UNIT-II

**DIGITAL LOGIC & FUNDAMENTALS**

Boolean Algebra is used to analyse and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as **Binary Algebra** or **logical Algebra**. Boolean algebra was invented by **George Boole** in 1854.

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

* Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
* Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as B Bar. Thus if B = 0 then B Bar = 1 and B = 1 then B Bar = 0.
* ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
* Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

**Boolean Laws**

There are six types of Boolean Laws.

**Commutative law**

Any binary operation which satisfies the following expression is referred to as commutative operation.

Commutative Law

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

**Associative law**

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

Associative Law

**Distributive law**

Distributive law states the following condition.

Distributive Law

**AND law**

**These laws** use the AND operation. Therefore, they are called as **AND** laws.

AND Law

**OR law**

These laws use the OR operation. Therefore, they are called as **OR** laws.

OR Law

**INVERSION law**

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

NOT LawLogic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

**Canonical Forms**

For a Boolean expression there are two kinds of canonical forms −

* The sum of minterms (SOM) form
* The product of maxterms (POM) form

**The Sum of Minterms (SOM) or Sum of Products (SOP) form**

A minterm is a product of all variables taken either in their direct or complemented form. Any Boolean function can be expressed as a sum of its 1-minterms and the inverse of the function can be expressed as a sum of its 0-minterms. Hence,

F (list of variables) = ∑ (list of 1-minterm indices)

and

F' (list of variables) = ∑ (list of 0-minterm indices)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **Term** | **Minterm** |
| 0 | 0 | 0 | x’y’z’ | m0 |
| 0 | 0 | 1 | x’y’z | m1 |
| 0 | 1 | 0 | x’yz’ | m2 |
| 0 | 1 | 1 | x’yz | m3 |
| 1 | 0 | 0 | xy’z’ | m4 |
| 1 | 0 | 1 | xy’z | m5 |
| 1 | 1 | 0 | xyz’ | m6 |
| 1 | 1 | 1 | xyz | m7 |

**The Product of Maxterms (POM) or Product of Sums (POS) form**

A maxterm is addition of all variables taken either in their direct or complemented form. Any Boolean function can be expressed as a product of its 0-maxterms and the inverse of the function can be expressed as a product of its 1-maxterms. Hence,

F(list of variables) = π� (list of 0-maxterm indices).

and

F'(list of variables) = π� (list of 1-maxterm indices).

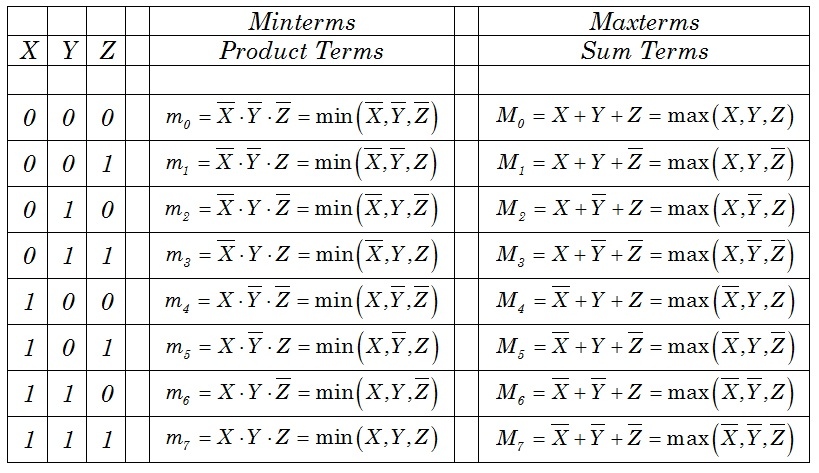
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **Term** | **Maxterm** |
| 0 | 0 | 0 | x + y + z | M0 |
| 0 | 0 | 1 | x + y + z’ | M1 |
| 0 | 1 | 0 | x + y’ + z | M2 |
| 0 | 1 | 1 | x + y’ + z’ | M3 |
| 1 | 0 | 0 | x’ + y + z | M4 |
| 1 | 0 | 1 | x’ + y + z’ | M5 |
| 1 | 1 | 0 | x’ + y’ + z | M6 |
| 1 | 1 | 1 | x’ + y’ + z’ | M7 |

**Standard Form –** A Boolean variable can be expressed in either true form or complemented form. In standard form Boolean function will contain all the variables in either true form or complemented form while in canonical number of variables depends on the output of SOP or POS.

