Module 1:

Number Systems and Codes: Binary Number system, Binary to decimal, decimal to binary, hexa decimal, ASCII code, Excess,3 Code, Gray code. Digital Logic: The Basic Gates, NOT, OR, AND, Universal Logic Gates, NOR, NAND.

Combinatorial Logic Circuits: Boolean Laws and Theorems. , Sum of Products method ,Truth table to Karnaugh Map – Pairs, Quads, Octets – Don't Care Conditions Product of sums method , Product of sums Simplifications

Combinational Circuits –Analysis and Design Procedures, Binary Adder, Subtractor, Decimal Adder, Magnitude Comparator, Decoder, Encoder, Multiplexers, Demultiplexers

Number Systems and Codes

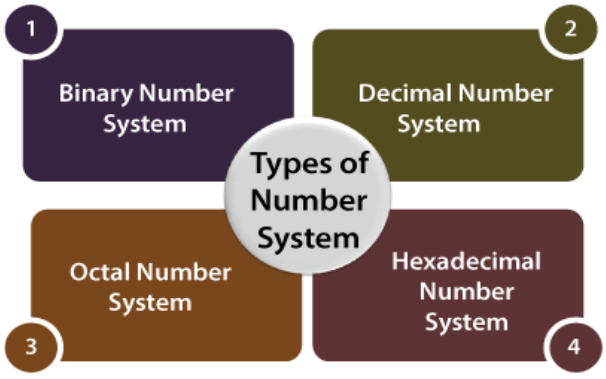
Number Systems

In a digital system, the system can understand only the optional number system. In these systems, digits symbols are used to represent different values, depending on the index from which it settled in the number system.

Types of Number System

In the digital computer, there are various types of number systems used for representing information.

1. Binary Number System
2. Decimal Number System
3. Hexadecimal Number System
4. Octal Number System



Binary Number system

- A binary number system is used in the digital computers. In this number system, it carries only two digits, either 0 or 1.
- There are two types of electronic pulses present in a binary number system.
  - **i.** the absence of an electronic pulse representing '0'.
  - **ii.** the presence of electronic pulse representing '1'.
- Each digit is known as a bit.
- A four-bit collection (1101) is known as a nibble.
- A collection of eight bits (11001010) is known as a byte.

Characteristics:

1. It holds only two values, i.e., either 0 or 1.
2. It is also known as the base 2 number system.
3. The position of a digit represents the 0 power of the base(2). Example: 2^0
4. The position of the last digit represents the x power of the base(2). Example: 2^x, where x represents the last position, i.e., 1

Decimal Number System

- The decimal number system contains ten digits from 0 to 9(base 10).
- Here, the successive place value or position, left to the decimal point holds units, tens, hundreds, thousands, and so on.
- The position in the decimal number system specifies the power of the base (10). The 0 is the minimum value of the digit, and 9 is the maximum value of the digit.
- For example, the decimal number 2541 consist of the digit 1 in the unit position, 4 in the tens position, 5 in the hundreds position, and 2 in the thousand positions and the value will be written as:

$$(2\times1000) + (5\times100) + (4\times10) + (1\times1)$$
$$(2\times10^3) + (5\times10^2) + (4\times10^1) + (1\times10^0)$$
$$2000 + 500 + 40 + 1$$
$$254$$

Octal Number System

- The octal number system has base 8(means it has only eight digits from 0 to 7).
- With the help of only three bits, an octal number is represented. Each set of bits has a distinct value between 0 and 7.

Characteristics:

1. An octal number system carries eight digits starting from 0, 1, 2, 3, 4, 5, 6, and 7.
2. It is also known as the base 8 number system.
3. The position of a digit represents the 0 power of the base(8). Example: 8^0
4. The position of the last digit represents the x power of the base(8). Example: 8^x, where x represents the last position, i.e., 1

Number	Octal Number
0	000
1	001
2	010
3	011

Examples:

$(273)_8$, $(5644)_8$, $(0.5365)_8$, $(1123)_8$, $(1223)_8$.

