UNIT -1

Introduction to digital Electronics:-

Digital concepts applied to electronics give rise to field of Digital electronics. Digital circuits form the backbone of modern day gadgets like cell phone, digital cameras, GPS displays, etc. since all these devices use information which is digital in nature. Digital systems find application in modern day traffic systems, control systems, weather forecasting systems, and internet, etc.

One of the reasons for widespread application of digital systems is use of Digital computers in applications which provides users with flexibility as any change can be incorporated with the change in system software thus reducing cost which also is an additional advantage. Discrete Information used by digital systems is represented in form of signals which can be classified as Discrete or Continuous signals and systems can be classified as Analog and digital systems.

Advantage of Digital Electronic

1. Ease of Programmability: - The digital systems can be used for different applications by simply changing the program without additional changes in hardware.
2. Reduction in cost of hardware:-
   The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.
3. High Speed:- Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.
4. High Reliability:- Digital systems are highly reliable one of the reasons for that is use of error correction codes.
5. Design to easy:- The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.
6. Result can reproduced easily:
   Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

Positive and Negative Logic:-

There are two types of representations used in digital systems, the positive logic and the negative logic representations.

In positive logic representation Bit 1 represents Logic high and Bit 0 represent a Logic low as shown in fig 2 a and b. High is represented by +5 Volts and low is represented by -5 Volts or 0 Volts

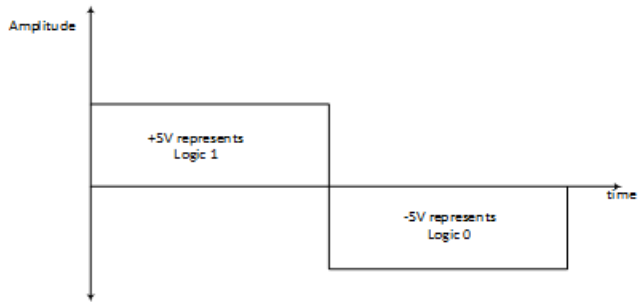


Fig. 2 a. Positive logic representation

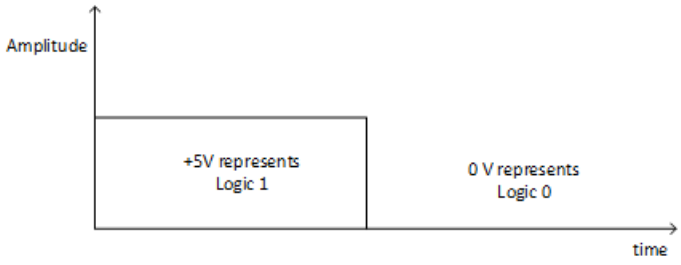


Fig. 2 b. Positive logic representation

In Negative logic representation Bit 1 represents logic low and Bit 0 represents logic high as shown in Fig 3 a and b. In terms of voltage level, bit 1 can be represented as +5V and bit 0 can be represented as 0 V or -5 Volts.

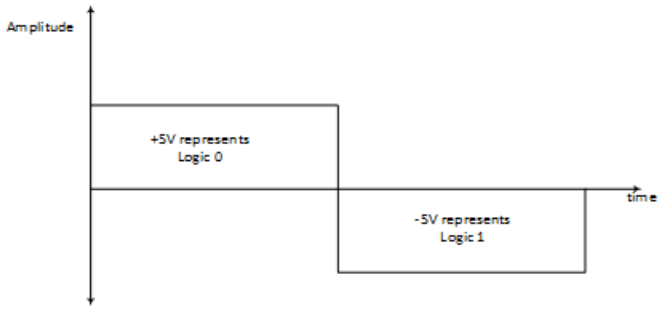


Fig. 3 a. Negative logic representation

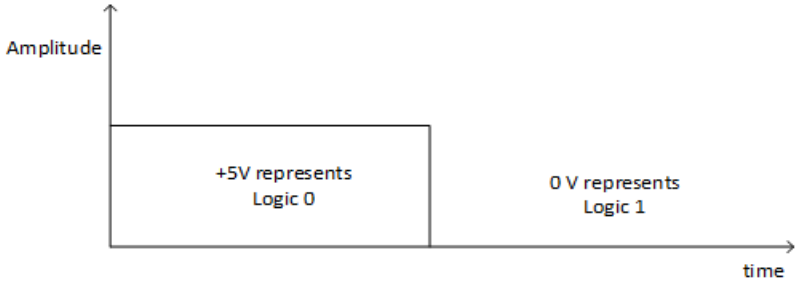


Fig. 3 b. Negative logic representation

Conversion of Number Systems :-

Any base R to Decimal Number system conversion

For conversion of any base R number to Decimal number system each coefficient is multiplied with the corresponding power of R and added to obtain the decimal number.