A Boolean function can be expressed algebraically from a given truth table by forming a :

* minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.
* maxterm for each combination of the variables that produces a 0 in the function and then taking the AND of all those terms.

**Truth table representing minterm and maxterm –** 



**Karnaugh Maps**

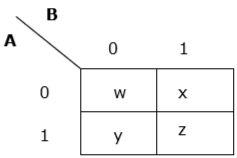
The Karnaugh map (K–map), introduced by Maurice Karnaughin in 1953, is a grid-like representation of a truth table which is used to simplify boolean algebra expressions. A Karnaugh map has zero and one entries at different positions. It provides grouping together Boolean expressions with common factors and eliminates unwanted variables from the expression. In a K-map, crossing a vertical or horizontal cell boundary is always a change of only one variable.

Example 1

An arbitrary truth table is taken below −

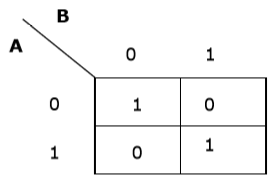
|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A operation B** |
| 0 | 0 | w |
| 0 | 1 | x |
| 1 | 0 | y |
| 1 | 1 | z |

Now we will make a k-map for the above truth table −



Example 2

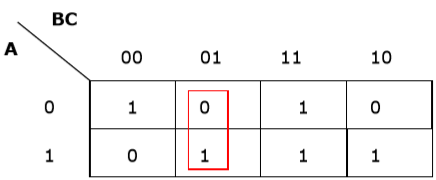
Now we will make a K-map for the expression − AB+ A’B’



**Simplification Using K-map**

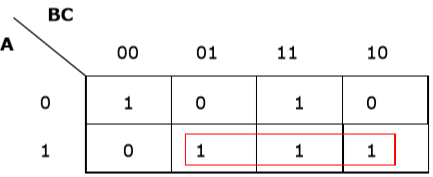
K-map uses some rules for the simplification of Boolean expressions by combining together adjacent cells into single term. The rules are described below −

**Rule 1** − Any cell containing a zero cannot be grouped.



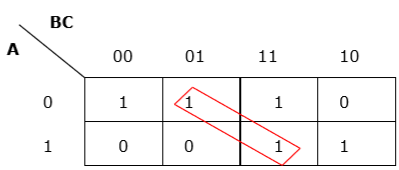
*Wrong grouping*

**Rule 2** − Groups must contain 2n cells (n starting from 1).

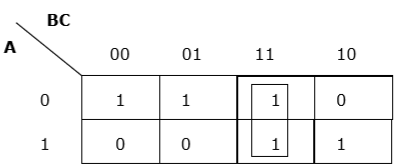


*Wrong grouping*

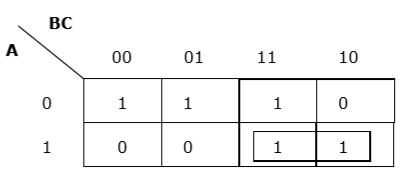
**Rule 3** − Grouping must be horizontal or vertical, but must not be diagonal.



