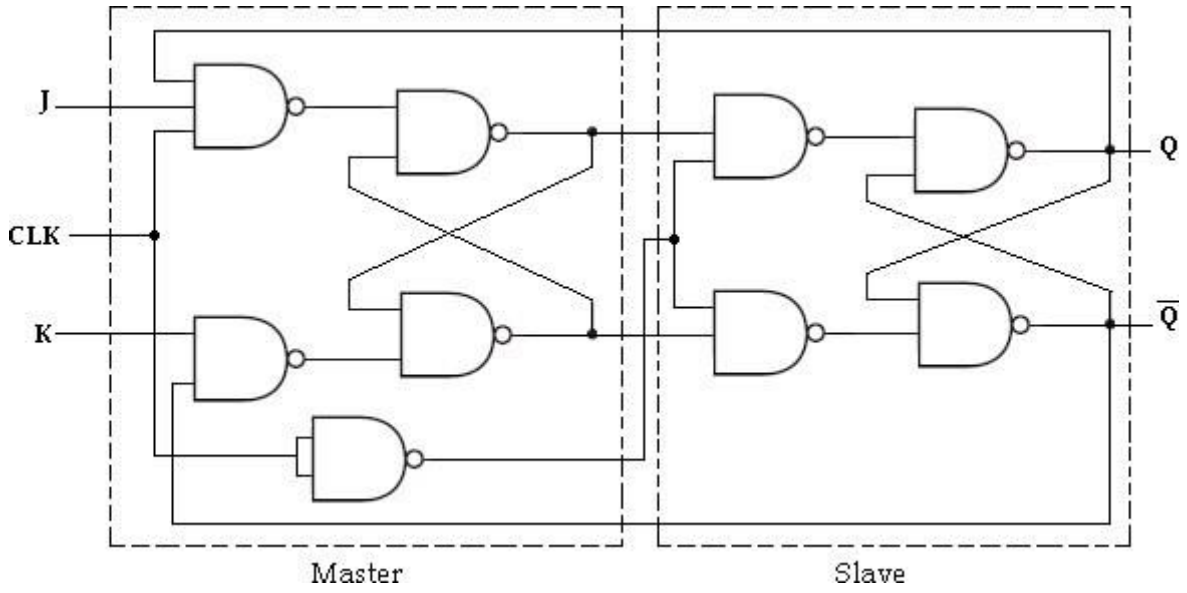


Fig : 3.33 - Master-Slave JK Flip-Flop

Register

A single flip-flop is able to store single bit information either 0 or 1, but to store more than one bit information, a group of flip-flops need to be connected. A group of flip-flops is called a register. If a register contains n flip-flops, it is able to store n bit information. Registers can be used to generate the specified sequence and can also be used to shift the content of flip-flop position wise, based on this the applications of registers are classified into two categories.

1. Shift registers and
2. Counters

1. Shift registers

A shift register is an entity of flip-flops, which are capable of shifting the state of flip-flop positionwise in one direction or two directions.

Example: To store 4-bit data 1001, four flip-flops are used.

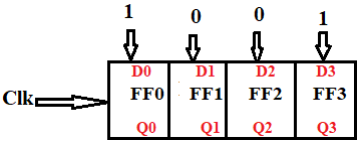


Figure 32: Basic shift register

Based on the direction of shifting the content of flip-flop, shift registers are classified into two types.

a) Unidirectional shift registers

The unidirectional shift registers, shifts the contents of flip-flops in only one direction

either right shift or left shift.

b) Bideirectional shift registers

The Bidirectional shift registers are capable of performing right shift as well as left shift through a proper control signal.

c) Universal shift register

The universal shift registers are capable of performing right shift as well as left shift through a proper control signal along with parallel loading and memory.

Shift registers further classified into four types, based on the way the input is loaded and the output is received.

a) Serial input and serial ouptut

The information will be loaded serially through a single line and output will be received serially through a single line is called serial input and serial output (SISO) shift register shown in figure (33).

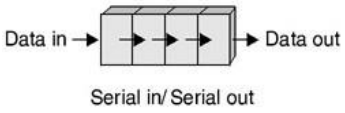


Figure 33: SISO Shift register

b) Serial input and parallel output

the information will be loaded serially through a single line and output will be received in parallel through multiple lines is called serial input parallel output (SIPO) shift register shown in figure (34).

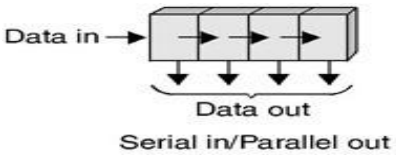


Figure 34: SIPO Shift register

c) Parallel input and serial output

The information bits will be loaded in parallel through multiple lines and output will be received through a single line is called parallel input serial output (PISO) shift register shown in figure (35).

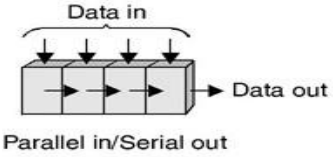


Figure 35: SISO Shift register

d) Parallel input and parallel output

The information bits will be loaded in parallel through multiple lines and output will be taken in parallel with multiple output lines is called parallel input parallel output (PIPO) shift register shown in figure (36).

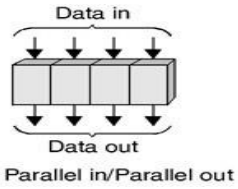


Figure 36: SISO Shift register

Bidirectional Shift register

Performs both left shift and right shift

M=0 (Shift left operation), M=1 (Shift right operation)

Logic diagram

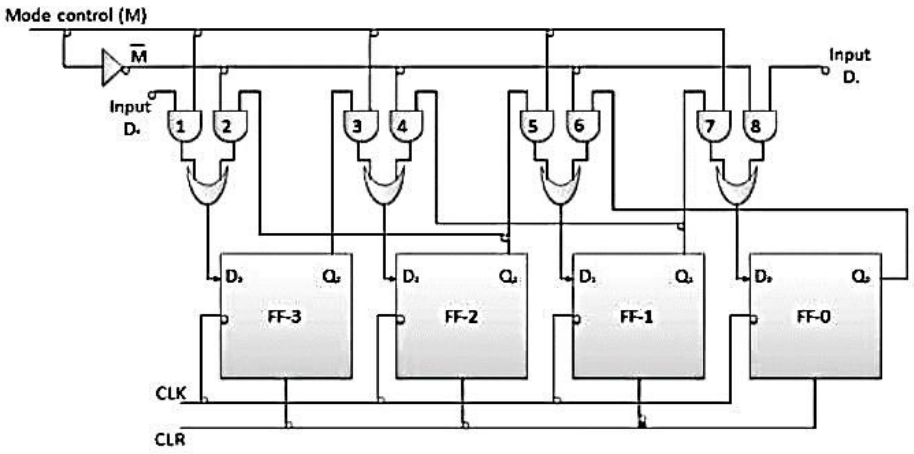


Figure 46: Logic diagram of bidirectional shift registers

Universal Shift Register

Bidirectional Shift Register + (SISO, SIPO, PIPO, PISO)+Memory

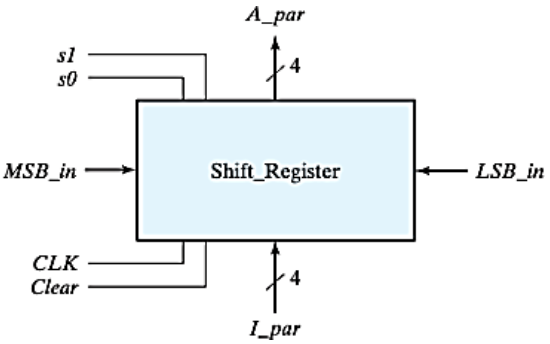


Figure 47: Logic symbol of universal shift register

S1	**S0**	**Register Operation**
0	0	No Change (Memory)
0	1	Shift left
1	0	Shift right
1	1	Parallel loading

