# Digital Logic Design

Essential Computing (Note-4)

**Number System**

If base or radix of a number system is 'r', then the numbers present in that number system are ranging from zero to r-1. The total numbers present in that number system is 'r'. So, we will get various number systems, by choosing the values of radix as greater than or equal to two.

The following number systems are the most commonly used.

- Decimal Number system
- Binary Number system
- Octal Number system
- Hexadecimal Number system

**Decimal Number System**

The **base** or radix of Decimal number system is **10**. So, the numbers ranging from 0 to 9 are used in this number system. The part of the number that lies to the left of the decimal point is known as integer part. Similarly, the part of the number that lies to the right of the decimal point is known as fractional part.

In this number system, the successive positions to the left of the decimal point having weights of $10^0$, $10^1$, $10^2$, $10^3$ and so on. Similarly, the successive positions to the right of the decimal point having weights of $10^{-1}$, $10^{-2}$, $10^{-3}$ and so on. That means, each position has specific weight, which is **power of base 10**

**Binary Number System**

All digital circuits and systems use this binary number system. The **base** or radix of this number system is **2**. So, the numbers 0 and 1 are used in this number system.

The part of the number, which lies to the left of the **binary point** is known as integer part. Similarly, the part of the number, which lies to the right of the binary point is known as fractional part.

In this number system, the successive positions to the left of the binary point having weights of $2^0$, $2^1$, $2^2$, $2^3$ and so on. Similarly, the successive positions to the right of the binary point having weights of $2^{-1}$, $2^{-2}$, $2^{-3}$ and so on. That means, each position has specific weight, which is **power of base 2**.

**Example**

Consider the **binary number 1101.011**. Integer part of this number is 1101 and fractional part of this number is 0.011. The digits 1, 0, 1 and 1 of integer part have weights of $2^0$, $2^1$, $2^2$, $2^3$ respectively. Similarly, the digits 0, 1 and 1 of fractional part have weights of $2^{-1}$, $2^{-2}$, $2^{-3}$ respectively.

Mathematically, we can write it as

$1101.011 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$

After simplifying the right hand side terms, we will get a decimal number, which is an equivalent of binary number on left hand side.

**Octal Number System**

The **base** or radix of octal number system is **8**. So, the numbers ranging from 0 to 7 are used in this number system. The part of the number that lies to the left of the **octal point** is known as integer part. Similarly, the part of the number that lies to the right of the octal point is known as fractional part.

In this number system, the successive positions to the left of the octal point having weights of $8^0$, $8^1$, $8^2$, $8^3$ and so on. Similarly, the successive positions to the right of the octal point having weights of $8^{-1}$, $8^{-2}$, $8^{-3}$ and so on. That means, each position has specific weight, which is **power of base 8**.

Example

Consider the **octal number 1457.236**. Integer part of this number is 1457 and fractional part of this number is 0.236. The digits 7, 5, 4 and 1 have weights of $8^0$, $8^1$, $8^2$ and $8^3$ respectively. Similarly, the digits 2, 3 and 6 have weights of $8^{-1}$, $8^{-2}$, $8^{-3}$ respectively.

Mathematically, we can write it as

$1457.236 = (1 \times 8^3) + (4 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (2 \times 8^{-1}) + (3 \times 8^{-2}) + (6 \times 8^{-3})$

After simplifying the right hand side terms, we will get a decimal number, which is an equivalent of octal number on left hand side.

**Hexadecimal Number System**

The **base** or radix of Hexadecimal number system is **16**. So, the numbers ranging from 0 to 9 and the letters from A to F are used in this number system. The decimal equivalent of Hexadecimal digits from A to F are 10 to 15.

The part of the number, which lies to the left of the **hexadecimal point** is known as integer part. Similarly, the part of the number, which lies to the right of the Hexadecimal point is known as fractional part.

In this number system, the successive positions to the left of the Hexadecimal point having weights of $16^0$, $16^1$, $16^2$, $16^3$ and so on. Similarly, the successive positions to the right of the Hexadecimal point having weights of $16^{-1}$, $16^{-2}$, $16^{-3}$ and so on. That means, each position has specific weight, which is **power of base 16**.

Example

Consider the Hexadecimal number **1A05.2C4**. Integer part of this number is 1A05 and fractional part of this number is 0.2C4. The digits 5, 0, A and 1 have weights of $16^0$, $16^1$, $16^2$ and $16^3$ respectively. Similarly, the digits 2, C and 4 have weights of $16^{-1}$, $16^{-2}$ and $16^{-3}$ respectively.