*Wrong diagonal grouping*

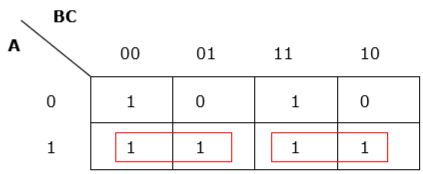


*Proper vertical grouping*

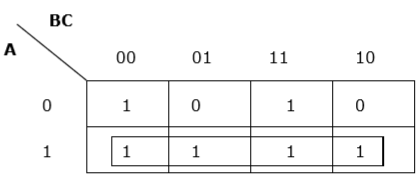


*Proper horizontal grouping*

**Rule 4** − Groups must be covered as largely as possible.

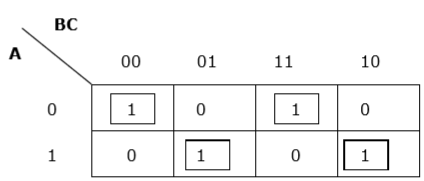


*Insufficient grouping*



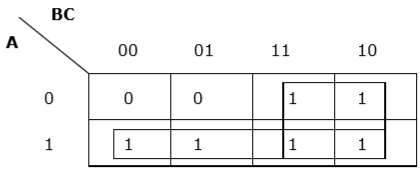
*Proper grouping*

**Rule 5** − If 1 of any cell cannot be grouped with any other cell, it will act as a group itself.



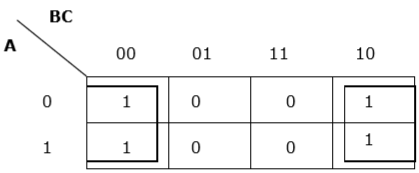
*Proper grouping*

**Rule 6** − Groups may overlap but there should be as few groups as possible.



*Proper grouping*

**Rule 7** − The leftmost cell/cells can be grouped with the rightmost cell/cells and the topmost cell/cells can be grouped with the bottommost cell/cells.



*Proper grouping*

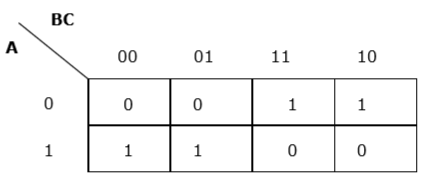
Problem

Minimize the following Boolean expression using K-map −

F(A,B,C)=A′BC+A′BC′+AB′C′+AB′C�(�,�,�)=�′��+�′��′+��′�′+��′�

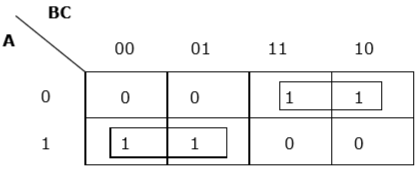
Solution

Each term is put into k-map and we get the following −



*K-map for F (A, B, C)*

Now we will group the cells of 1 according to the rules stated above −

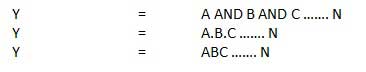


*K-map for F (A, B, C)*

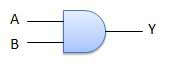
We have got two groups which are termed as A′B�′� and AB′��′. Hence, F(A,B,C)=A′B+AB′=A⊕B�(�,�,�)=�′�+��′=�⊕�. It is the minimized form.

AND Gate

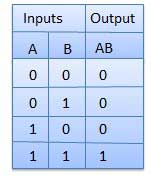
A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.



Logic diagram



Truth Table



OR Gate

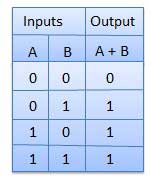
A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

OR gate

Logic diagram

OR Logical Diagram

Truth Table

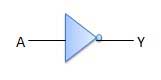


NOT Gate

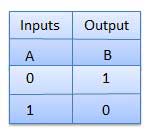
NOT gate is also known as **Inverter**. It has one input A and one output Y.

NOT gate

Logic diagram



Truth Table

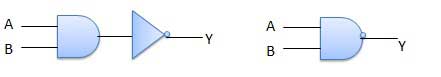


NAND Gate

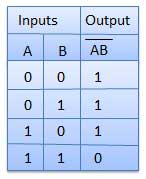
A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

NAND gate

Logic diagram



Truth Table

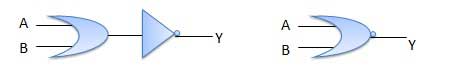


NOR Gate

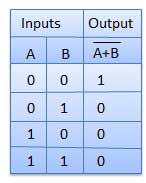
A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

NOR gate

Logic diagram

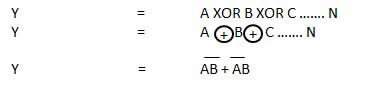


Truth Table

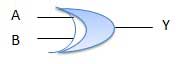


XOR Gate

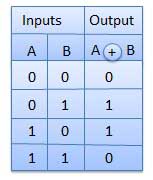
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.