Table 28: Universal shift opearions table

Logic Diagram

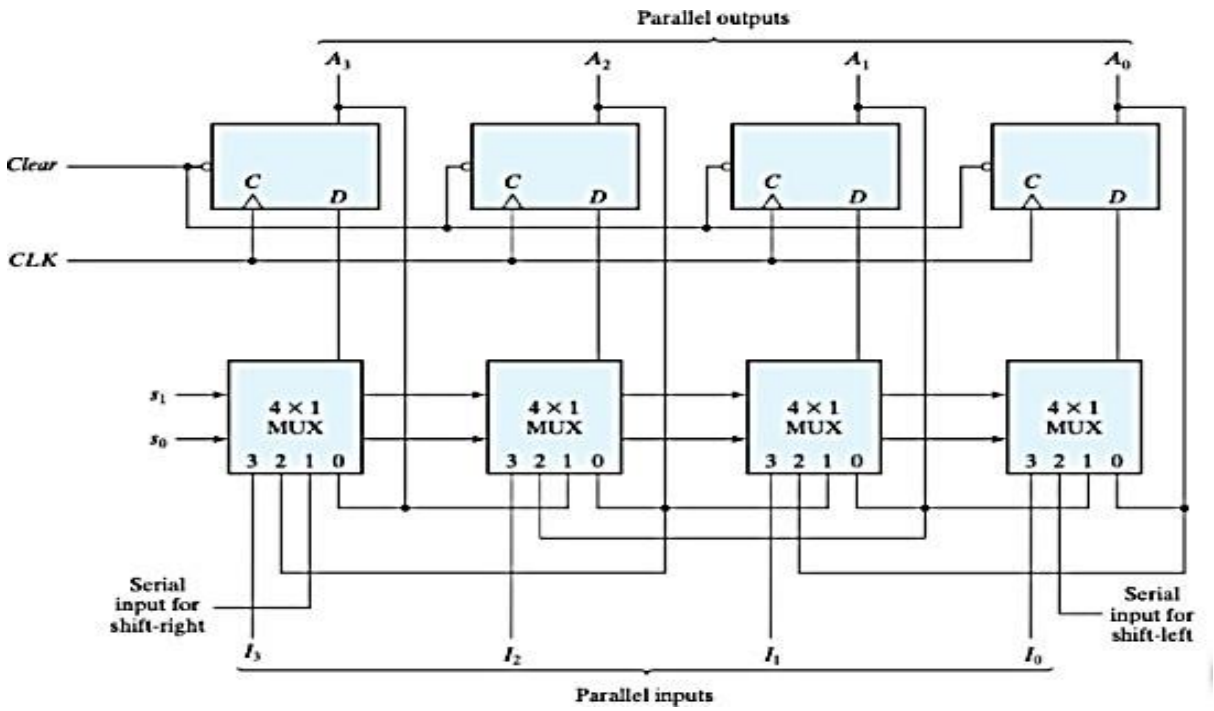


Figure 48: Logic diagram of universal shift register

Ripple Counter

Ripple counter is a special type of **Asynchronous** counter in which the clock pulse ripples through the circuit. The n-MOD ripple counter forms by combining n number of flip-flops. The n-MOD ripple counter can count 2n states, and then the counter resets to its initial value.

Features of the Ripple Counter:

- Different types of flip flops with different clock pulse are used.
- It is an example of an asynchronous counter.
- The flip flops are used in toggle mode.
- The external clock pulse is applied to only one flip flop. The output of this flip flop is treated as a clock pulse for the next flip flop.
- In counting sequence, the flip flop in which external clock pulse is passed, act as LSB.

Based on their circuitry design, the counters are classified into the following types:

Up Counter

The up-counter counts the states in ascending order.

Down Counter

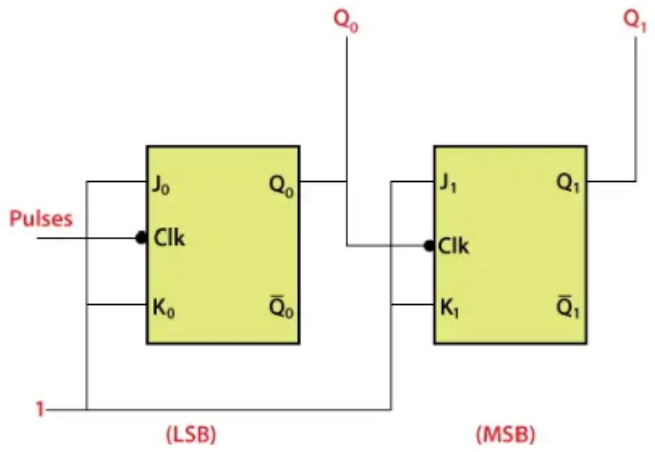
The down counter counts the states in descending order.

Up-Down Counter

The up and down counter is a special type of bi-directional counter which counts the states either in the forward direction or reverse direction. It also refers to a reversible counter.

A **Binary counter** is a **2-Mod counter** which counts up to 2-bit state values, i.e., 22 = 4 values. The flip flops having similar conditions for toggling like T and JK are used to construct the **Ripple counter**. Below is a circuit diagram of **a binary ripple counter**.

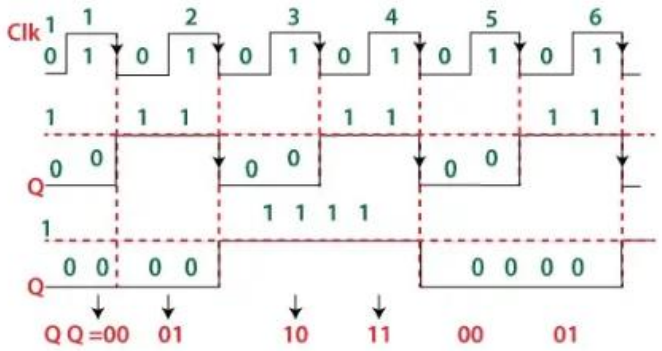
In the circuit design of the binary ripple counter, two JK flip flops are used. The high voltage signal is passed to the inputs of both flip flops. This high voltage input maintains the flip flops at a state 1. In JK flip flops, the negative triggered clock pulse use.