4	100
5	101
6	110
7	111

Hexadecimal Number System

- The number system has a base of 16 means there are total 16 symbols(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) used for representing a number.
- The single-bit representation of decimal values10, 11, 12, 13, 14, and 15 are represented by A, B, C, D, E, and F.
- Only 4 bits are required for representing a number in a hexadecimal number.
- Each set of bits has a distinct value between 0 and 15.

Characteristics:

1. It has ten digits from 0 to 9 and 6 letters from A to F.
2. The letters from A to F defines numbers from 10 to 15.
3. It is also known as the base 16number system.
4. In hexadecimal number, the position of a digit represents the 0 power of the base(16). Example: 16^0
5. In hexadecimal number, the position of the last digit represents the x power of the base(16). Example: 16^x, where x represents the last position, i.e., 1

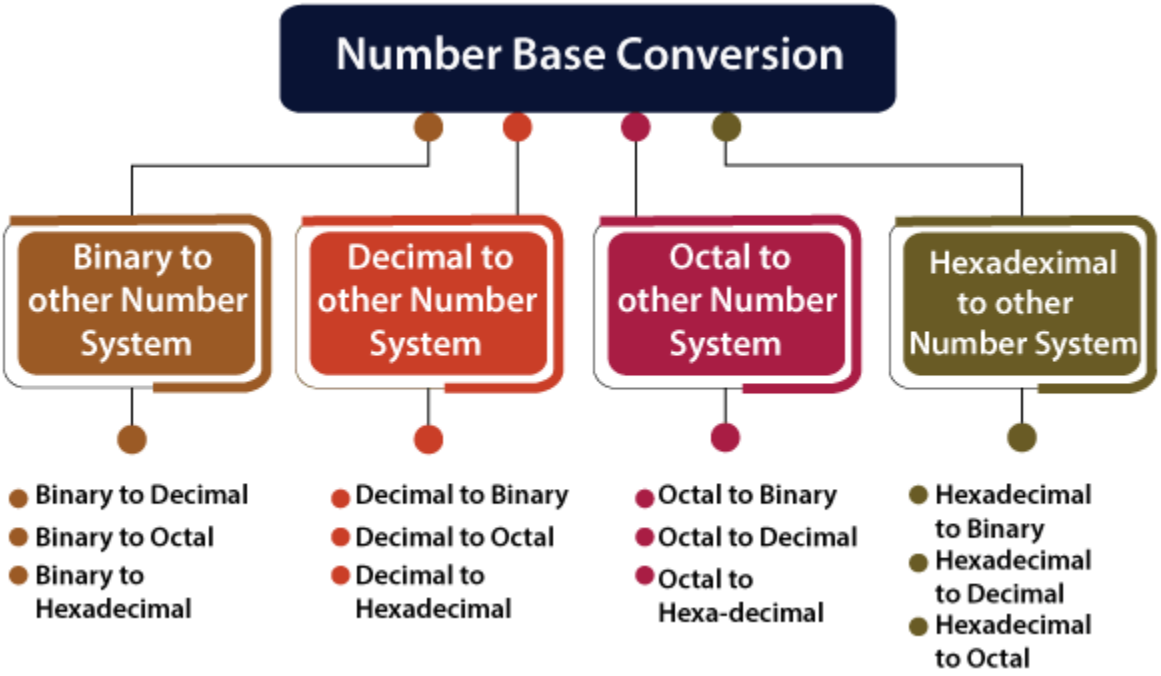
Binary Number	Hexadecimal Number
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Examples:

$(FAC2)_{16}$, $(564)_{16}$, $(0ABD5)_{16}$, $(1123)_{16}$, $(11F3)_{16}$.

Number Base Conversion

1. Binary to other Number Systems.
2. Decimal to other Number Systems.
3. Octal to other Number Systems.
4. Hexadecimal to other Number Systems.



➢ Binary to Decimal Conversion

- The process starts from multiplying the bits of binary number with its corresponding positional weights. And lastly, we add all those products.