Logic diagram

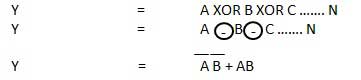


Truth Table

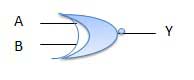


XNOR Gate

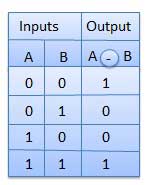
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.



Logic diagram



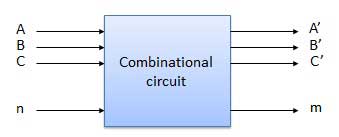
Truth Table



Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following −

* The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
* The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
* A combinational circuit can have an n number of inputs and m number of outputs.

Block diagram

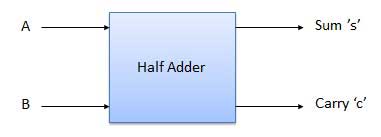


We're going to elaborate few important combinational circuits as follows.

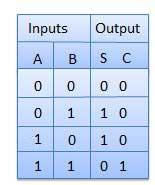
Half Adder

Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

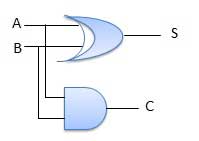
Block diagram



Truth Table



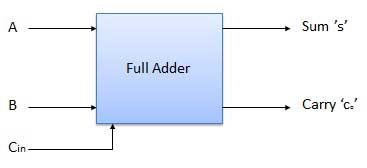
Circuit Diagram



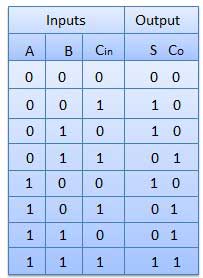
Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

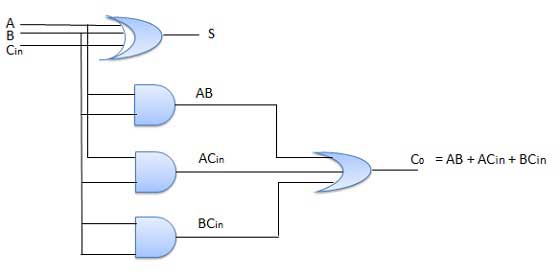
Block diagram



Truth Table



Circuit Diagram



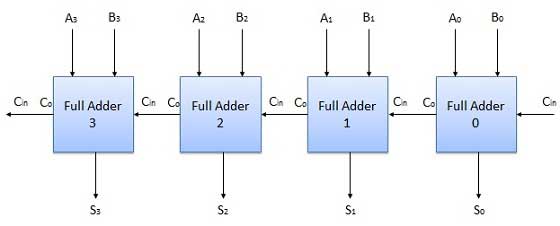
N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A0 and B0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its Cin has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

Block diagram



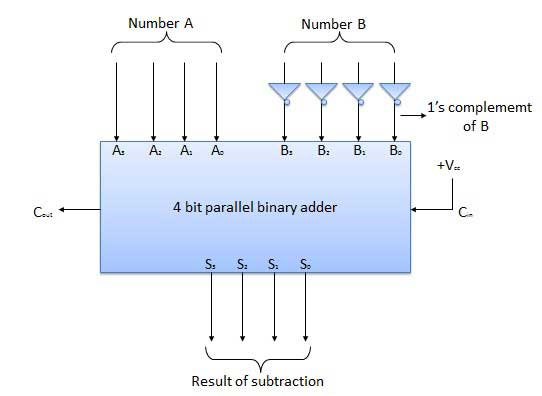
N-Bit Parallel Subtractor

The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction (A-B) by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted (B) is first passed through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S3 S2 S1 S0 represents the result of binary subtraction (A-B) and carry output Cout represents the polarity of the result. If A > B then Cout = 0 and the result of binary form (A-B) then Cout = 1 and the result is in the 2's complement form.