Binary to Decimal conversion

$(1101.01)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{(-1)} + 1 \times 2^{(-2)} = (13.25)_{10}$

Octal to Decimal conversion

$(431.2)_8 = 4 \times 8^2 + 3 \times 8^1 + 1 \times 8^0 + 2 \times 8^{(-1)} = (281.25)_{10}$

Hexadecimal to Decimal conversion

$(6E9.D8)_{16} = 6 \times 16^2 + 14 \times 16^1 + 9 \times 16^0 + 13 \times 16^{(-1)} + 8 \times 16^{(-2)} = (1769.84375)_{10}$

Base 5 to Decimal conversion

$(421.3)_5 = 4 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 3 \times 5^{(-1)} = (111.6)_{10}$

Likewise any number of base R can be converted to Decimal number system

Decimal to any base R number system conversion

For conversion of Decimal number system to any base R, the decimal number (Integer part) is divided by R and the remainders obtained at each stage are used to represent the base R representation of the decimal number system. For fractional part the digits are multiplied by R and resulting digits integer part is used to define the number.

Decimal to Binary conversion

$$(31.6875)_{10} = (?)_2$$

For Integer part, the number is divided by 2 and the remainders are read in the direction of arrow from down to top corresponding to MSB and LSB as shown below

2	31	Remainder
2	15	1 LSB
2	7	1
2	3	1
	1	1
		1 MSB

$(31)_{10} = (11111)_2$

For fractional part, the digits are multiplied by two and integer part defines the number as shown below, the numbers are now read from top to bottom as shown by arrow, the multiplication is done ideally till the decimal part or fractional part becomes zero or the repetition starts.

0.6875 x 2 = **1**.3750	1
0.375 x 2 = **0**.7500	0
0.7500 x 2 = **1**.500	1
0.500 x 2 = **1**.000	1

$(0.6875)_{10} = (0.1011)_2$

Complete answer is $(31.6875)_{10} = (11111.1011)_2$

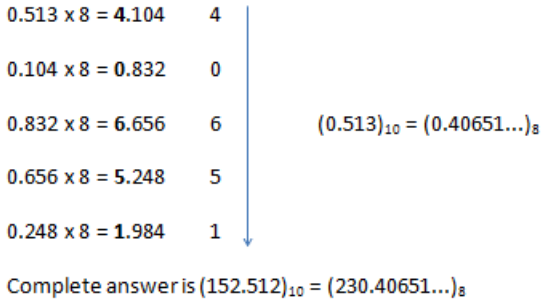
Decimal to Octal

$(152.512)_{10} = (?)_8$

For Integer part, the number is divided by 8 and the remainders are read in the direction of arrow from down to top corresponding to MSB and LSB as shown, this gives $(152)_{10} = (230)_8$

8	152	Remainder
8	19	0 LSB
	2	3
		2 MSB

For fractional part, the digits are multiplied by eight and integer part defines the number as shown below, the numbers are now read from top to bottom as shown by arrow, the multiplication is done ideally till the decimal part or fractional part becomes zero or the repetition starts.

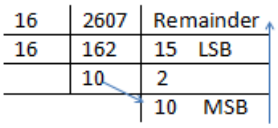


Complete answer is $(152.512)_{10} = (230.40651...)_8$

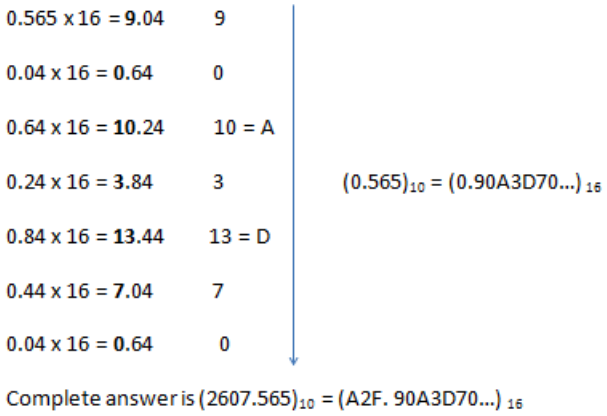
Decimal to Hexadecimal

$(2607.565)^{10} = (?)_{16}$

For Integer part, the number is divided by 16 and the remainders are read in the direction of arrow from down to top corresponding to MSB and LSB as shown, this gives $(2607)_{10} = (A2F)_{16}$



For fractional part, the digits are multiplied by sixteen and integer part defines the number as shown below, the numbers are now read from top to bottom as shown by arrow, the multiplication is done ideally till the decimal part or fractional part becomes zero or the repetition starts.