Mathematically, we can write it as

$1A05.2C4 = (1 \times 16^3) + (10 \times 16^2) + (0 \times 16^1) + (5 \times 16^0) + (2 \times 16^{-1}) + (12 \times 16^{-2}) + (4 \times 16^{-3})$

In the coding, when numbers or letters are represented by a specific group of symbols, it is said to be that number or letter is being encoded. The group of symbols is called as **code**. The digital data is represented, stored and transmitted as group of bits. This group of bits is also called as **binary code**.

Binary codes can be classified into two types.

- Weighted codes
- Unweighted codes

If the code has positional weights, then it is said to be **weighted code**. Otherwise, it is an unweighted code.

**Binary Codes for Decimal digits**

The following table shows the various binary codes for decimal digits 0 to 9.

| Decimal Digit | 8421 Code | 2421 Code | 84-2-1 Code | Excess 3 Code |
|---|---|---|---|---|
| 0 | 0000 | 0000 | 0000 | 0011 |
| 1 | 0001 | 0001 | 0111 | 0100 |
| 2 | 0010 | 0010 | 0110 | 0101 |
| 3 | 0011 | 0011 | 0101 | 0110 |
| 4 | 0100 | 0100 | 0100 | 0111 |
| 5 | 0101 | 1011 | 1011 | 1000 |
| 6 | 0110 | 1100 | 1010 | 1001 |
| 7 | 0111 | 1101 | 1001 | 1010 |
| 8 | 1000 | 1110 | 1000 | 1011 |
| 9 | 1001 | 1111 | 1111 | 1100 |

We have 10 digits in decimal number system. To represent these 10 digits in binary, we require minimum of 4 bits. But, with 4 bits there will be 16 unique combinations of zeros and ones. Since, we have only 10 decimal digits, the other 6 combinations of zeros and ones are not required.

**8 4 2 1 code**

- The weights of this code are 8, 4, 2 and 1.
- This code has all positive weights. So, it is a positively weighted code.
- This code is also called as **natural BCD** (Binary Coded Decimal) **code**.

**Example**

Let us find the BCD equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the BCD (8421) codes of 7, 8 and 6 are 0111, 1000 and 0110 respectively.

$\therefore (786)_{10} = (011110000110)_{BCD}$

There are 12 bits in BCD representation, since each BCD code of decimal digit has 4 bits.

**2 4 2 1 code**

- The weights of this code are 2, 4, 2 and 1.
- This code has all positive weights. So, it is a **positively weighted code**.
- It is an **unnatural BCD** code. Sum of weights of unnatural BCD codes is equal to 9.

Example

Let us find the 2421 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 2421 codes of 7, 8 and 6 are 1101, 1110 and 1100 respectively.

Therefore, the 2421 equivalent of the decimal number 786 is **110111101100**.

**8 4 -2 -1 code**

- The weights of this code are 8, 4, -2 and -1.
- This code has negative weights along with positive weights. So, it is a **negatively weighted code**.
- It is an **unnatural BCD** code.
- It is a **self-complementing** code.

**Example**

Let us find the 8 4-2-1 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 8 4 -2 -1 codes of 7, 8 and 6 are 1001, 1000 and 1010 respectively.

Therefore, the 8 4 -2 -1 equivalent of the decimal number 786 is **100110001010**.

**Excess 3 code**

- This code doesn't have any weights. So, it is an **un-weighted code**.
- We will get the Excess 3 code of a decimal number by adding three (0011) to the binary equivalent of that decimal number. Hence, it is called as Excess 3 code.
- It is a **self-complementing** code.

**Example**

Let us find the Excess 3 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the Excess 3 codes of 7, 8 and 6 are 1010, 1011 and 1001 respectively.

Therefore, the Excess 3 equivalent of the decimal number 786 is **101010111001**

**Boolean Algebra**

Boolean Algebra is an algebra, which deals with binary numbers & binary variables. Hence, it is also called as Binary Algebra or logical Algebra. The variables used in this algebra are also called as Boolean variables.

The range of voltages corresponding to Logic ‘High’ is represented with ‘1’ and the range of voltages corresponding to logic ‘Low’ is represented with ‘0’.