The outputs Q_0 and Q_1 are the LSB and MSB bits, respectively. The truth table of JK flip flop helps us to understand the functioning of the counter.

J_n	K_n	Qn+1
0	0	Q_n
1	0	1
0	1	0
1	1	Q_n

When the high voltage to the inputs of the flip flops, the fourth condition is of the JK flip flop occurs. The flip flops will be at the state 1 when we apply high voltage to the input of the flip-flop. So, the states of the flip flops passes are toggled at the negative going end of the clock pulse. In simple words, the flip flop



The state of the output Q_0 change when the negative clock edge passes to the flip flop. Initially, all the flip flops are set to 0. These flip flop changes their states when the passed clock goes from 1 to 0. The JK flip flop toggles when the inputs of the flip flops are one, and then the flip flop changes its state from 0 to 1. For all the clock pulse, the process remains the same.

Number of input pulses	Q_1	Q_0
0	–	-
1	0	0
2	0	1
3	1	0
4	1	1

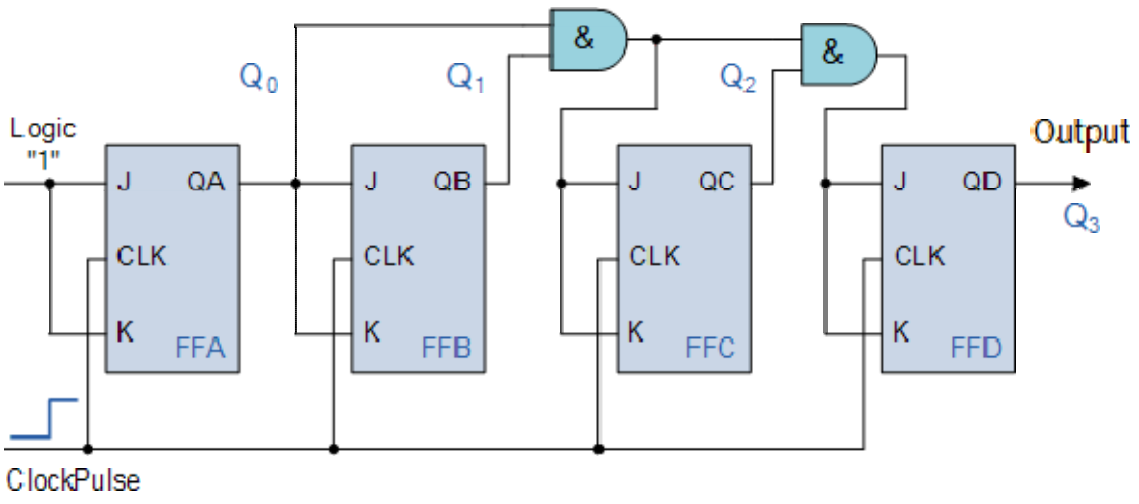
The output of the first flip flop passes to the second flip flop as a clock pulse. From the above timing diagram, it is clear that the state of the second flip flop is changed when the output Q_0 goes transition from 1 to 0. The outputs Q_0 and Q_1 treat as LSB and MSB. The counter counts the values 00, 01, 10, 11. After counting these values, the counter resets itself and starts counting again from 00, 01, 10, and 1. The count values until the clock pulses are passed to J_0K_0 flip flop.

Synchronous Counter

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

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

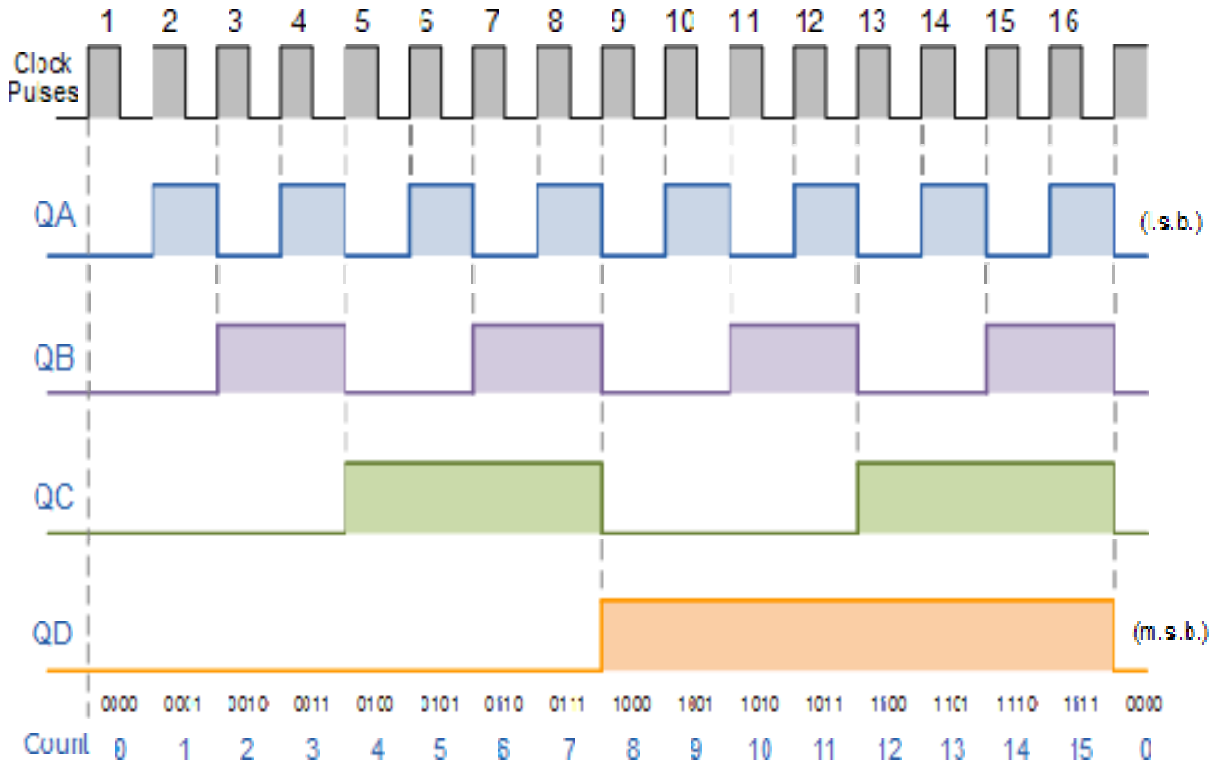
Binary 4-bit Synchronous Counter



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

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

bit Synchronous Counter Timing Diagram.