Likewise any decimal number can be converted into any base R by following the procedure as illustrated

Binary to Octal conversion

Any octal digit corresponds to three binary digits. So to represent any binary number into octal number system the binary digits are grouped as three bits together from LSB to MSB for integer part of number, if groupings of three are not possible then zeroes are padded before the MSB and for fractional part the groupings are done towards right of decimal point and zeroes are padded towards right at the end of the number to make groupings of three. Since each binary number is a positional number and each bit corresponding to weights of powers of two, groupings of three gives equivalent octal number. For example $(110)_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (6)_8$. This can also be understood as 421 coding where each 421 represents the weights of the three positions and 1 at any position is equal to that weight being accounted. For example 101 means weight 4 and weight 1 are accounted, so equivalent is 4+1 =5 in octal representation. Below table gives the binary and its equivalent octal representation

Octal Number	Equivalent Binary number (421 coding)
0	000 (0x4+0x2+0x1)
1	001 (0x4+0x2+1x1)

2	010 (0x4+1x2+0x1)
3	011 (0x4+1x2+1x1)
4	100 (1x4+0x2+0x1)
5	101 (1x4+0x2+1x1)
6	110 (1x4+1x2+0x1)
7	111 (1x4+1x2+1x1)

Table 1 : Octal Binary equivalent

Say a binary number is given as (11010101011111000101.1110000011), to find its equivalent octal representation as per the method detailed above

Octal to Binary conversion:-

For converting any octal number to binary, the binary representation as detailed above in Table 1 is to be written for each octal digit.

$(672.421)_8 = (110\ 111\ 010.\ 100\ 010\ 001)_2$

Binary to Hexadecimal conversion:

Any hexadecimal digit corresponds to four binary digits. So to represent any binary number into hexadecimal number system the binary digits are grouped as four bits together from LSB to MSB for integer part of number, if groupings of four are not possible then zeroes are padded before the MSB and for fractional part the groupings are done towards right of decimal point and zeroes are padded towards right at the end of the number to make groupings of four.

Since each binary number is a positional number and each bit corresponding to weights of powers of two, groupings of four gives equivalent hexadecimal number. For example $(1110)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (14 = E)_{16}$.

This can also be understood as 8421 coding where each 8421 represents the weights of the four positions and 1 at any position is equal to that weight being accounted. For example 0101 means weight 4 and weight 1 are

accounted, so equivalent is 4+1 =5 in hexadecimal representation. Below table gives the binary and its equivalent hexadecimal representation

Hexadecimal Number	Equivalent Binary number (8421 coding)
0	0000 (0X8+0x4+0x2+0x1)
1	0001 (0X8+0x4+0x2+1x1)
2	0010 (0X8+0x4+1x2+0x1)
3	0011 (0X8+0x4+1x2+1x1)
4	0100 (0X8+1x4+0x2+0x1)
5	0101 (0X8+1x4+0x2+1x1)
6	0110 (0X8+1x4+1x2+0x1)
7	0111 (0X8+1x4+1x2+1x1)
8	1000 (1X8+0x4+0x2+0x1)
9	1001 (1X8+0x4+0x2+1x1)
A	1010 (1X8+0x4+1x2+0x1)
B	1011 (1X8+0x4+1x2+1x1)
C	1100 (1X8+1x4+0x2+0x1)
D	1101 (1X8+1x4+0x2+1x1)
E	1110 (1X8+1x4+1x2+0x1)
F	1111 (1X8+1x4+1x2+1x1)

Table 2: Hexadecimal Binary equivalent

Say a binary number is given as (1101010101111000101.1110000011), to find its equivalent hexadecimal representation as per the method detailed above

Hexadecimal to Binary conversion:-

For converting any hexadecimal number to binary, the binary representation as detailed above in Table 2 is to be written for each hexadecimal digit.
$(2C6B.E2)_{16} = (0010\ 1100\ 0110\ 1011.\ 1110\ 0010)_2$

Binary Codes

Digital computers are based on binary number system. Binary Code can represent numbers, characters and operations to be performed.
The group of 1's and 0's in binary numbers is a code representing the decimal number. When a decimal number is a represented by its equivalent binary number, we call it straight binary coding.
<u>BCD (Binary Coded Decimal) Code</u>- If decimal number is represented by its binary equivalent, resulting code is called binary coded decimal. As the largest decimal digit is 9, it can be represented by a minimum of 4 bits.
To illustrate the BCD code, $(234)_{10}$ can be represented as

2	3	4	Decimal
↓	↓	↓	
0010	0011	0100	4 bit binary of each digit

If we combine the individual binary equivalent of each digit, we result in BCD code *i.e.*

$$(234)_{10} = (001000110100)_{BCD}$$

In binary $(234)_{10}$ can be represented as

$$(234)_{10} = (11101010)_2$$

If we reverse the procedure *i.e.* BCD equivalent is to be converted to decimal, then we undergo following steps-

(i) Divide the given BCD code in group of fours.
(ii) Take decimal equivalent of each group.

(iii) Combine the decimal equivalents and we get the resulting decimal equivalent.

Ex. Convert $(001000110100)_{BCD}$ to decimal.