Example 1: $(10110.001)_2$

We multiplied each bit of $(10110.001)_2$ with its respective positional weight, and last we add the products of all the bits with its weight.

$\mathbf{(10110.001)_2}=(1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0)+(0\times2^{-1})+(0\times2^{-2})+(1\times2^{-3})$

$\mathbf{(10110.001)_2}=(1\times16)+(0\times8)+(1\times4)+(1\times2)+(0\times1)+(0\times1/2)+(0\times1/4)+(1\times1/8)$

$\mathbf{(10110.001)_2}=16+0+4+2+0+0+0+0.125$
$\mathbf{(10110.001)_2}=\mathbf{(22.125)_{10}}$

➢ Decimal to other Number System

The decimal number can be an integer or floating-point integer. When the decimal number is a floating-point integer, then we convert both part (integer and fractional) of the decimal number in the isolated form(individually). There are the following steps that are used to convert the decimal number into a similar number of any base **'r'**.

1. In the first step, we perform the division operation on integer and successive part with base **'r'**. We will list down all the remainders till the quotient is zero. Then we find out the remainders in reverse order for getting the integer part of the equivalent number of base **'r'**. In this, the least and most significant digits are denoted by the first and the last remainders.
2. In the next step, the multiplication operation is done with base **'r'** of the fractional and successive fraction. The carries are noted until the result is zero or when the required number of the equivalent digit is obtained. For getting the fractional part of the equivalent number of base **'r'**, the normal sequence of carrying is considered.

Decimal to Binary Conversion

For converting decimal to binary, there are two steps required to perform, which are as follows:

1. In the first step, we perform the division operation on the integer and the successive quotient with the base of binary(2).
2. Next, we perform the multiplication on the integer and the successive quotient with the base of binary(2).

Example 1: $(152.25)_{10}$

Step 1:

Divide the number 152 and its successive quotients with base 2.

Operation	Quotient	Remainder
152/2	76	0 (LSB)
76/2	38	0
38/2	19	0
19/2	9	1
9/2	4	1
4/2	2	0
2/2	1	0
1/2	0	1(MSB)

$(152)_{10} = (10011000)_2$

Step 2:

Now, perform the multiplication of 0.27 and successive fraction with base 2.

Operation	Result	carry
0.25×2	0.50	0
0.50×2	0	1

$(0.25)_{10} = (.01)_2$

- ## Hexa-decimal to Decimal Conversion

The process of converting hexadecimal to decimal is the same as binary to decimal. The process starts from multiplying the digits of hexadecimal numbers with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from hexadecimal to decimal.

Example 1: $(152A.25)_{16}$

Step 1:

We multiply each digit of **152A.25** with its respective positional weight, and last we add the products of all the bits with its weight.

$(152A.25)_{16} = (1\times16^3)+(5\times16^2)+(2\times16^1)+(A\times16^0)+(2\times16^{-1})+(5\times16^{-2})$

$(152A.25)_{16} = (1\times4096)+(5\times256)+(2\times16)+(10\times1)+(2\times16^{-1})+(5\times16^{-2})$

$(152A.25)_{16} = 4096+1280+32+10+(2\times1/16)+(5\times1/256)$

$(152A.25)_{16} = 5418+0.125+0.125$

$(152A.25)_{16} = 5418.14453125$

So, the decimal number of the hexadecimal number 152A.25 is **5418.14453125**

- ## Hexadecimal to Binary Conversion

The process of converting hexadecimal to binary is the reverse process of binary to hexadecimal. We write the four bits binary code of each hexadecimal number digit.

Example 1: $(152A.25)_{16}$

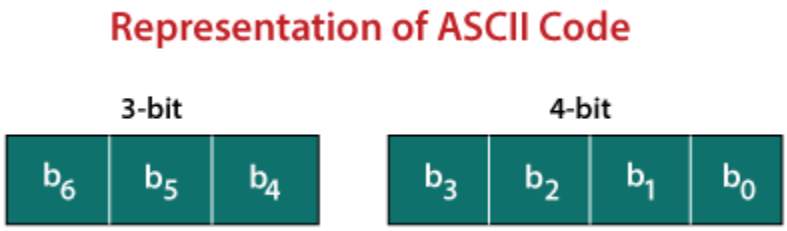
We write the four-bit binary digit for 1, 5, A, 2, and 5.