**Boolean Postulates**

Consider the binary numbers 0 and 1, Boolean variable (x) and its complement (x’). Either the Boolean variable or complement of it is known as **literal**. The four possible **logical OR** operations among these literals and binary numbers are shown below.

$$x + 0 = x$$
$$x + 1 = 1$$
$$x + x = x$$
$$x + x' = 1$$

Similarly, the four possible **logical AND** operations among those literals and binary numbers are shown below.

$$x.1 = x$$
$$x.0 = 0$$
$$x.x = x$$
$$x.x' = 0$$

These are the simple Boolean postulates. We can verify these postulates easily, by substituting the Boolean variable with ‘0’ or ‘1’.

**Basic Laws of Boolean Algebra**

Following are the three basic laws of Boolean Algebra.

- Commutative law
- Associative law
- Distributive law

**Commutative Law**

If any logical operation of two Boolean variables give the same result irrespective of the order of those two variables, then that logical operation is said to be **Commutative**.

$$x + y = y + x$$

$$x.y = y.x$$

The symbol ‘+’ indicates logical OR operation. Similarly, the symbol ‘.’ indicates logical AND operation and it is optional to represent.

**Associative Law**

If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be **Associative**. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x + (y + z) = (x + y) + z$$

$$x.(y.z) = (x.y).z$$

**Distributive Law**

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be **Distributive**. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x.(y + z) = x.y + x.z$$

$$x + (y.z) = (x + y).(x + z)$$

Distributive law obeys for logical OR and logical AND operations. These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with '0' or '1'.

**Theorems of Boolean Algebra**

The following two theorems are used in Boolean algebra.

- Duality theorem
- DeMorgan's theorem

**Duality Theorem**

This theorem states that the **dual** of the Boolean function is obtained by interchanging the logical AND operator with logical OR operator and zeros with ones. For every Boolean function, there will be a corresponding Dual function.

| Group1 | Group2 |
|---|---|
| $x + 0 = x$ | $x.1 = x$ |
| $x + 1 = 1$ | $x.0 = 0$ |
| $x + x = x$ | $x.x = x$ |
| $x + x' = 1$ | $x.x' = 0$ |
| $x + y = y + x$ | $x.y = y.x$ |
| $x + (y + z) = (x + y) + z$ | $x.(y.z) = (x.y).z$ |
| $x.(y + z) = x.y + x.z$ | $x + (y.z) = (x + y).(x + z)$ |

In each row, there are two Boolean equations and they are dual to each other. We can verify all these Boolean equations of Group1 and Group2 by using duality theorem.

**DeMorgan's Theorem**

This theorem is useful in finding the **complement of Boolean function**. It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable.

DeMorgan's theorem with 2 Boolean variables x and y can be represented as

$$(x + y)' = x'.y'$$

The dual of the above Boolean function is

$$(x.y)' = x' + y'$$

Therefore, the complement of logical AND of two Boolean variables is equal to the logical OR of each complemented variable. Similarly, we can apply DeMorgan's theorem for more than 2 Boolean variables also.

**Basic Logic gate**

The basic digital electronic circuit that has one or more inputs and single output is known as Logic gate. Hence, the Logic gates are the building blocks of any digital system. We can classify these Logic gates into the following three categories.

- Basic gates
- Universal gates
- Special gates

**Basic Gates**

We can implement these Boolean functions by using basic gates. The basic gates are AND, OR & NOT gates.

**AND gate**

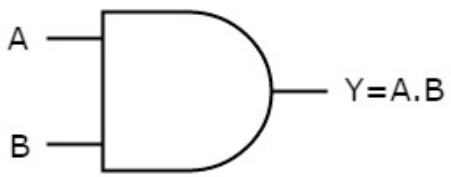
An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '.'.

The following table shows the truth table of 2-input AND gate.

| **A** | **B** | **Y = A.B** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Here A, B are the inputs and Y is the output of two input AND gate. If both inputs are '1', then only the output, Y is '1'. For remaining combinations of inputs, the output, Y is '0'.

The following figure shows the **symbol** of an AND gate, which is having two inputs A, B and one output, Y.



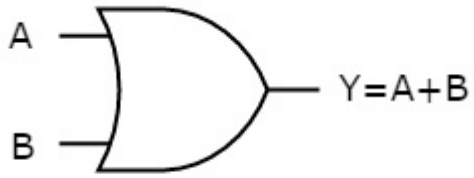
This AND gate produces an output (Y), which is the **logical AND** of two inputs A, B. Similarly, if there are ‘n’ inputs, then the AND gate produces an output, which is the logical AND of all those inputs. That means, the output of AND gate will be ‘1’, when all the inputs are ‘1’.