0010 0011 0100 Binary grouped in 4's

↓ ↓ ↓

2 3 4 Decimal Equivalent

$(001000110100)_{BCD} = (234)_{10}$

From the above examples it is clear that BCD code is a weighted code i.e. each digit has a specific weight attached according to their position. So, BCD code is also known as 8-4-2-1 code. The designation 8421 indicates the binary weights of 4 bits (2^3, 2^2, 2^1, 2^0).

The six code combinations- 1010, 1011, 1100, 1101, 1110, 111 are not used in 8421 BCD code. Addition with BCD: To perform addition in BCD, we first add-up in binary format and then perform the BCD conversion i.e. if the result is any of the six combinations, which are not used in BCD, we add 6 to each group of four digits.

Ex. Add 1001 & 0101.

Solution:

1 0 0 1
+ 0 1 0 1
———————
1 1 1 0

This resulting combination is invalid in BCD. So, we add 6 to this result.

1 1 1 0
+ 0 1 1 0
———————
1 0 1 0 0

⇒ 1001 + 0101 = 0001 0100

↓ ↓

1 4

= $(14)_{10}$

<u>Excess-3 Code</u> : It is another BCD code. Excess-3 is a digital code in which decimal digit is converted to BCD and then adding decimal 3 to each decimal digit BCD equivalent

Ex. Convert the given decimal number 159 to Excess-3 Code.

Solution:

1 5 9

0001 0101 1001 BCD Equivalent

+ 0011 0011 0011 Adding decimal 3 to each digit

0100 1000 1100

So, $(159)_{10} = (010010001100)_{Excess-3}$

<u>Gray Code</u> : Gray weight is an unweighted code i.e. there are no specific weights assigned to the bit positions. It is a minimum change code in which only a single bit changes from one number to the next. E.g. Gray code for 3 is 0010 and for 4 is 0110. So there is change in one bit only.Similar to binary, gray code can have any number of bits. Gray code is a reflected code. Table shows 4 bit gray code for decimal and binary numbers.

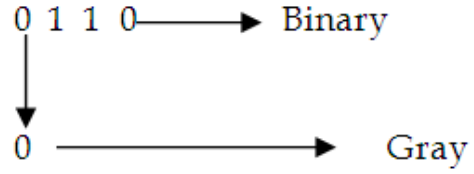
Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

<u>Binary to Gray conversion</u> : To convert a binary number to Gray code we undergo the following steps:

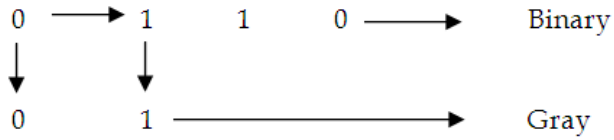
- ⇨ **The MSB (most Significant Bit) of gray Code is same as of corresponding binary number.**
- ⇨ **Going from left to right, each left digit is added to its adjacent right digit and sum is the resulting digit in Gray Code. If there is a carry in the addition, it is discarded.**

E.g., Convert $(0110)_2$ to Gray Code.

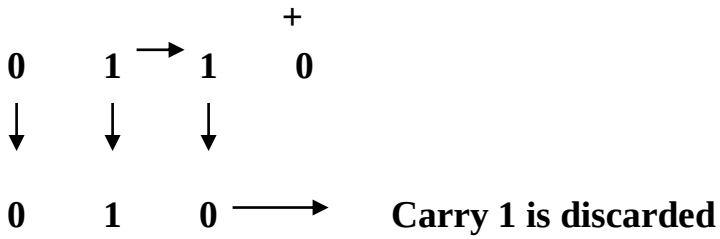
Solution: Step1. MSB remains the same.



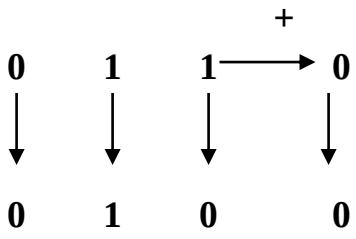
Step 2. Add left bit to the adjacent right bit and take the sum and discard carry if any.



Step 3. Add the next adjacent pair and discard the carry.



Step 4. Repeat step 3.



So, $(0110)_2 = (0101)_{Gray}$

<u>Gray to Binary Conversion</u> : To convert gray code to binary, we undergo following steps-

- ⇨ **The MSB of binary code is same as of corresponding gray code.**
- ⇨ **Going from left to right, each resulting digit of gray code is added to the adjacent right digit of the gray code and sum is the resulting digit in the binary code. Carry, if there is any, is discarded.**

Ex. Convert $(0101)_{Gray}$ to binary.

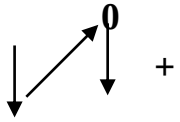
Step 1 : MSB remains the same.

0 1 0 1 Gray

0 Binary

Step 2 : Resulting binary digit added to adjacent gray code digit.

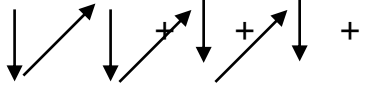
0 1 0 1 Gray



0 1 Binary

Step 3 : Repeating step2 twice we get the resulting binary.