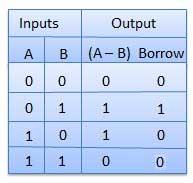
Block diagram



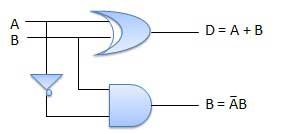
Half Subtractors

Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Truth Table



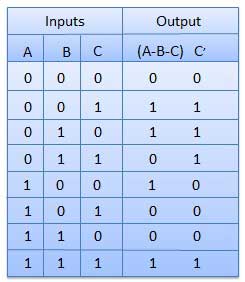
Circuit Diagram



Full Subtractors

The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table

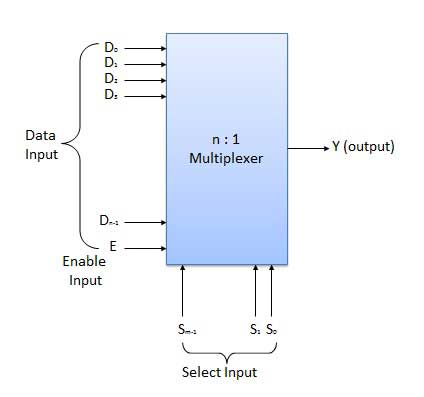


Circuit Diagram

Multiplexers

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with 2m = n. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the strobe or enable input which is useful for the cascading. It is generally an active low terminal that means it will perform the required operation when it is low.

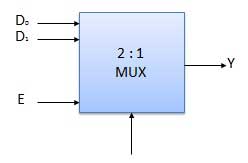
Block diagram



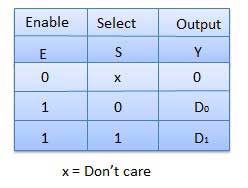
Multiplexers come in multiple variations

* 2 : 1 multiplexer
* 4 : 1 multiplexer
* 16 : 1 multiplexer
* 32 : 1 multiplexer

Block Diagram



Truth Table



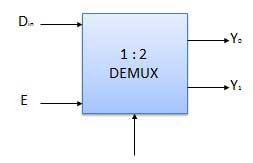
Demultiplexers

A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in fig.

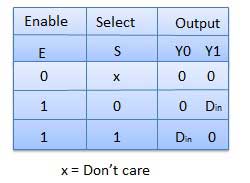
Demultiplexers comes in multiple variations.

* 1 : 2 demultiplexer
* 1 : 4 demultiplexer
* 1 : 16 demultiplexer
* 1 : 32 demultiplexer

Block diagram



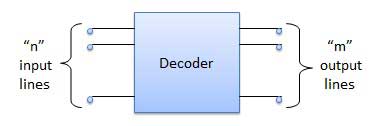
Truth Table



Decoder

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Block diagram



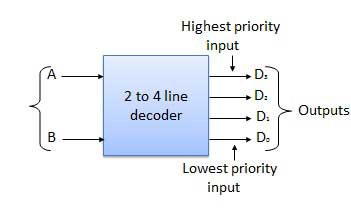
Examples of Decoders are following.

* Code converters
* BCD to seven segment decoders
* Nixie tube decoders
* Relay actuator

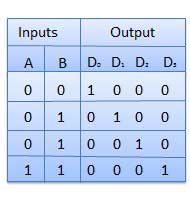
2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

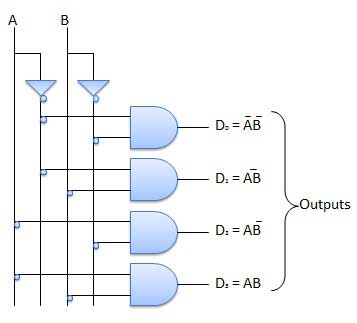
Block diagram



Truth Table



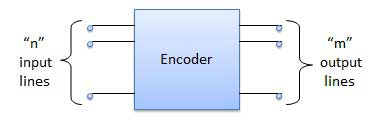
Logic Circuit



Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.