**OR gate**

An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol ‘+’. The following table shows the **truth table** of 2-input OR gate.

| A | B | Y = A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Here A, B are the inputs and Y is the output of two input OR gate. If both inputs are ‘0’, then only the output, Y is ‘0’. For remaining combinations of inputs, the output, Y is ‘1’. The following figure shows the **symbol** of an OR gate, which is having two inputs A, B and one output, Y.




This OR gate produces an output (Y), which is the **logical OR** of two inputs A, B. Similarly, if there are ‘n’ inputs, then the OR gate produces an output, which is the logical OR of all those inputs. That means, the output of an OR gate will be ‘1’, when at least one of those inputs is ‘1’.

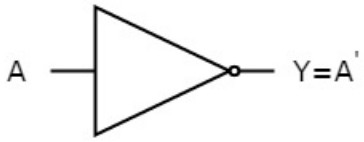
**NOT gate**

A NOT gate is a digital circuit that has single input and single output. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter.

The following table shows the **truth table** of NOT gate.

| A | Y = A’ |
|---|---|
| 0 | 1 |
| 1 | 0 |

Here A and Y are the input and output of NOT gate respectively. If the input, A is ‘0’, then the output, Y is ‘1’. Similarly, if the input, A is ‘1’, then the output, Y is ‘0’. The following figure shows the **symbol** of NOT gate, which is having one input, A and one output, Y.




This NOT gate produces an output (Y), which is the **complement** of input, A.

**Universal gates**

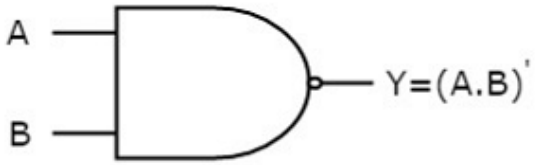
NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function, which is in sum of products form by using NAND gates alone. Similarly, we can implement any Boolean function, which is in product of sums form by using NOR gates alone.

**NAND gate**

NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs. The following table shows the truth table of 2-input NAND gate.

| **A** | **B** | **Y = (A.B)'** |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Here A, B are the inputs and Y is the output of two input NAND gate. When both inputs are '1', the output, Y is '0'. If at least one of the input is zero, then the output, Y is '1'. This is just opposite to that of two input AND gate operation.



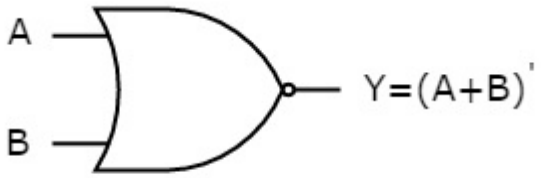

**NOR gate**

NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

The following table shows the **truth table** of 2-input NOR gate

| **A** | **B** | **Y = (A+B)'** |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Here A, B are the inputs and Y is the output. If both inputs are '0', then the output, Y is '1'. If at least one of the input is '1', then the output, Y is '0'. This is just opposite to that of two input OR gate operation. The following figure shows the **symbol** of NOR gate, which is having two inputs A, B and one output, Y.




NOR gate operation is same as that of OR gate followed by an inverter. That's why the NOR gate symbol is represented like that.

**Special Gates**

X-OR & X-NOR gates are called as special gates. Because, these two gates are special cases of OR & NOR gates.

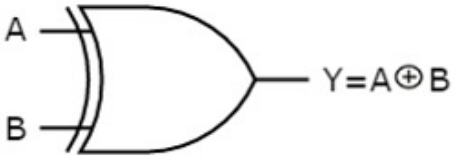
**Ex-OR gate**

The full form of X-OR gate is **Exclusive-OR** gate. Its function is same as that of OR gate except for some cases, when the inputs having even number of ones. The following table shows the **truth table** of 2-input Ex-OR gate.

| **A** | **B** | **Y = A⊕B** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Here A, B are the inputs and Y is the output of two input X-OR gate. The truth table of X-OR gate is same as that of OR gate for first three rows. The only modification is in the fourth row.

Therefore, the output of X-OR gate is '1', when only one of the two inputs is '1'. And it is zero, when both inputs are same. Below figure shows the **symbol** of X-OR gate, which is having two inputs A, B and one output, Y.




The output of Ex-OR gate is '1', when odd number of ones present at the inputs. Hence, the output of Ex-OR gate is also called as an **odd function**.