$(152A.25)_{16} = (0001\ 0101\ 0010\ 1010.0010\ 0101)_2$

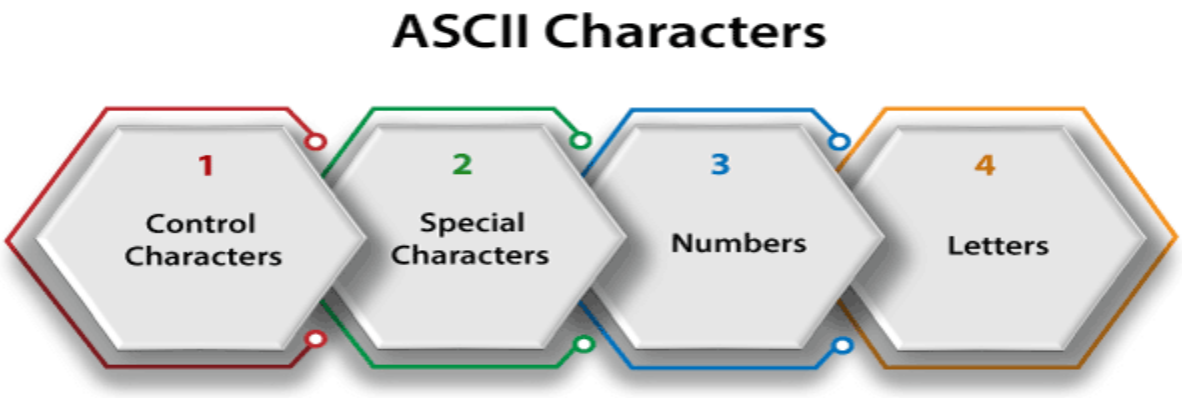
So, the binary number of the hexadecimal number 152.25 is **$(1010100101010.00100101)_2$**

ASCII Code

- The ASCII stands for American Standard Code for Information Interchange.
- The ASCII code is an alphanumeric code used for data communication in digital computers.
- The ASCII is a 7-bit code capable of representing 2^7 or 128 number of different characters.
- The ASCII code is made up of a three-bit group, which is followed by a four-bit code.



- The ASCII code starts from 00h to 7Fh. In this, the code from 00h to 1Fh is used for control characters, and the code from 20h to 7Fh is used for graphic symbols.
- The 8-bit code holds ASCII, which supports 256 symbols where math and graphic symbols are added.
- The range of the extended ASCII is 80h to FFh.



Control Characters

The non-printable characters used for sending commands to the PC or printer are known as control characters. We can set tabs, and line breaks functionality by this code. The control characters are based on telex technology. Nowadays, it's not so much popular in use. The character from 0 to 31 and 127 comes under control characters.

Special Characters

All printable characters that are neither numbers nor letters come under the special characters. These characters contain technical, punctuation, and mathematical characters with space also. The character from 32 to 47, 58 to 64, 91 to 96, and 123 to 126 comes under this category.

Numbers Characters

This category of ASCII code contains ten Arabic numerals from 0 to 9.

Letters Characters

In this category, two groups of letters are contained, i.e., the group of uppercase letters and the group of lowercase letters. The range from 65 to 90 and 97 to 122 comes under this category.

Excess-3 Code

The excess-3 code is also treated as **XS-3 code**. The excess-3 code is a non-weighted and self-complementary BCD code used to represent the decimal numbers. This code has a biased representation. This code plays an important role in arithmetic operations because it resolves deficiencies encountered when we use the 8421 BCD code for adding two decimal digits whose sum is greater than 9. The Excess-3 code uses a special type of algorithm, which differs from the binary positional number system or normal non-biased BCD.