0 1 0 1



0 1 1 0 Carry is discarded.

⇒ $(0101)_{Gray} = (0110)_2$

Power Dissipation

This is the amount of power dissipated in an IC. It is determined by the current, I_{CC}, that it draws from the V_{CC} supply, and is given by $V_{CC} \times I_{CC}$. I_{CC} is the average value of $I_{CC}(0)$ and $I_{CC}(1)$. This power is specified in milliwatts.

Fan-Out

This is the number of similar gates which can be driven by a gate. High fan-out is advantageous because it reduces the need for additional drivers to drive more gates.

Noise Immunity

The input and output voltage levels defined above are shown in Fig. 4.3. Stray electric and magnetic fields may induce unwanted voltages, known as *noise*, on the connecting wires between logic circuits. This may cause the voltage at the input to a logic circuit to drop below V_{IH} or rise above V_{IL} and may produce undesired operation. The circuit's ability to tolerate noise signals is referred to as the *noise immunity*, a quantitative measure of which is called *noise margin*. Noise margins are illustrated in Fig. 4.3.

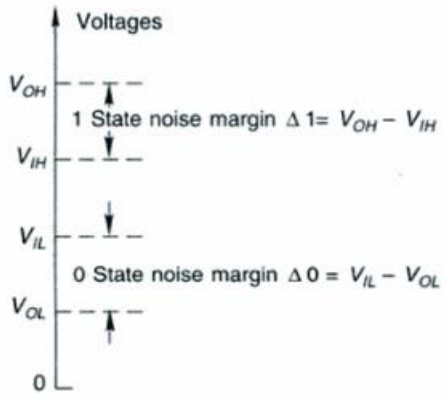


Fig. 4.3
Voltage levels and noise margins of ICs.

The noise margins defined above are referred to as *dc noise margins*. Strictly speaking, the noise is generally thought of as an a.c. signal with amplitude and pulse width. For high speed ICs, a pulse width of a few microseconds is extremely long in comparison to the propagation delay time of the circuit and therefore, may be treated as d.c. as far as the response of the logic circuit is concerned. As the noise pulse width decreases and approaches the propagation delay time of

Half Adder

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value, which are both binary digits.

Half adder accepts two binary digits say, *A*, *B* and produces two outputs Sum (*S*) and Carry (*C*). Table shows the truth table for half adder.

Inputs		Output	
A	***B***	***S***	***C***
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

Table : Truth Table for half adder

The logical expressions for *S* and *C* evaluated from table above are

***S* = *A'B* + *AB'* = *A* ⊕ B**

C* = *AB

The logical realization of half adder is shown in fig. below

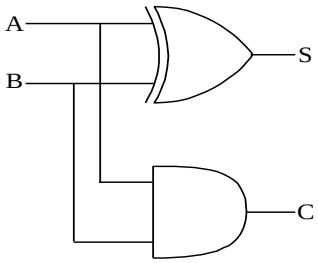


Fig.: Logic Circuit for half-adder.

The drawback of this circuit is that in case of a multibit addition, it cannot include a carry. As half adder has two outputs, the sum *S* and carry *C*, *C* is the most significant of these two outputs.

Full Adder

The second basic category of adder is the full-adder. This combinational circuit performs the arithmetic addition of three input bits. The difference between the full- and the half-adder is the ability of the former to handle input carries (*CI*). A full adder has three inputs *A*, *B*, and a carry in *CI*, such that multiple adders can be used to add larger numbers. To remove ambiguity between the input and output carry lines, the carry in is labeled *CI* while the carry out is labeled *C*. Table shows the truth table for full adder

Input			Output	
A	*B*	*CI*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table: Truth table for Full adder

The logical expression for *S* and *C* can be derived from the truth table using *K*-Map 1 and *K*-Map 2.

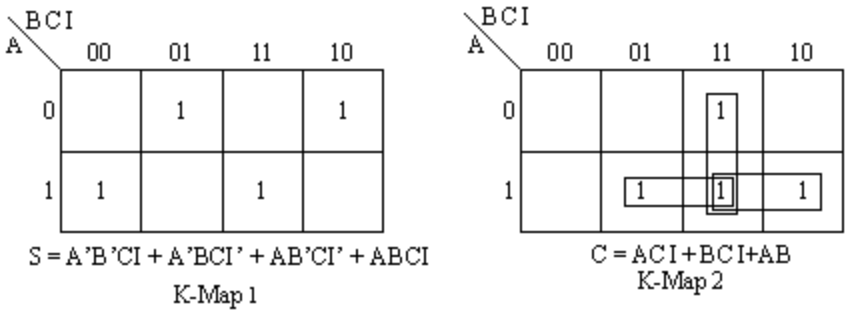


Fig. *K*-Maps for Sum and carry.

The logic circuit for *S* and *C* are shown in fig (a) and fig. (b) respectively.