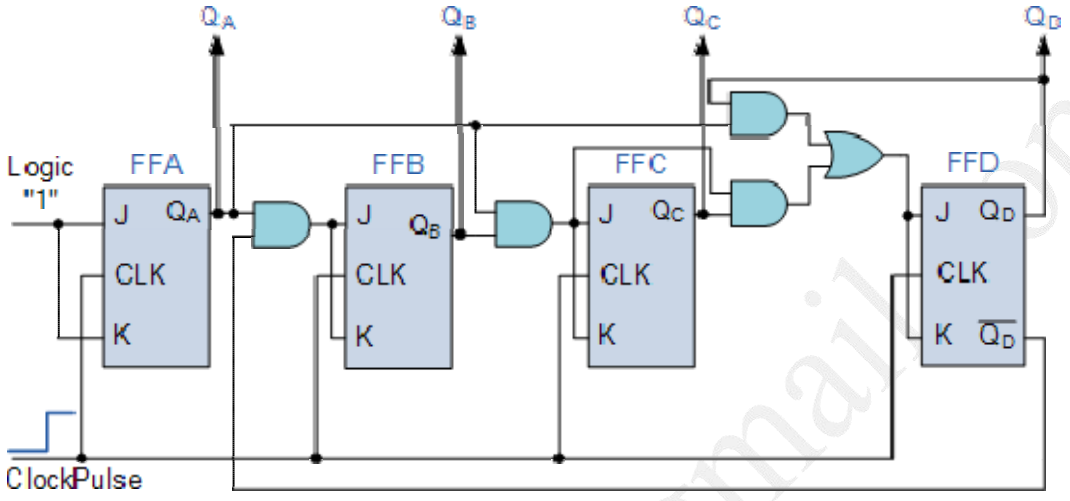
Because this 4-bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 ("0000") to 15 ("1111"). Therefore, this type of counter is also known as a **4-bit Synchronous Up Counter**.

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

Decade 4-bit Synchronous Counter

Another name for Decade Counter is Modulo 10 counter. A 4-bit decade synchronous counter can also be built using synchronous binary counters to produce a count sequence from 0 to 9. A standard binary counter can be converted to a decade (decimal 10) counter with the aid of some additional logic to implement the desired state sequence. After reaching the count of "1001", the counter recycles back to "0000". We now have a decade or **Modulo-10** counter.

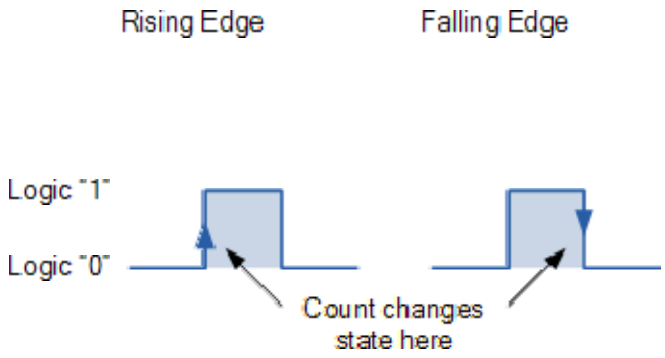
Decade 4-bit Synchronous Counter



The additional AND gates detect when the sequence reaches "1001", (Binary 10) and causes flip-flop FF3to toggle on the next clock pulse. Flip-flop FF0 toggles on every clock pulse. Thus, the count starts over at "0000" producing a synchronous decade counter. We could quite easily re-arrange

the additional ANDgates to produce other counters such as a Mod-12 Up counter which counts 12 states from"0000" to "1011" (0 to 11) and then repeats making them suitable for clocks.

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



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

Summary:

- **Synchronous Counters** can be made from Toggle or D-type flip-flops.
- They are called synchronous counters because the clock input of the flip-flops are clocked with the same clock signal.
- Due to the same clock pulse all outputs change simultaneously.

- Synchronous counters are also called parallel counters as the clock is fed in parallel to all flip-flops.
- Synchronous binary counters use both sequential and combinational logic elements.
- The memory section keeps track of the present state.
- The sequence of the count is controlled by combinational logic.

Advantages of Synchronous Counters:

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

Other Counters

Other types of digital counters include **up-down counters**, which can count in either direction, and **ring counters**, which use a recirculating shift register to pass a data pattern. There are also specialized counters like **decade (or modulo-10) counters** that count through a specific number of states, and counters with features such as **synchronous load** and **asynchronous clear** for resetting or preloading values.

Types of Counters

- **Asynchronous (Ripple) Counters**
  - **Clocking:** The clock signal is applied to the first flip-flop, and each subsequent flip-flop is clocked by the output of the previous one.
  - **Behavior:** This creates a delay, or "ripple," as each flip-flop changes state sequentially.
- **Synchronous Counters**
  - **Clocking:** All flip-flops receive the clock signal simultaneously.
  - **Behavior:** All outputs change at the same time, preventing propagation delays.
- **Up-Down Counters**
  - **Functionality:** Can count in either ascending (up) or descending (down) order.
  - **Control:** An external control input determines the counting mode.
- **Ring Counters**
  - **Structure:** A shift register with the output of the last flip-flop connected to the input of the first.
  - **Operation:** A specific data pattern (e.g., a single '1') circulates between the flip-flops with each clock pulse.
- **Decade (or Modulo-10) Counters**
  - **Function:** A specific type of counter that counts through ten distinct states (typically 0 to 9) before resetting.

Key Concepts

- **Modulus (MOD Number)**: The total number of different states a counter can go through in a complete cycle before returning to its initial state.
- **Flip-Flops**: The basic building blocks of counters, used to store binary states and count clock pulses.
- **Synchronous Load & Asynchronous Clear**: Additional features that allow for presetting a specific count into the counter or immediately resetting it to zero, respectively.

HDL for Registers and Counters

1. Introduction to Registers and Counters

◈ Registers:

- **Definition**: A register is a group of flip-flops used to store multiple bits of data.
- **Purpose**: Temporary data storage, data transfer, data manipulation.
- **Types**:
    - **Parallel Load Register**: All bits are loaded simultaneously.
    - **Shift Register**: Data is shifted left or right.
    - **Bidirectional Register**: Can shift both left and right.

◈ Counters:

- **Definition**: A sequential circuit that counts in binary (or other codes).
- **Types**:
    - **Up Counter**: Increments the count.
    - **Down Counter**: Decrements the count.
    - **Up/Down Counter**: Can do both.
    - **Synchronous vs Asynchronous**:
        - *Synchronous*: All flip-flops triggered simultaneously.
        - *Asynchronous (Ripple)*: Flip-flop output triggers next stage.

2. HDL Basics

◈ Common HDL Languages:

- **Verilog**: C-like syntax, popular in industry.
- **VHDL**: ADA-like syntax, strong typing, used in academia and industry.

◈ Common Elements in HDL:

- **module (Verilog) / entity (VHDL)**: Describes the block.
- **always (Verilog) / process (VHDL)**: Describes behavior.
- **if-else, case, for loops**: Control structures.
- **Signals/Wires/Registers**: Used for interconnection and storage.