We can easily get an excess-3 code of a decimal number by simply adding 3 to each decimal digit. And then we write the 4-bit binary number for each digit of the decimal number. We can find the excess-3 code of the given binary number by using the following steps:

1. We find the decimal number of the given binary number.
2. Then we add 3 in each digit of the decimal number.
3. Now, we find the binary code of each digit of the newly generated decimal number.

We can also add 0011 in each 4-bit BCD code of the decimal number for getting excess-3 code.

The Excess-3 code for the decimal number is as follows:

Decimal Digit	BCD Code	Excess-3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Gray Code

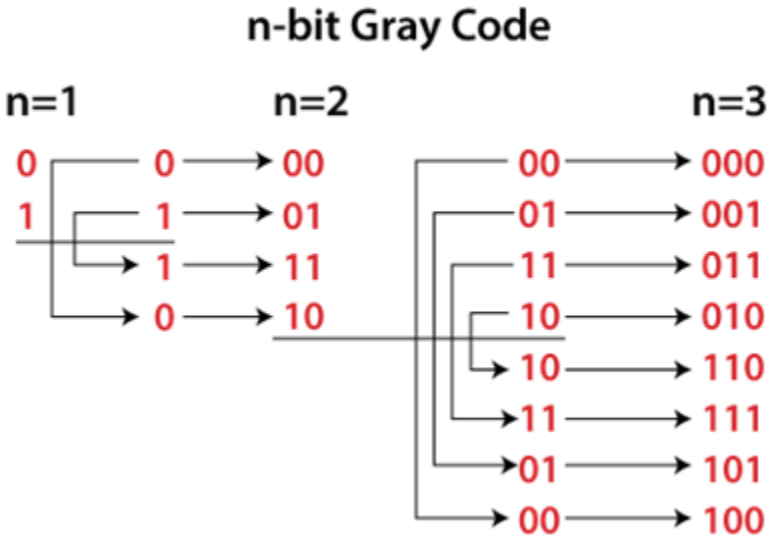
The **Gray Code** is a sequence of binary number systems, which is also known as **reflected binary code**. The reason for calling this code as reflected binary code is the first N/2 values compared with those of the last N/2 values in reverse order. In this code, two consecutive values are differed by one bit of binary digits. Gray codes are used in the general sequence of hardware-generated binary numbers. These numbers cause ambiguities or errors when the transition from one number to its successive is done. This code simply solves this problem by changing only one bit when the transition is between numbers is done.

The gray code is a very light weighted code because it doesn't depend on the value of the digit specified by the position. This code is also called a cyclic variable code as the transition of one value to its successive value carries a change of one bit only.