Block diagram

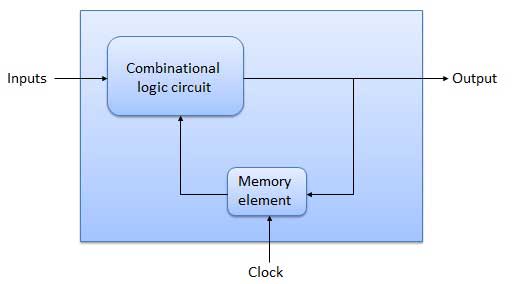


Examples of Encoders are following.

* Priority encoders
* Decimal to BCD encoder
* Octal to binary encoder
* Hexadecimal to binary encoder

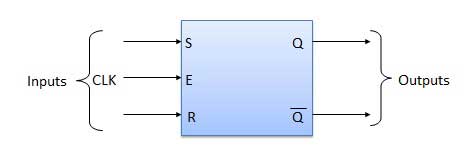
Sequential Circuits

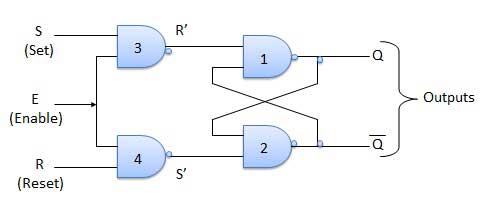
* The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.
* Block diagram

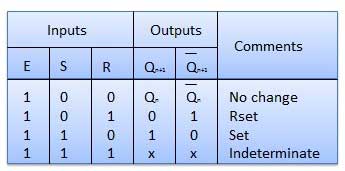


Flip Flop

* Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip flop is said to be edge sensitive or edge triggered rather than being level triggered like latches.
* S-R Flip Flop
* It is basically S-R latch using NAND gates with an additional **enable** input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.
* Block Diagram

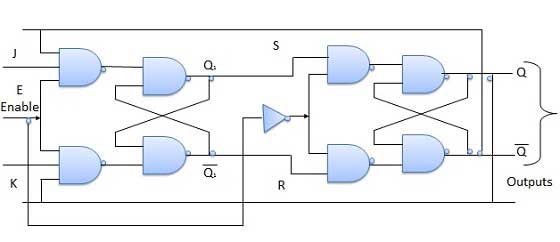


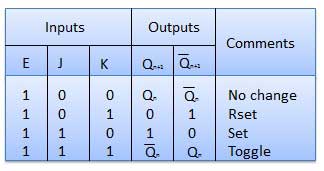
* Circuit Diagram
* 
* Truth Table



Operation

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Condition** | **Operation** |
| 1 | **S = R = 0 : No change** | If S = R = 0 then output of NAND gates 3 and 4 are forced to become 1.  Hence R' and S' both will be equal to 1. Since S' and R' are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs. |
| 2 | **S = 0, R = 1, E = 1** | Since S = 0, output of NAND-3 i.e. R' = 1 and E = 1 the output of NAND-4 i.e. S' = 0.  Hence Qn+1 = 0 and Qn+1 bar = 1. This is reset condition. |
| 3 | **S = 1, R = 0, E = 1** | Output of NAND-3 i.e. R' = 0 and output of NAND-4 i.e. S' = 1.  Hence output of S-R NAND latch is Qn+1 = 1 and Qn+1 bar = 0. This is the reset condition. |
| 4 | **S = 1, R = 1, E = 1** | As S = 1, R = 1 and E = 1, the output of NAND gates 3 and 4 both are 0 i.e. S' = R' = 0.  Hence the **Race** condition will occur in the basic NAND latch. |

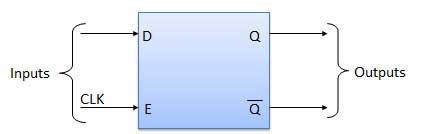
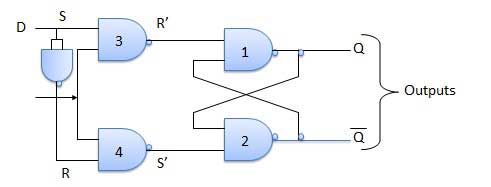
* Master Slave JK Flip Flop
* Master slave JK FF is a cascade of two S-R FF with feedback from the output of second to input of first. Master is a positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and master is inactive.
* Circuit Diagram
* 
* Truth Table