3. Register Design in HDL

3.1 Parallel Load Register (4-bit)

Verilog Code:

```
module parallel_register (
  input clk, load,
  input [3:0] d,
  output reg [3:0] q
);
  always @(posedge clk) begin
    if (load)
      q <= d;
  end
endmodule
```

VHDL Code:

```
entity parallel_register is
  Port (
    clk : in std_logic;
    load : in std_logic;
    d : in std_logic_vector(3 downto 0);
    q : out std_logic_vector(3 downto 0)
  );
end parallel_register;

architecture Behavioral of parallel_register is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if load = '1' then
        q <= d;
      end if;
    end if;
  end process;
end Behavioral;
```

4. Shift Register Design

4.1 Right Shift Register (4-bit)

Verilog:

```
module shift_register (
```

```
  input clk, reset, shift_en,
  input in_bit,
  output reg [3:0] q
);
  always @(posedge clk or posedge reset) begin
    if (reset)
      q <= 4'b0000;
    else if (shift_en)
      q <= {in_bit, q[3:1]};
  end
endmodule
```

VHDL:

```
entity shift_register is
  Port (
    clk, reset, shift_en : in std_logic;
    in_bit : in std_logic;
    q : out std_logic_vector(3 downto 0)
  );
end shift_register;

architecture Behavioral of shift_register is
  signal tmp : std_logic_vector(3 downto 0);
begin
  process(clk, reset)
  begin
    if reset = '1' then
      tmp <= (others => '0');
    elsif rising_edge(clk) then
      if shift_en = '1' then
        tmp <= in_bit & tmp(3 downto 1);
      end if;
    end if;
  end process;
  q <= tmp;
end Behavioral;
```

5. Counter Design in HDL

5.1 4-bit Synchronous Up Counter

Verilog:

```
module up_counter (
  input clk, reset,
  output reg [3:0] count
);
  always @(posedge clk or posedge reset) begin
    if (reset)
      count <= 4'b0000;
    else
      count <= count + 1;
  end
endmodule
```

VHDL:

```
entity up_counter is
  Port (
    clk, reset : in std_logic;
    count : out std_logic_vector(3 downto 0)
  );
end up_counter;

architecture Behavioral of up_counter is
  signal tmp : std_logic_vector(3 downto 0);
begin
  process(clk, reset)
  begin
    if reset = '1' then
      tmp <= (others => '0');
    elsif rising_edge(clk) then
      tmp <= std_logic_vector(unsigned(tmp) + 1);
    end if;
  end process;
  count <= tmp;
end Behavioral;
```

5.2 4-bit Up/Down Counter with Enable

Verilog:

```
module up_down_counter (
  input clk, reset, up_down, enable,
  output reg [3:0] count
);
```

```
  always @(posedge clk or posedge reset) begin
    if (reset)
      count <= 4'b0000;
    else if (enable) begin
      if (up_down)
        count <= count + 1;
      else
        count <= count - 1;
    end
  end
endmodule
```

VHDL:

```
entity up_down_counter is
  Port (
    clk, reset, up_down, enable : in std_logic;
    count : out std_logic_vector(3 downto 0)
  );
end up_down_counter;

architecture Behavioral of up_down_counter is
  signal tmp : std_logic_vector(3 downto 0);
begin
  process(clk, reset)
  begin
    if reset = '1' then
      tmp <= (others => '0');
    elsif rising_edge(clk) then
      if enable = '1' then
        if up_down = '1' then
          tmp <= std_logic_vector(unsigned(tmp) + 1);
        else
          tmp <= std_logic_vector(unsigned(tmp) - 1);
        end if;
      end if;
    end if;
  end process;
  count <= tmp;
end Behavioral;
```

6. Key Concepts to Remember

Concept	Description
Clock	Synchronizes state changes.
Reset	Used to initialize counters/registers.
Enable	Controls when operations happen.
Load	Loads new data into register.
Shift	Moves bits left/right in shift registers.
Up/Down	Changes count direction in counters.
Synchronous vs Asynchronous	Timing of updates related to clock.

7. Simulation and Testing

- Use testbenches to verify behavior:
  - Apply various inputs.
  - Observe outputs over clock cycles.
- Tools: ModelSim, Vivado Simulator, GHDL, etc.

8. Applications

- **Registers**:
  - Temporary storage in processors.
  - Pipeline registers in data paths.
  - Serial-to-parallel/parallel-to-serial conversion.
- **Counters**:
  - Frequency division.
  - Digital clocks/timers.
  - State machine state tracking.

UNIT V

Random-Access Memory

- A memory unit stores binary information in groups of bits called words. 1 byte = 8 bits

1 word = 2 bytes

2 The communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

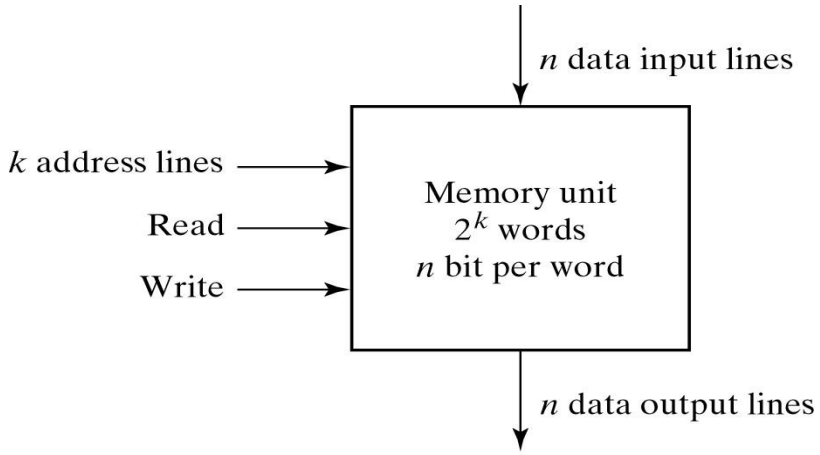


Fig. 7-2 Block Diagram of a Memory Unit

Content of a memory:

- Each word in memory is assigned an identification number, called an address, starting from 0 up to 2^k-1, where k is the number of address lines.
- The number of words in a memory with one of the letters K=2^{10}, M=2^{20}, or G=2^{30}. 64K = 2^{16} 2M = 2^{21}

  4G = 2^{32}

Write and Read operations :

- Transferring a new word to be stored into memory:

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data input lines.
3. Activate the write input.

- Transferring a stored word out of memory:

1. Apply the binary address of the desired word to the address lines.
2. Activate the read input.

- Commercial memory sometimes provide the two control inputs for reading and writing in a somewhat different configuration in table 7-1.

Table 7-1
Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

Memory address		Memory contest
Binary	decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
	•	•
	•	•
	•	•
	•	•
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Fig. 7-3 Content of a 1024 × 16 Memory

Types of memories:

- In random-access memory, the word locations may be thought of as being separated in space,

with each word occupying one particular location.

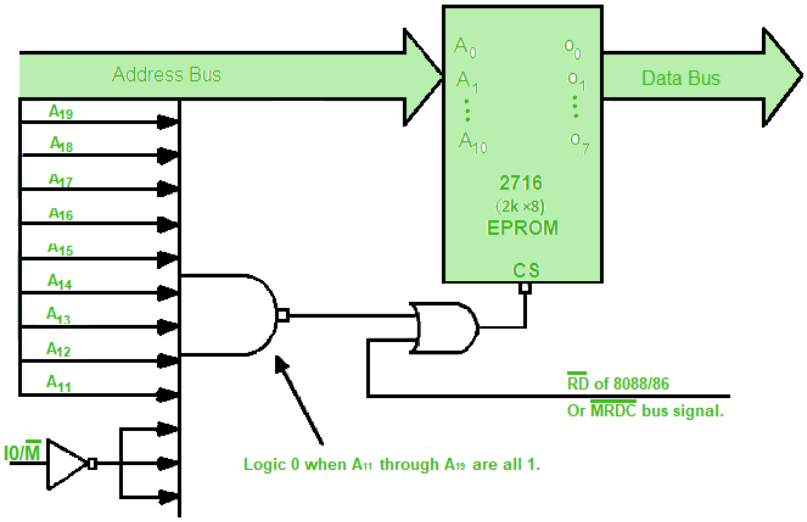
- In sequential-access memory, the information stored in some medium is not immediately accessible, but is available only certain intervals of time. A magnetic disk or tape unit is of this type.