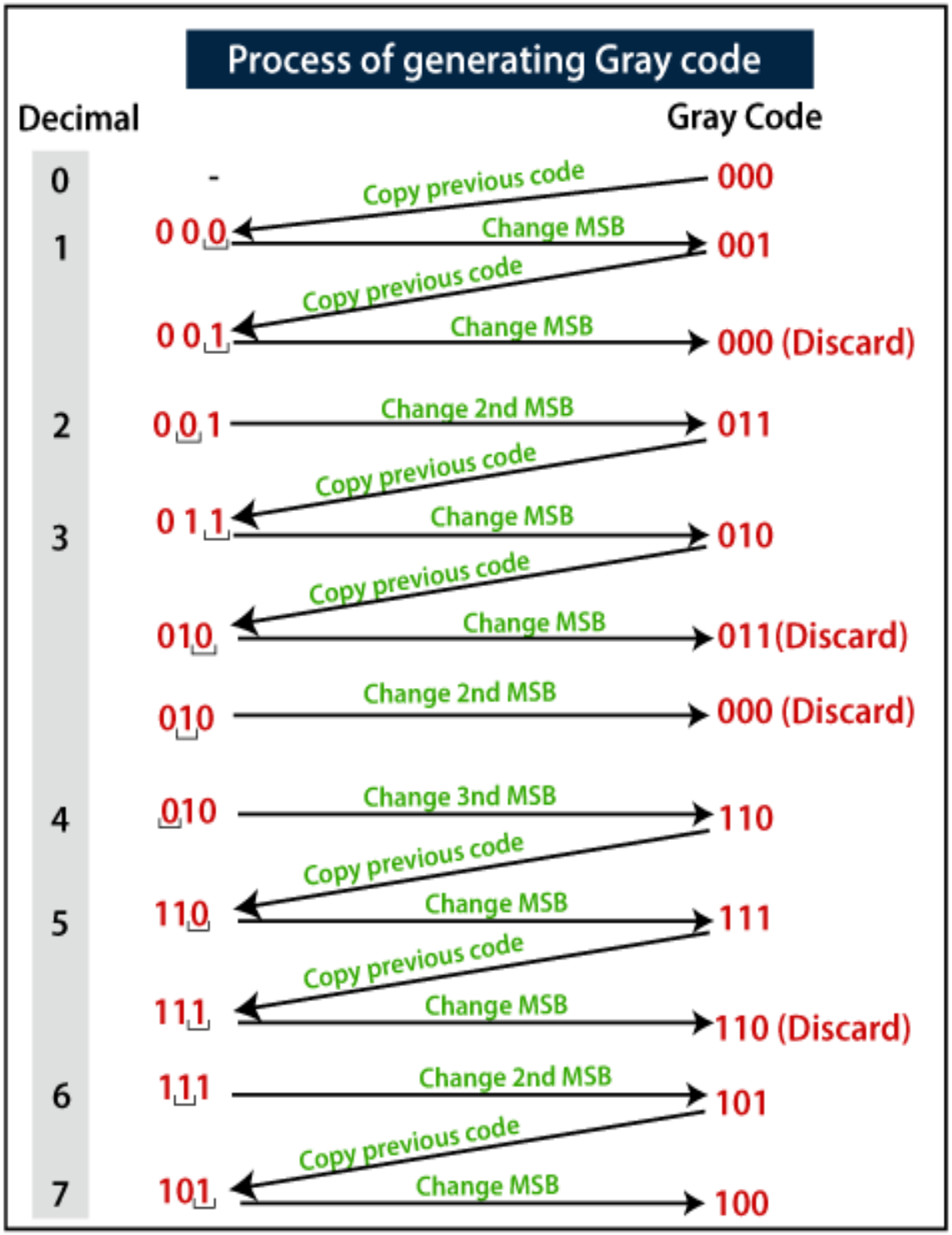
How to generate Gray code?

The prefix and reflect method are recursively used to generate the Gray code of a number. For generating gray code:

1. We find the number of bits required to represent a number.
2. Next, we find the code for 0, i.e., 0000, which is the same as binary.
3. Now, we take the previous code, i.e., 0000, and change the most significant bit of it.
4. We perform this process reclusively until all the codes are not uniquely identified.
5. If by changing the most significant bit, we find the same code obtained previously, then the second most significant bit will be changed, and so on.



Process of generating Gray Code



Gray Code Table

Decimal Number	Binary Number	Gray Code
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
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Digital Logic:

The Basic Gates

The basic gates are AND, OR & NOT gates.

AND gate

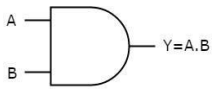
An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '.'.

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 inputsare '1'.

OR gate

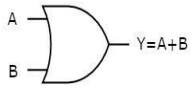
An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol '+'.

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

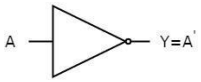
A NOT gate is a digital circuit that has single input and single output. The output ofNOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter.

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', thenthe 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 oneoutput, 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

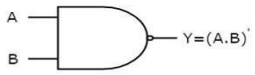
NAND gate is a digital circuit that has two or more inputs and produces an output,which is the **inversion of logical AND** of all those inputs.

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.

The following image shows the **symbol** of NAND gate, which is having two inputs A, B andone output, Y.