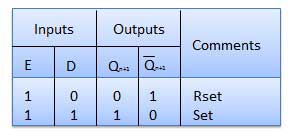


Operation

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Condition** | **Operation** |
| 1 | **J = K = 0 (No change)** | When clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged. Therefore outputs will not change if J = K =0. |
| 2 | **J = 0 and K = 1 (Reset)** | Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q1 = 0 and Q1 bar = 1. That means S = 0 and R =1.  Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 0 and Q bar = 1.  Again clock = 1 − Master active, slave inactive. Therefore even with the changed outputs Q = 0 and Q bar = 1 fed back to master, its output will be Q1 = 0 and Q1 bar = 1. That means S = 0 and R = 1.  Hence with clock = 0 and slave becoming active the outputs of slave will remain Q = 0 and Q bar = 1. Thus we get a stable output from the Master slave. |
| 3 | **J = 1 and K = 0 (Set)** | Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q1 = 1 and Q1 bar = 0. That means S = 1 and R =0.  Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 1 and Q bar = 0.  Again clock = 1 − then it can be shown that the outputs of the slave are stabilized to Q = 1 and Q bar = 0. |
| 4 | **J = K = 1 (Toggle)** | Clock = 1 − Master active, slave inactive. Outputs of master will toggle. So S and R also will be inverted.  Clock = 0 − Slave active, master inactive. Outputs of slave will toggle.  These changed output are returned back to the master inputs. But since clock = 0, the master is still inactive. So it does not respond to these changed outputs. This avoids the multiple toggling which leads to the race around condition. The master slave flip flop will avoid the race around condition. |

Delay Flip Flop / D Flip Flop

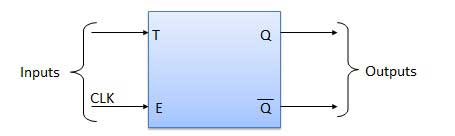
* Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs. It has only one input. The input data is appearing at the output after some time. Due to this data delay between i/p and o/p, it is called delay flip flop. S and R will be the complements of each other due to NAND inverter. Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoid by SR = 00 and SR = 1 conditions.
* Block Diagram
* 
* Circuit Diagram
* 
* Truth Table

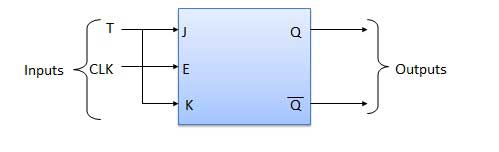


* Operation

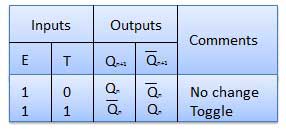
|  |  |  |
| --- | --- | --- |
| **S.N.** | **Condition** | **Operation** |
| 1 | **E = 0** | Latch is disabled. Hence no change in output. |
| 2 | **E = 1 and D = 0** | If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is Qn+1 = 0 and Qn+1 bar = 1. This is the reset condition. |
| 3 | **E = 1 and D = 1** | If E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and Qn+1 = 1 and Qn+1 bar = 0 irrespective of the present state. |

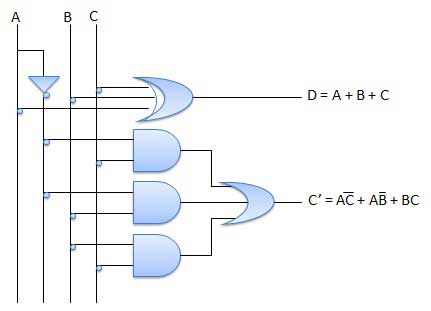
Toggle Flip Flop / T Flip Flop

* Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only input denoted by **T** as shown in the Symbol Diagram. The symbol for positive edge triggered T flip flop is shown in the Block Diagram.
* Symbol Diagram
* 
* Block Diagram



* Truth Table





* Operation

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Condition** | **Operation** |
| 1 | **T = 0, J = K = 0** | The output Q and Q bar won't change |
| 2 | **T = 1, J = K = 1** | Output will toggle corresponding to every leading edge of clock signal. |

Basic Logical Operations

**1. Negation:** It means the opposite of the original statement. If p is a statement, then the negation of p is denoted by ~p and read as 'it is not the case that p.' So, if p is true then ~ p is false and vice versa.