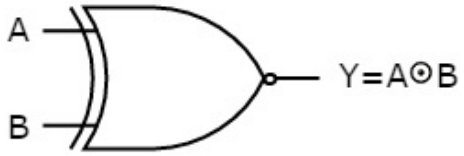
**X-NOR gate**

The full form of X-NOR gate is **Exclusive-NOR** gate. Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones. The following table shows the **truth table** of 2-input Ex-NOR gate.

| **A** | **B** | **Y = A⊙B** |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Here A, B are the inputs and Y is the output. The truth table of X-NOR gate is same as that of NOR gate for first three rows. The only modification is in the fourth row.

Therefore, the output of X-NOR gate is '1', when both inputs are same. And it is zero, when both the inputs are different. The following figure shows the **symbol** of X-NOR gate, which is having two inputs A, B and one output, Y.




The output of X-NOR gate is '1', when even number of ones present at the inputs. Hence, the output of X-NOR gate is also called as an **even function**.

**Combinational Circuit**

Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer.

**Block diagram**


**Half Adder**

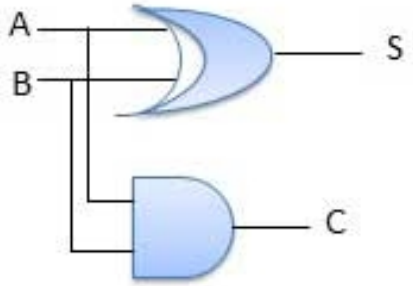
Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

**Block diagram**


**Truth Table**

| Inputs | | Output | |
|---|---|---|---|
| A | B | S | C |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

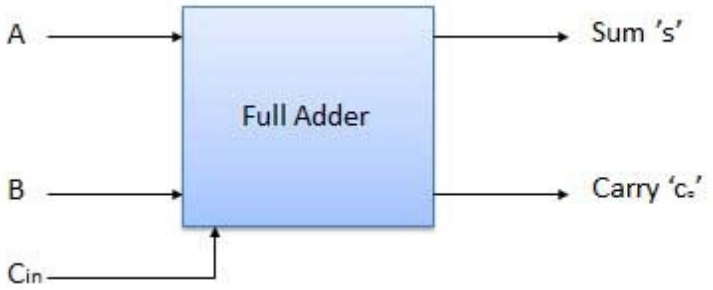
**Circuit Diagram**




**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.

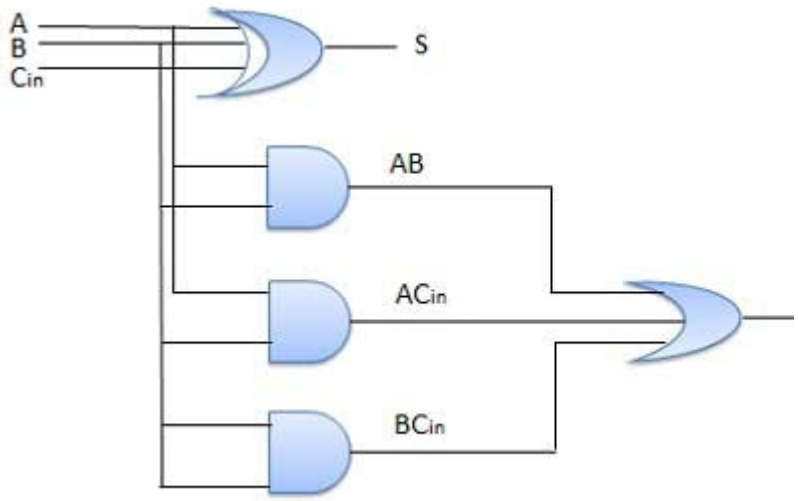
**Block diagram**



**Truth Table**

| Inputs | | | Output | |
|---|---|---|---|---|
| A | B | Cin | S | Co |
| 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 |