NAND gate operation is same as that of AND gate followed by an inverter. That’s why the NAND gate symbol is represented like that.

NOR gate

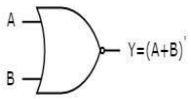
NOR gate is a digital circuit that has two or more inputs and produces an output,which is the **inversion of logical OR** of all those inputs.

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.

COMBINATIONAL LOGIC CIRCUIT

Boolean Laws and Theorems

Boolean Algebra is an algebra, which deals with binary numbers & binary variables. Hence, it is also called as Binary Algebra or logical Algebra. A mathematician, named George Boole had developed this algebra in 1854. The variables used in this algebra are alsocalled as Boolean variables.

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

Postulates and Basic Laws of Boolean Algebra

In this section, let us discuss about the Boolean postulates and basic laws that are used inBoolean algebra. These are useful in minimizing Boolean functions.

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, bysubstituting the Boolean variable with '0' or '1'.

Note− The complement of complement of any Boolean variable is equal to the variableitself. i.e., $(x')'=x$.

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**. The logical OR & logical AND operations of two Boolean variables x & y are shown below

$$x + y = y + x$$

$$x.y = y.x$$

The symbol '+' indicates logical OR operation. Similarly, the symbol '.' indicates logical AND operation and it is optional to represent. Commutative law obeys for logical OR & logical AND operations.

Associative Law

If a logical operation of any two Boolean variables is performed first and then thesame 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 ofthree Boolean variables x, y & z are shown below.

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

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

Associative law obeys for logical OR & logical AND operations.

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, bysubstituting 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.

Let us make the Boolean equations (relations) that we discussed in the section of Booleanpostulates and basic laws into two groups. The following table shows these two groups.

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 verifyall 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.

Simplification of Boolean Functions

Till now, we discussed the postulates, basic laws and theorems of Boolean algebra. Now, let us simplify some Boolean functions.

Eg

Let us **simplify** the Boolean function, f = p'qr + pq'r + pqr' + pqrWe can

simplify this function in two methods.

Method 1

Given Boolean function, f = p'qr + pq'r + pqr' +pqr.

Step 1 − In first and second terms r is common and in third and fourth terms pq is common.So, take the common terms by using **Distributive law**.

$$\Rightarrow f = (p'q + pq')\, r + pq(r' + r)$$

Step 2 − The terms present in first parenthesis can be simplified to Ex-OR operation. Theterms present in second parenthesis can be simplified to '1' using **Boolean postulate**

$$\Rightarrow f = (p \oplus q)\, r + pq(1)$$

Step 3 − The first term can't be simplified further. But, the second term can be simplified topq using **Boolean postulate**.

$$\Rightarrow f = (p \oplus q)r + pq$$

Therefore, the simplified Boolean function is

$$\mathbf{f = (p \oplus q)r + pq}$$

Method 2

Given Boolean function, f = p'qr + pq'r + pqr' + pqr.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

$$\Rightarrow f = p'qr + pq'r + pqr' + pqr + pqr + pqr$$

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.

$$\Rightarrow f = qr(p' + p) + pr(q' + q) + pq(r' + r)$$

Step 3 − Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in eachparenthesis.

$$\Rightarrow f = qr(1) + pr(1) + pq(1)$$

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying the above three terms.

$$\Rightarrow f = qr + pr + pq$$

$$\Rightarrow f = pq + qr + pr$$ Therefore, the

simplified Boolean function is **f = pq + qr + pr**.

So, we got two different Boolean functions after simplifying the given Boolean function in each method. Functionally, those two Boolean functions are same. So, based on the requirement, we can choose one of those two Boolean functions.

Eg

Let us find the **complement** of the Boolean function, f = p'q + pq'.

The complement of Boolean function is f' = (p'q + pq')'.