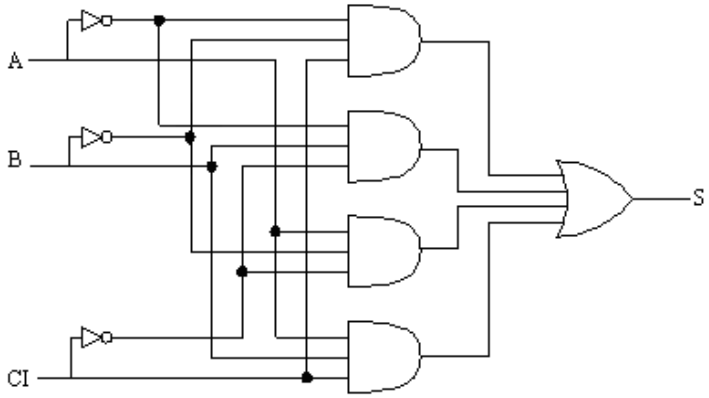


Fig (a) : Logic Circuit for SUM (*S*).

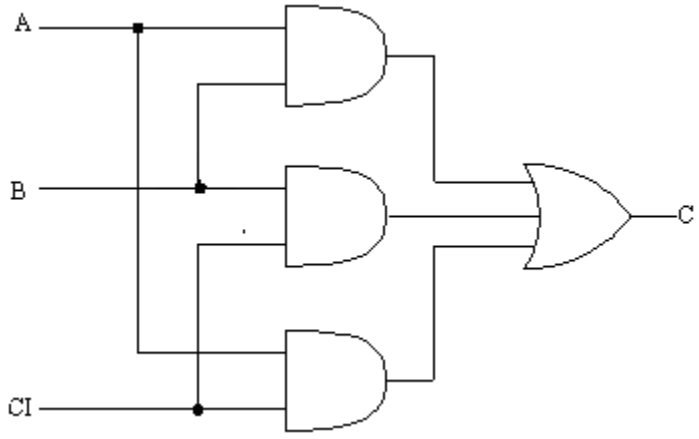


Fig (b) : Logic Circuit for CARRY (*C*).

The expression for *S* and *C* can be further simplified as :

S = A'B'CI + A'BCI' + AB'CI' + ABCI

= *AB' CI'+ A'BCI' + ABCI + A'B'CI*

= *CI'* (*AB'+ A'B*) + *CI* (*AB + A'B'*)

but, *AB + A'B'* = (*AB' + A'B*)'

So, *S* = *CI'* (*AB' + A' B*) + *CI* (*AB' + A' B*)'

= (*A* ⊕ *B* ⊕ *CI*)

The logic circuit for above simplified equation is shown in fig..

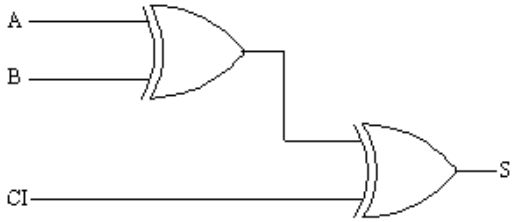


Fig: Logic Circuit for *S* = (*A* ⊕ *B* ⊕ *CI*).

The output of the full-adder is the two bit arithmetic sum of three 1-bit numbers. *A* full adder can be constructed from two half adders by connecting *A* and *B* to the input of one half adder, connecting the sum from that to an input to the second adder, connecting *CI* to the other input and *OR* the two carry outputs. Equivalently, *S* could be made the three-bit x or of *A*, *B*, and *CI* and *C* could be made the three-bit majority function of *A*, *B*, and *CI*. Fig shows logic circuit for constructing full adder from two half adders.

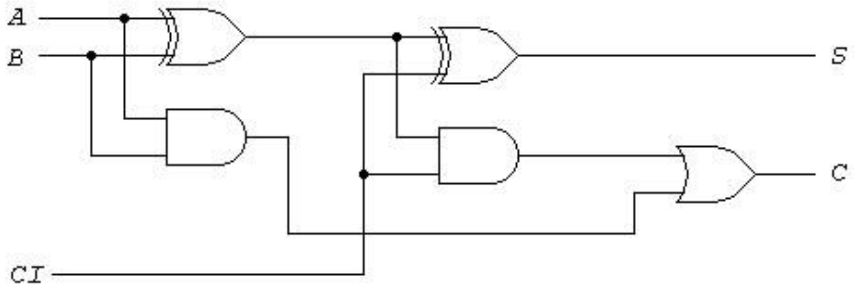


Fig: Full Adder using two half adders.

Multiplexers

Multiplexer is abbreviated as MUX. It is a digital device that selects one of the several input signals and passes it on the output. Because of this reason, it is also known as data selector. The input selected is controlled by a set of select inputs. Fig shows the functional block diagram of multiplexer.