- **There are two basic types of RAM :**

- (i) Dynamic Ram
- (ii) Static RAM

- Dynamic RAM : loses its stored information in a very short time (for milli sec.) even when power supply is on. D-RAM's are cheaper & lower

Memory Decoding



Memory decoding circuit

◈ Digital Circuit Basics

- Digital circuits process signals with **two states**: 0 (low) and 1 (high).
- **Transistors** perform Boolean logic functions.
- **Memory decoding** enables access to specific memory locations using **binary addresses**.

◈ Internal Construction of Memory

- Built from **binary storage cells** arranged in a **matrix**.
- Each **cell stores 1 bit**; multiple cells form **memory words**.
- **Word lines** (rows) are controlled by an **address decoder**.
- **Bit lines** (columns) are connected to **sense/write circuits**:
  - **Read**: Senses and outputs stored bit.
  - **Write**: Inputs and stores bit value.

◈ Memory Decoding Process

- **Purpose**: Identify and access specific memory locations.
- Requires a **decoder** to select memory locations using address inputs.

◈ Memory Cell Operation

- Contains **4–6 transistors**.
- Controlled via **Read/Write (R/W)** signals:
  - **A1** = Read operation (output from latch).
  - **A0** = Write operation (input to latch).
- Each **memory word** has a unique **address** (0 to 2^k – 1).
- **Small RAM** example: 4 words × 4 bits = 16 binary cells.
- **2:4 decoder** used to select among 4 words.

◈ Read/Write Operation

- **Read**: Selected word's bits are sent to the output.

- **Write**: Input data is written into the selected word's cells.
- Unselected words remain unaffected ("dummy cells").

◈ Memory Address Decoding

- For **2^k words**, k address lines are used.
- **Decoder** maps address inputs to unique word selections.
- **Example**: 8088 processor (20-bit address) → 1MB space. EPROM 2716 (2KB) has 11 address pins and fits in 2KB block using decoders for unused pins.

◈ Coincident Decoding

- Reduces hardware in large memory arrays.
- Uses **two smaller decoders** (row and column).
- **Example**: For 1K memory words:
    - Instead of 1×10-line decoder (1,024 AND gates),
    - Use two 5×32 decoders (64 AND gates).
    - Memory accessed at the **intersection** of row and column.

◈ Address Multiplexing

- Reduces **pin count** on memory ICs.
- **Same lines** carry address and data (at different times).
- Address split into:
    - **Row address** → RAS (Row Address Strobe).
    - **Column address** → CAS (Column Address Strobe).
- Common in **DRAM** (Dynamic RAM):
    - Uses fewer transistors (higher density, cheaper).
    - Needs **multiplexing** due to large size.
- **Example**: 64K words = 256 rows × 256 columns.

◈ Comparison: SRAM vs DRAM

Feature	**SRAM**	**DRAM**
Transistors	6 per cell	1 per cell
Speed	Faster	Slower
Cost	More expensive	Cheaper
Density	Lower	Higher
Power usage	Higher	Lower

ERROR DETECTION AND CORRECTION CODES

We know that the bits 0 and 1 corresponding to two different range of analog voltages. So, during transmission of binary data from one system to the other, the noise may also be added. Due to this, there may be

errors in the received data at other system.

That means a bit 0 may change to 1 or a bit 1 may change to 0. We can't avoid the interference of noise. But, we can get back the original data first by detecting whether any errors present and then correcting those errors. For this purpose, we can use the following codes.

- Error detection codes
- Error correction codes

Error detection codes − are used to detect the errors present in the received data bit stream. These codes contain some bits, which are included appended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bit stream. **Example** − Parity code, Hamming code.
Error correction codes − are used to correct the errors present in the received data bit stream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes. **Example** − Hamming code.
Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

Parity Code

It is easy to include append one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in **even parity code**. Even parity code contains the data bits and even parity bit.

The following table shows the **even parity codes** corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101

011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- If the other system receives other than even parity codes, then there will be an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Odd Parity Code

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.

The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	**Odd Parity bit**	**Odd Parity Code**
000	1	0001

001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Hamming Code

Hamming code is useful for both detection and correction of error present in the received data. This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.

The **minimum value of 'k'** for which the following relation is correct valid is nothing but the required number of parity bits.

$2^k \geq n+k+1$ $2^k \geq n+k+1$ Where,

'n' is the number of bits in the binary code information

'k' is the number of parity bits

Therefore, the number of bits in the Hamming code is equal to n + k.

Let the **Hamming code** is bn+kbn+k−1.....b3b2b1bn+kbn+k−1. .....b3b2b1 &

parity bits pk,pk−1,....p1pk,pk−1,....p1. We can place the 'k' parity bits in powers of 2 positions only. In remaining bit positions, we can place the 'n' bits of binary code.

Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.

Follow this procedure for finding **parity bits**.

- Find the value of $\mathbf{p_1}$, based on the number of ones present in bit positions b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of $\mathbf{p_2}$, based on the number of ones present in bit positions b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of $\mathbf{p_3}$, based on the number of ones present in bit positions b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of parity bits.

Follow this procedure for finding **check bits**.

- Find the value of c_1, based on the number of ones present in bit positions b_1, b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of c_2, based on the number of ones present in bit positions b_2, b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of c_3, based on the number of ones present in bit positions b_4, b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of check bits.

The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present. Just complement the value present in that bit position. Therefore, we will get the original binary code after removing parity bits.

READ ONLY MEMORY

A ROM is essentially a memory device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established it stays within the unit even when power is turned off and on again.

A block diagram of ROM is shown in the Figure 1. It consists of k inputs and n outputs. The inputs provide the address for the memory and the outputs give the data bits of the stored word which is selected by the address. The number of words in a ROM is determined from the fact that k address input lines are needed to specify 2^k words.