**Circuit Diagram**



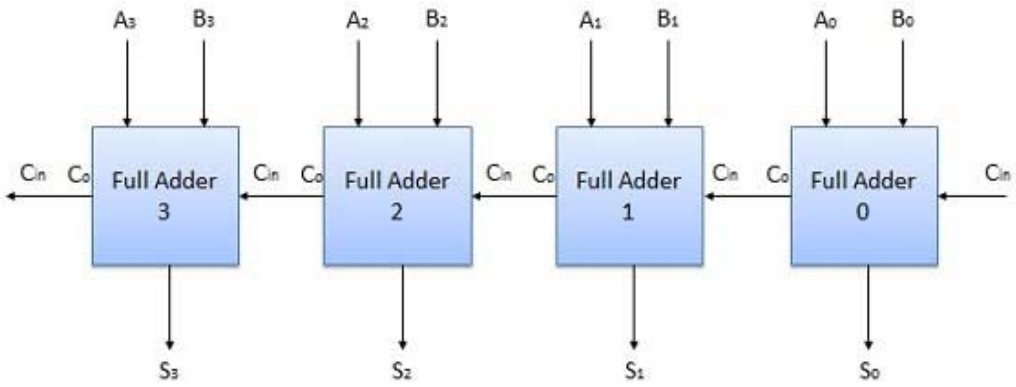

**N-Bit Parallel Adder**

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.

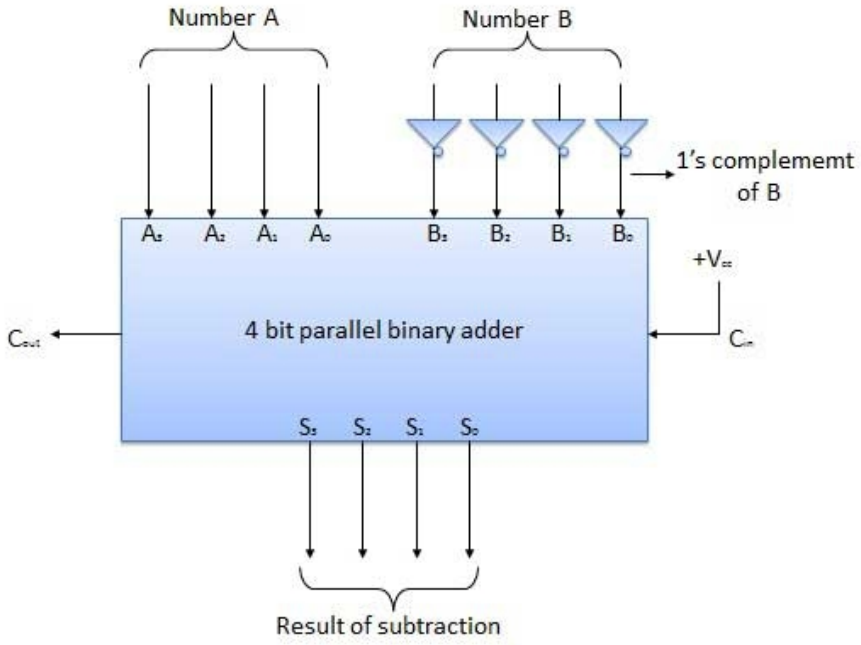
**4 Bit Parallel Adder**

In the block diagram, $A_0$ and $B_0$ represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its $C_{in}$ 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**



**Block diagram**



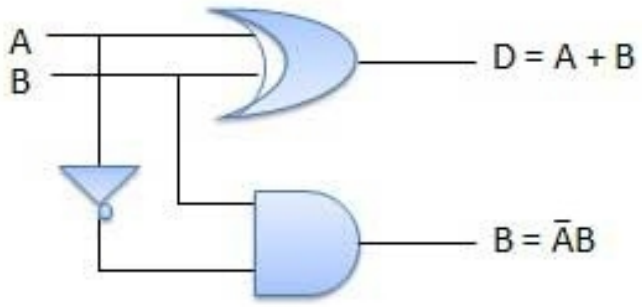

**Half Subtractors**

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**

| Inputs | | Output | |
|---|---|---|---|
| A | B | (A – B) | Borrow |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**Circuit Diagram**