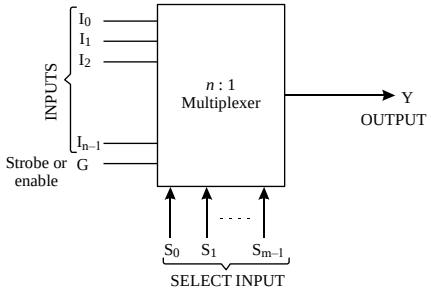


Fig DIGITAL MULTIPLEXER (Block Diagram of *N* : 1 MUX).

If we have *n* input lines then for connection to the output, a set of *m* select inputs is required; where $2^m = n$. Depending upon the digital code applied at the select inputs one out of *n* data sources is selected and transmitted to a single output channel. Normally, a strobe (or enable) input (*G)* is incorporated which helps in cascading and it is generally active-low, which means it performs its intended operation when it is LOW.

Now let us consider an example of 4 : 1 MUX. The block diagram for 4 : 1 MUX is shown in fig.

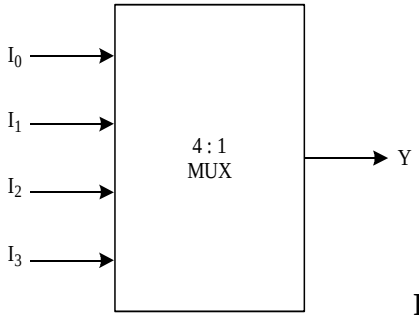


Fig. Block Diagramof 4 : 1 MUX.

As shown in fig it will have 4 inputs *i.e.* I_0, I_1, I_2, I_3; 1 output *i.e.* *Y* and two selection lines as $2^2 = 4$ *i.e.* S_1, S_0. Table lists the input-to-output path for each possible bit combination of the select lines.

Table: Truth Table of a 4 : 1 multiplexer

Select inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1

1	**0**	I_2
1	**1**	I_3

From the above truth table the output *Y* can be expressed as :

$$Y = I_0 \bullet \bar{S}_1 \bullet \bar{S}_0 + I_1 \bullet \bar{S}_1 \bullet S_0 + I_2 \bullet S_1 \bullet \bar{S}_0 + I_3 \bullet S_1 \bullet S_0$$

The logic circuit for the above equation is realized in fig.

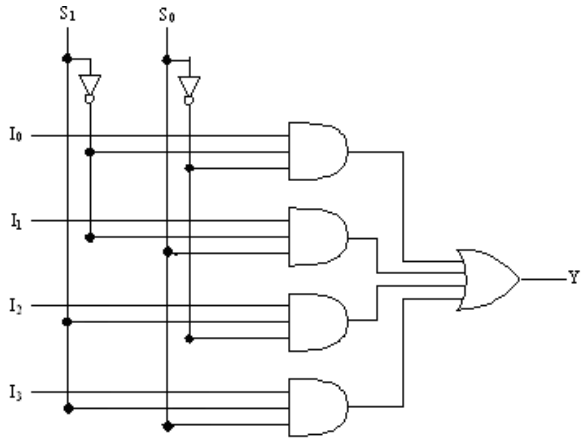


Fig : Logic circuit for 4:1 MUX.

We can realize MUX using universal gates. The 4 : 1 MUX shown in fig. above can be realized using Enable or Strobe signal (*G*) and NAND gates as shown in fig. below

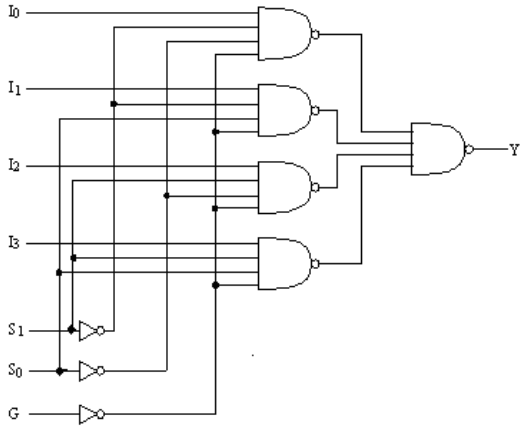


Fig : Logic circuit for 4 : 1 MUX using Universal gates and Enable.

We can design large MUX using two or more MUX *e.g.* We can design a 32 : 1 MUX using two 16 : 1 MUX as shown in fig.

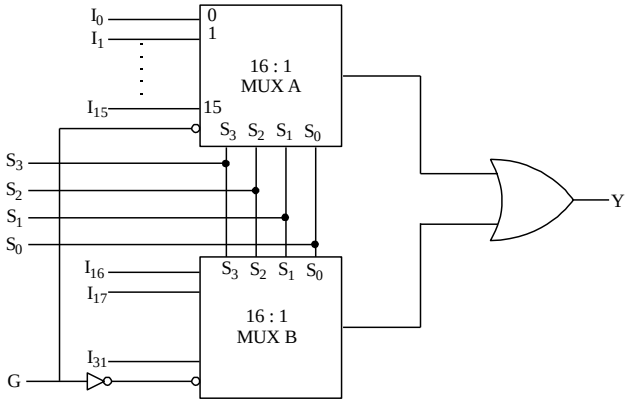


Fig: 32 : 1 MUX using two 16 : 1 MUX.