**Example:** If statement p is Paris is in France, then ~ p is 'Paris is not in France'.

|  |  |
| --- | --- |
| p | ~ p |
| T | F |
| F | T |

**2. Conjunction:** It means Anding of two statements. If p, q are two statements, then "p and q" is a compound statement, denoted by p ∧ q and referred as the conjunction of p and q. The conjunction of p and q is true only when both p and q are true. Otherwise, it is false.

|  |  |  |
| --- | --- | --- |
| p | q | p ∧ q |
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

**3. Disjunction:** It means Oring of two statements. If p, q are two statements, then "p or q" is a compound statement, denoted by p ∨ q and referred to as the disjunction of p and q. The disjunction of p and q is true whenever at least one of the two statements is true, and it is false only when both p and q are false.

|  |  |  |
| --- | --- | --- |
| p | q | p ∨ q |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

**4. Implication / if-then (⟶):** An implication p⟶q is the proposition "if p, then q." It is false if p is true and q is false. The rest cases are true.

|  |  |  |
| --- | --- | --- |
| p | q | p ⟶ q |
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | F |

**5. If and Only If (↔):** p ↔ q is bi-conditional logical connective which is true when p and q are same, i.e., both are false or both are true.

|  |  |  |
| --- | --- | --- |
| p | q | p ↔ q |
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | T |

# Basics Of Integrated Circuits

An **electronic circuit** is a group of electronic components connected for a specific purpose.

A simple electronic circuit can be designed easily because it requires few discrete electronic components and connections. However, designing a complex electronic circuit is difficult, as it requires a greater number of discrete electronic components and their connections. It is also time taking to build such complex circuits and their reliability is also less. These difficulties can be overcome with Integrated Circuits.

Integrated Circuit (IC)

If multiple electronic components are interconnected on a single chip of semiconductor material, then that chip is called as an **Integrated Circuit (IC)**. It consists of both active and passive components.

This chapter discusses the advantages and types of ICs.

Advantages of Integrated Circuits

Integrated circuits offer many advantages. They are discussed below −

* **Compact size** − For a given functionality, you can obtain a circuit of smaller size using ICs, compared to that built using a discrete circuit.
* **Lesser weight** − A circuit built with ICs weighs lesser when compared to the weight of a discrete circuit that is used for implementing the same function of IC. using ICs, compared to that built using a discrete circuit.
* **Low power consumption** − ICs consume lower power than a traditional circuit,because of their smaller size and construction.
* **Reduced cost** − ICs are available at much reduced cost than discrete circuits because of their fabrication technologies and usage of lesser material than discrete circuits.
* **Increased reliability** − Since they employ lesser connections, ICs offer increased reliability compared to digital circuits.
* **Improved operating speeds** − ICs operate at improved speeds because of their switching speeds and lesser power consumption.

Types of Integrated Circuits

Integrated circuits are of two types**− Analog Integrated Circuits and Digital Integrated Circuits**.

Analog Integrated Circuits

Integrated circuits that operate over an entire range of continuous values of the signal amplitude are called as **Analog Integrated Circuits.** These are further classified into the two types as discussed here −

* **Linear Integrated Circuits** − An analog IC is said to be Linear, if there exists a linear relation between its voltage and current. IC 741, an 8-pin Dual In-line Package (DIP)op-amp, is an example of Linear IC.
* **Radio Frequency Integrated Circuits** − An analog IC is said to be Non-Linear, if there exists a non-linear relation between its voltage and current. A Non-Linear IC is also called as Radio Frequency IC.

Digital Integrated Circuits

If the integrated circuits operate only at a few pre-defined levels instead of operating for an entire range of continuous values of the signal amplitude, then those are called as **Digital Integrated Circuits**.

In the coming chapters, we will discuss about various Linear Integrated Circuits and their applications.