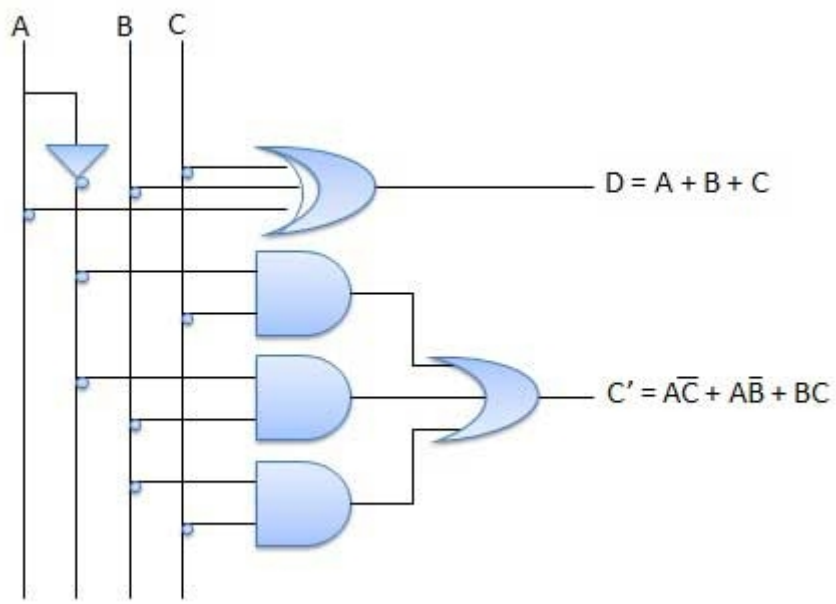

**Full Subtractors**

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**

| Inputs | | | Output | |
|---|---|---|---|---|
| A | B | C | (A-B-C) | C′ |
| 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 |

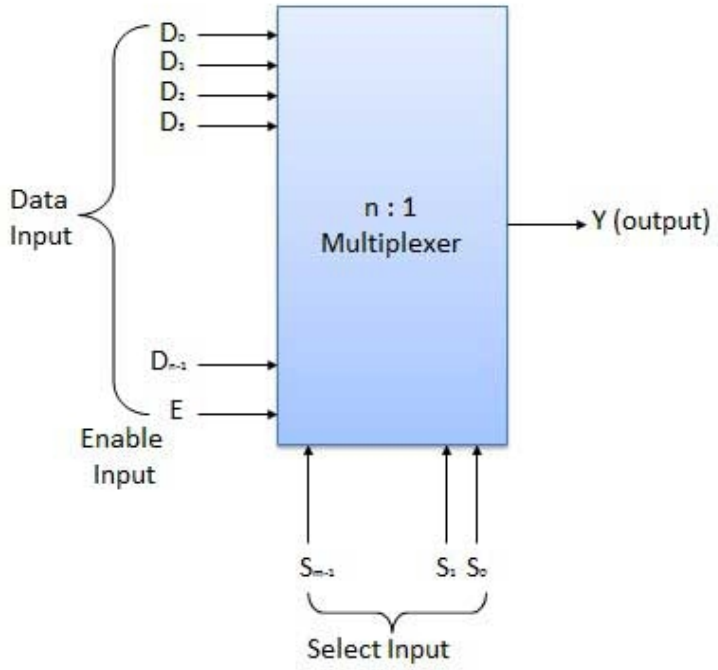
**Circuit Diagram**




**Multiplexers**

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with $2^m = n$. 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.

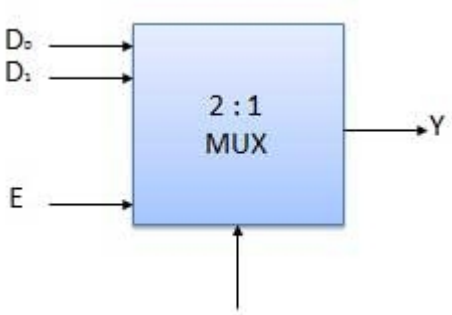
**Block diagram**



Multiplexers come in multiple variations

- 2 : 1 multiplexer
- 4 : 1 multiplexer
- 16 : 1 multiplexer
- 32 : 1 multiplexer

**Block Diagram**




**Truth Table**

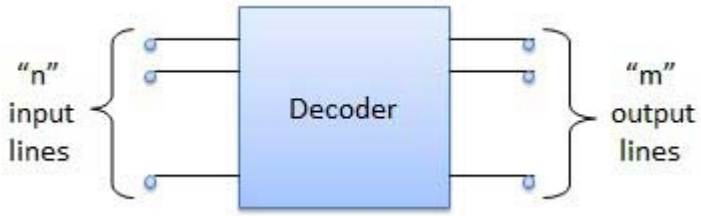
| Enable | Select | Output |
|---|---|---|
| E | S | Y |
| 0 | x | 0 |
| 1 | 0 | $D_0$ |
| 1 | 1 | $D_1$ |

x = Don't care

**Decoder**

A decoder is a combinational circuit. It has n input and to a maximum m = $2^n$ 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**