In this realization, when strobe signal *G* = 0 then MUX *A* is enabled and one of the inputs from $I_0 - I_{15}$ is selected as an output. When *G* = 1 then MUX *B* is enabled and one of the inputs from $I_{16} - I_{31}$ is selected as an output.

Advantages of use of multiplexers

1. **Simplification of logic expression is not required.**
2. **It minimizes the IC package count.**
3. **Logic design is simplified.**

Combinational Logic Design Using Multiplexers

The multiplexing function discussed above can be conveniently used as logic element in the design of combinational circuits. Standard ICs are available for 2 : 1, 4 : 1, 8 : 1 and 16 : 1 multiplexers.

Available multiplexer ICs are listed in table.

IC No.	Description	Output
74157	Quad 2 : 1 Multiplexer	Same as input
74158	Quad 2 : 1 Multiplexer	Inverted input
74153	Dual 4 : 1 Multiplexer	Same as input
74352	Dual 4 : 1 Multiplexer	Inverted input
74151A	8 : 1 Multiplexer	Complementary outputs
74152	8 : 1 Multiplexer	Inverted input
74150	16 : 1 Multiplexer	Inverted input

Table : List of available multiplexer ICs

Ex : Use a 4 line to 1 line MUX to implement the function shown in the following truth table ($Y = A'B' + AB$).

Sol:

A	B	Y	
0	0	1	$= I_0$
0	1	0	$= I_1$
1	0	0	$= I_2$
1	1	1	$= I_3$

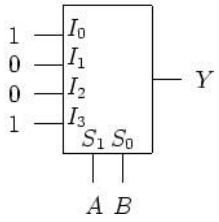


Fig. A 4-line to 1-line MUX implementation of a function of 2 variables.

Simply connecting $I_0 = 1$, $I_1 = 0$, $I_2 = 0$, $I_3 = 1$ and the inputs A and B to the S_1 and S_0 selector inputs of the 4-line to 1-line MUX implement this truth table, as shown in figure .

Demultiplexers

The opposite of the multiplexer circuit, logically enough, is the demultiplexer. It is abbreviated as DEMUX. This circuit receives information on a single line and transmits this information on one of various possible output lines. It consists of one input line, n selection lines and m output lines where $2^n = m$. The block diagram of demultiplexer is shown in fig.

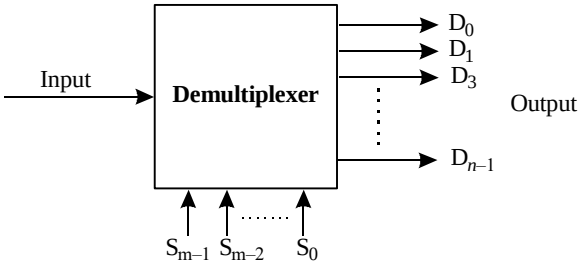


Fig. Block Diagram of Demultiplexer.

The select input determines to which output the data input will be transmitted. The number of output lines is n and the number of select lines is m, where $n = 2^m$. The data input line is to be connected to logic 1 level. This is device is available as an MSI IC and can conveniently be used for the design of combinational circuits. These devices are available as 2-line-to-4 line, 3-line-to-8-line, and 4-line-to-16-line-decoders. The output of most of these devices are active-low, also there is an active-low enable/data input terminal available.

Available demultiplexer ICs

IC No.	Description	Output
74139	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Inverted input
74155	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	1Y-Inverted input 2Y-same as input
74156	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Open-collector 1Y-Inverted input 2Y-same as input
74138	1:8 Demultiplexer (3-line-to-8-line decoder)	Inverted input
74154	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input
74159	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input Open-collector

We can design large Demux cascading two or more Demux *e.g.* We can design a 1 : 32 DEMUX using two 1 : 16 DEMUX as shown in fig.

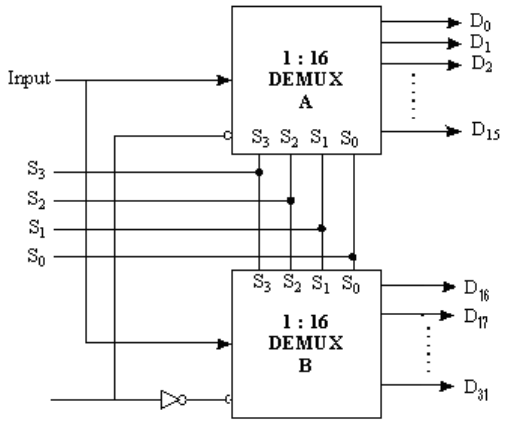


Fig : DEMUX using two 1 : 16 DEMUX.