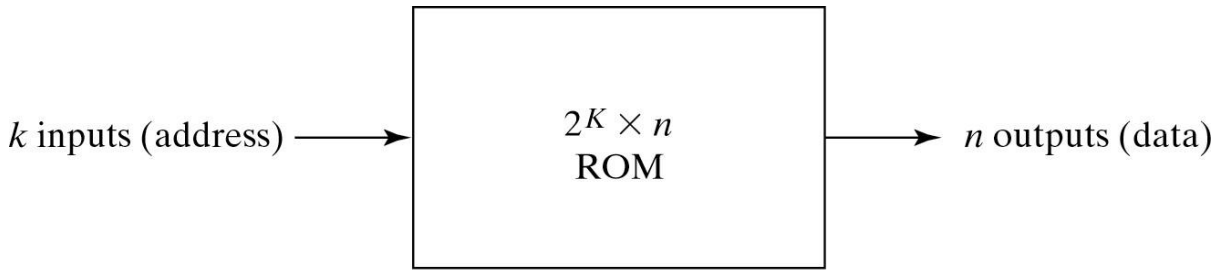


Fig. 7-9 ROM Block Diagram

ROM does not have data inputs because it does not have a write operation.

Consider for example a 32 x 8 ROM. The unit consists of 32 words of 8 bits each. There are five input lines that form the binary numbers from 0 through 31 for the address. The Figure 2 shows the internal logic construction of the ROM. The five inputs are decoded into 32 distinct outputs by means of a 5 x 32 decoder. Each output of the decoder represents a memory address. The 32 outputs of the decoder are connected to each of the eight OR gates.

The diagram shows the array logic convention used in complex circuits . Each OR gate must be considered as having 32 inputs. Each output of the decoder is connected to one of the inputs of each OR gate. Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains 32 x 8 = 256 internal connections.

In general, a 2^k x n ROM will have an internal k x 2^k decoder and n OR gates. Each OR gate has 2^k inputs, which are connected to each of the outputs of the decoder.

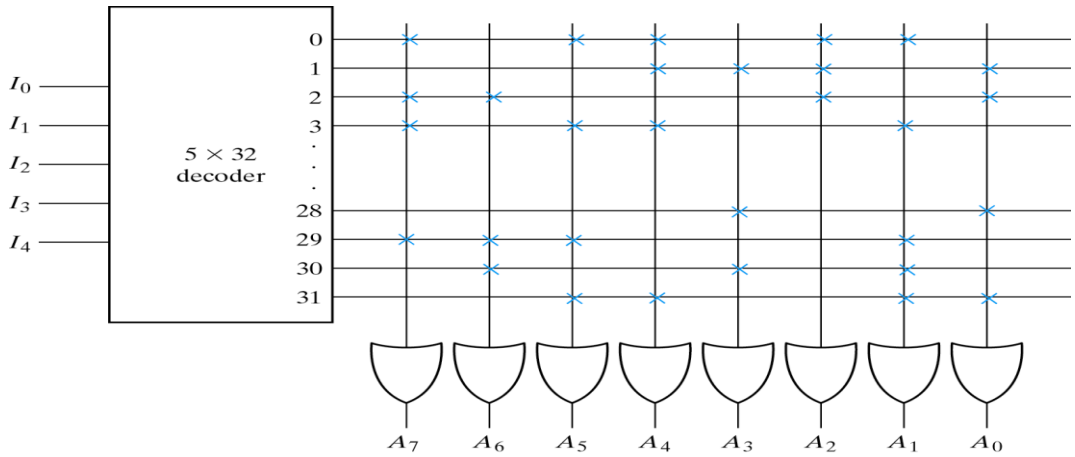


Fig. 7-11 Programming the ROM According to Table 7-3

Inputs					Outputs				
I4	I3	I2	I1	I0	A7	A6	A5	A4	A3
0	0	0	0	0	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1
0	0	0	1	0	1	1	0	0	0
0	0	0	1	1	1	0	1	1	0
		⋮					⋮		
1	1	1	0	0	0	0	0	0	1
1	1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0

Every 0 listed in the truth table specifies a no connection and every 1 listed specifies a path that is obtained by a connection. The four 0's in the word are programmed by blowing the fuse links between output 3 of the decoder and the inputs of the OR gates associated with outputs A6 , A3, A2 and A0. The four 1's in the word are marked in the diagram with a X to denote a connection in place of a dot used for permanent connection in logic diagrams.

When the input of the ROM is 00011, all the outputs of the decoder are 0 except for output 3, which is at logic 1. The signal equivalent to logic 1 at decoder output 3 propagates through the connections to the OR gate outputs of A7 , A5, A4 and A1.The other four outputs remain at 0. The result is that the stored word 10110010 is applied to the eight data outputs.

Types of ROMs:

ROM : Read only memory: Its non volatile memory, ie, the information stored in it, is not lost even if the power supply goes off. It's used for the permanent storage of information. It also posses random access property. Information can not be written into a ROM by the users/programmers. In other words the contents of ROMs are decided by the manufactures.

The following types of ROMs an listed below :

(i) PROM : It's programmable ROM. Its contents are decided by the user. The user can store permanent programs, data etc in a PROM. The data is fed into it using a PROM programs.

(ii) EPROM : An EPROM is an erasable PROM. The stored data in EPROM's can be erased by exposing it to UV light for about 20 min. It's not easy to erase it because the EPROM IC has to be removed from the computer and exposed to UV light. The entire data is erased and not selected portions by the user. EPROM's are cheap and reliable.

(iii) EEPROM (Electrically Erasable PROM) : The chip can be erased & reprogrammed on the board easily byte by byte. It can be erased with in a few milliseconds. There is a limit on the number of times the EEPROM's can be reprogrammed, i.e.; usually around 10,000 times.

- A combinational PLD is an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of product implementation.
- PROM: fixed AND array constructed as a decoder and programmable OR array.
- PAL: programmable AND array and fixed OR array.
- PLA: both the AND and OR arrays can be programmed.
- The required paths in a ROM may be programmed in four different ways.

1. Mask programming: fabrication process
2. Read-only memory or PROM: blown fuse /fuse intact
3. Erasable PROM or EPROM: placed under a special ultraviolet light for a given period of time will erase the pattern in ROM.
4. Electrically-erasable PROM(EEPROM): erased with an electrical signal instead of ultraviolet light.

Switching Circuit

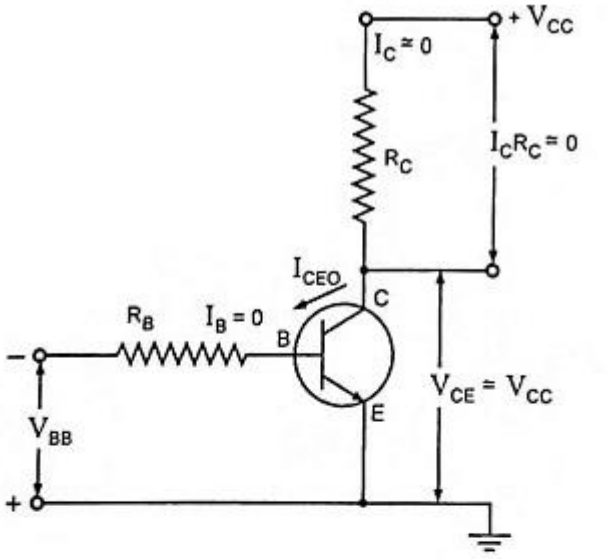
In practice, we often need making (switching on) and breaking (switching off) of an electrical circuit. It is also desirable and sometimes essential that the switching operation (making and breaking of an electrical circuit) be very fast and without sparking. For this purpose, we need a switch, which is defined as a device that can turn on or off current in an electrical circuit. On the other hand, a circuit that can turn on or off current in an electrical circuit is referred to as **switching circuit**.