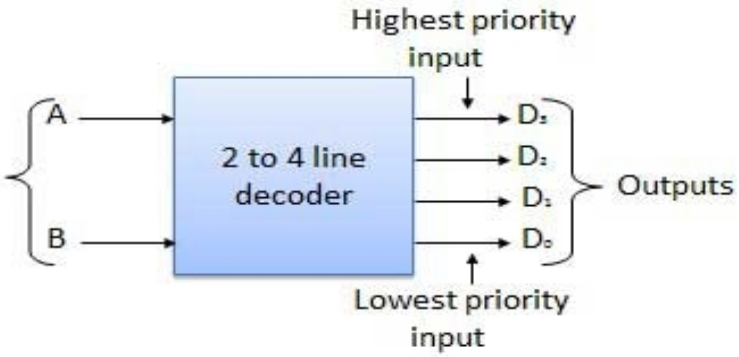

Examples of Decoders are following.

- 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.

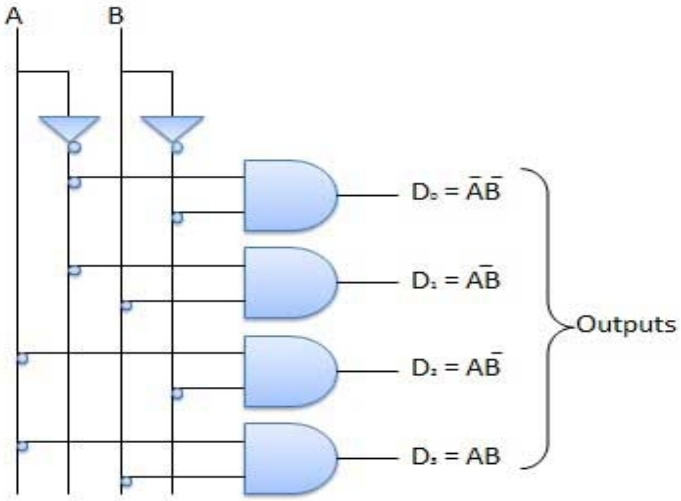
**Block diagram**




**Truth Table**

| Inputs | | Output | | | |
|---|---|---|---|---|---|
| A | B | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

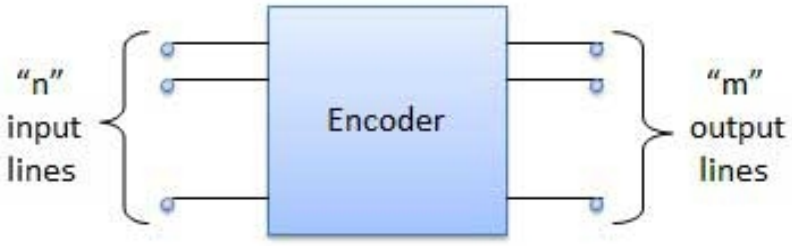
**Logic Circuit**




**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**



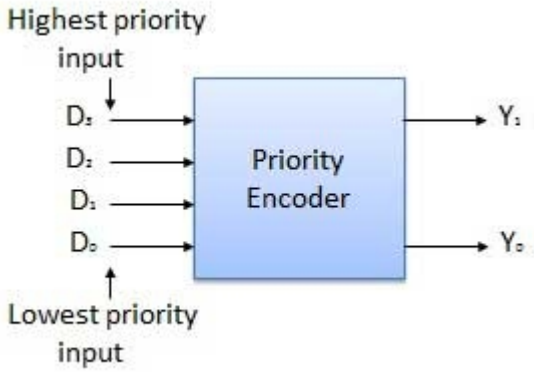
Examples of Encoders are following.

- Priority encoders
- Octal to binary encoder
- Hexadecimal to binary encoder

**Priority Encoder**

This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered. There are four input $D_0$, $D_1$, $D_2$, $D_3$ and two output $Y_0$, $Y_1$. Out of the four input $D_3$ has the highest priority and $D_0$ has the lowest priority. That means if $D_3 = 1$ then $Y_1$ $Y_1 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then $Y_1$ $Y_0 = 10$ irrespective of the other inputs.

**Block diagram**




**Truth Table**

| Highest | Inputs | | Lowest | Outputs | |
|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | $Y_0$ | $Y_1$ |
| 0 | 0 | 0 | 0 | x | x |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | x | 0 | 1 |
| 0 | 1 | x | x | 1 | 0 |
| 1 | x | x | x | 1 | 1 |

**Logic Circuit**