The switching circuit essentially consists of a **switch** and an **associated circuitry**. The switch is the most vital part of the switching circuit as it is the component which actually makes or breaks the electrical circuit. The switches are of several types such as mechanical switch like the common tumbler switch used in house wiring, electromechanical switch like the contactor or relay and electronic switch like the transistor switch having no moving part.

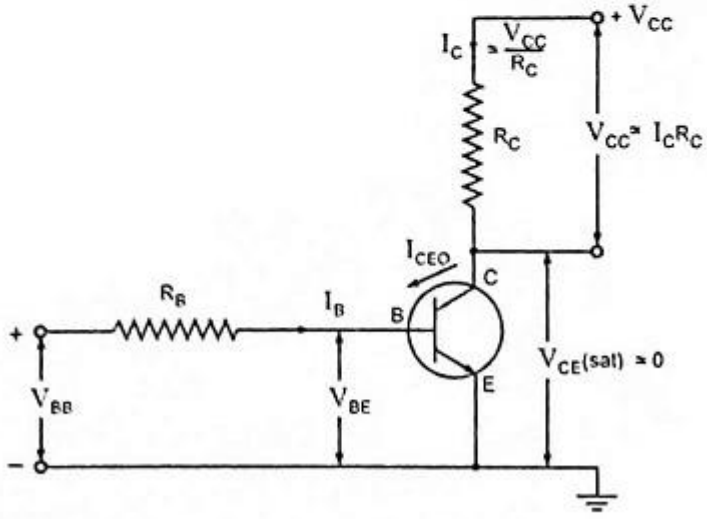
Nowadays, electronic switches are becoming more and more popular because of

1. absence of moving parts resulting in no wear and tear and providing noiseless operation
2. long life
3. smaller size and weight
4. lower cost
5. trouble free service and
6. high operating speed (up to 10^9 operations per second).

The associated circuitry, particularly used in electronic switching, is to help the switch in turning on or off current in the circuit. A transistor can be employed as an electronic switch. Operating a transistor as a switch means operating it at either saturation or cutoff but nowhere else along the load line. When a transistor is saturated, it is like a closed switch from the collector to emitter. When the transistor is cutoff, it is like an open switch. The operation of a transistor as a switch is illustrated in Fig. 31.1.



(a) Off-Biased Transistor



(b) On-Biased Transistor

Fig. 31.1 *Operation of a Transistor as a Switch*

1. Switching Circuits

Overview:

Switching circuits operate in two distinct states — ON (logic 1 or HIGH) and OFF (logic 0 or LOW). They are the fundamental building blocks of digital systems.

Key Concepts:

- **Binary Switching**: Only two states — 0 and 1.
- **Types of Switching Circuits**:
    - **Combinational Circuits**: Output depends only on current input.
        - Examples: Adders, Multiplexers, Decoders.
    - **Sequential Circuits**: Output depends on current input and past inputs (memory).
        - Examples: Flip-flops, Counters, Shift Registers.

Logic Levels:

- Logic 0: 0V – 0.8V (LOW)
- Logic 1: 2V – 5V (HIGH) *(for TTL logic family)*

2. 7400 Series TTL (Transistor-Transistor Logic)

What is TTL?

TTL is a class of digital circuits built using **bipolar junction transistors (BJTs)**. The 7400 series is the most widely used TTL family.

Features:

- Operates on 5V DC supply.
- Fast switching speed (~10 ns).
- Compatible with a wide range of ICs.

Common 7400 Series ICs:

IC Number	Function
7400	Quad 2-input NAND gates
7402	Quad 2-input NOR gates
7404	Hex Inverters
7408	Quad 2-input AND gates
7432	Quad 2-input OR gates
7474	Dual D Flip-Flops
7490	Decade Counter
74138	3-to-8 Line Decoder
74153	Dual 4-to-1 Multiplexer

3. TTL Parameters

Key Electrical Characteristics:

Parameter	Description
VIL	Maximum input voltage for logic 0 (typically 0.8V)
VIH	Minimum input voltage for logic 1 (typically 2V)
VOL	Maximum output voltage for logic 0
VOH	Minimum output voltage for logic 1
IIL	Input current for logic 0
IIH	Input current for logic 1
IOL	Output current when output is LOW
IOH	Output current when output is HIGH
Fan-out	Number of inputs a single output can drive (typically 10 for TTL)
Propagation Delay	Time taken for the output to respond to an input change (~10 ns for TTL)

4. TTL Overview

Advantages:

- Fast switching.
- Easy interfacing.
- Wide availability (standard ICs).

Disadvantages:

- High power consumption (compared to CMOS).
- Limited fan-out.
- Fixed voltage supply requirement (5V).

TTL Families:

TTL Family	Features
Standard TTL	Basic performance, moderate speed
74LS	Low power Schottky, faster, low power
74S	Schottky TTL, very fast, high power
74ALS	Advanced Low Power Schottky

TTL Family	Features
74H	High-speed TTL

Applications of Digital Integrated Circuits

1. Multiplexing Displays

Purpose:

Used to control multiple 7-segment displays or LED arrays using fewer I/O lines.

Working:

- Each digit/display is activated one at a time in rapid succession.
- Persistence of vision makes it appear all are ON simultaneously.
- Controlled using **multiplexers, BCD counters (e.g., 7490), and decoders (e.g., 7447)**.

Benefits:

- Reduced wiring complexity.
- Saves microcontroller/microprocessor pins.

2. Frequency Counters

Purpose:

Measures the frequency of a periodic signal.

Block Diagram Components:

- **Input Signal Conditioning**: Converts signal to digital pulses.
- **Gate Control**: Opens for a specific time window.
- **Counter (7490/7493)**: Counts pulses during the gate period.
- **Display Unit**: Shows the count (frequency value).

Working:

- Number of pulses in a fixed time (e.g., 1 second) gives frequency in Hz.

3. Time Measurement (Time Interval Counters)

Used for:

- Measuring time between two events.
- Applications in physics experiments, radar, communication systems.

Working Principle:

- **Start Pulse**: Opens gate.
- **Stop Pulse**: Closes gate.
- **High-frequency clock** pulses counted between start and stop.

Circuit Elements:

- Start/Stop logic (flip-flops).

- High-frequency clock (crystal oscillator).
- Counters (7490/7493).
- Display.

4. Digital Voltmeter (DVM)

Purpose:

Measures analog voltage and displays it in digital form.

Types:

- Ramp Type
- Dual Slope Integrating Type (common in IC-based DVMs)

General Working:

1. **Analog Input → ADC** (Analog-to-Digital Converter)
2. **Digital Output → Display** (7-segment or LCD)
3. Control circuitry using digital ICs (counters, gates, etc.)

Key ICs:

- ADC0804: ADC
- 7447: BCD to 7-segment decoder
- 7490: BCD Counter

Features:

- High accuracy.
- Less parallax error.
- Suitable for portable equipment.