Step 1 − Use DeMorgan's theorem, (x + y)' = x'.y'.

$$\Rightarrow f' = (p'q)'.(pq')'$$

Step 2 − Use DeMorgan's theorem, (x.y)' = x' + y'

$$\Rightarrow f' = \{(p')' + q'\}.\{p' + (q')'\}$$

Step3 − Use the Boolean postulate, (x')'=x.

$$\Rightarrow f' = \{p + q'\}.\{p' + q\}$$

$$\Rightarrow f' = pp' + pq + p'q' + qq'$$

Step 4 − Use the Boolean postulate, xx'=0.

$$\Rightarrow f = 0 + pq + p'q' + 0$$

$$\Rightarrow f = pq + p'q'$$

Therefore, the **complement** of Boolean function, p'q + pq' is **pq + p'q'**.

Sum of product Method(SOP)

A canonical sum of products is a boolean expression that entirely consists of minterms. The Boolean function F is defined on two variables X and Y. The X and Y are the inputs of the boolean function F whose output is true when any one of the inputs is set to true.The truth table for Boolean Expression F is as follows:

Inputs		**Output**
X	**Y**	**F**
0	0	0
0	1	1
1	0	1
1	1	1

we can form the minterm from the variable's value. Now, a column will be added forthe minterm in the above table. The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs		**Output**	**Minterm**
X	**Y**	**F**	**M**
0	0	0	X'Y'
0	1	1	X'Y
1	0	1	XY'
1	1	1	XY

Now, we will add all the minterms for which the output is true to find the desired canonical SOP (Sum of Product) expression.

```
F=X' Y+XY'+XY
```

Converting Sum of Products (SOP) to shorthand notation

The process of converting SOP form to shorthand notation is the same as the process of finding shorthand notation for minterms. There are the following steps to find the shorthandnotation of the given SOP expression.

- Write the given SOP expression.
- Find the shorthand notation of all the minterms.
- Replace the minterms with their shorthand notations in the given expression.

Example: F = X'Y+XY'+XY

1. Firstly, we write the SOP expression:

F = X'Y+XY'+XY

2. Now, we find the shorthand notations of the minterms X'Y, XY', and XY.

$X'Y = (01)_2 = m_1$

$XY' = (10)_2 = m_2$

$XY = (11)_2 = m_3$

3. In the end, we replace all the minterms with their shorthand notations:

F=m1+m2+m3

Converting shorthand notation to SOP expression

The process of converting shorthand notation to SOP is the reverse process of converting SOP expression to shorthand notation. Let's see an example to understand this conversion.

Example:

Let us assume that we have a boolean function F, which defined on two variables X and Y.The minterms for the function F are expressed as shorthand notation is as follows:

$F=\sum(1,2,3)$

Now, from this expression, we will find the SOP expression. The Boolean function F has twoinput variables X and y and the output of F=1 for m1, m2, and m3, i.e., 1st, 2nd, and 3rd combinations. So,

$F=\sum(1,2,3)$

F= m1 + m2 + m3

F= 01 + 10 + 11

Now, we replace zeros with either X' or Y' and ones with either X or Y. Simply, the complement variable is used when the variable value is 1 otherwise the non-complement variable is used.

$F = \sum(1,2,3)$

F=01+10+11

F= A'B + AB' + AB

Karnaugh Map(K-Map) method

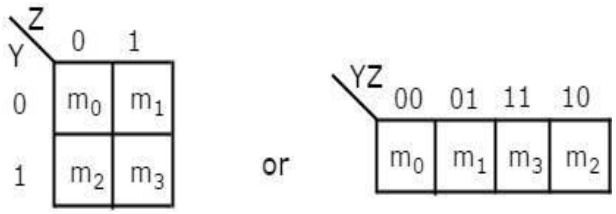
Karnaugh introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 5 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

2 Variable K-Map

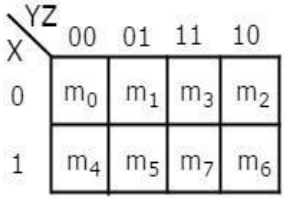
The number of cells in 2 variable K-map is four, since the number of variables is two.
The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3),(m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map

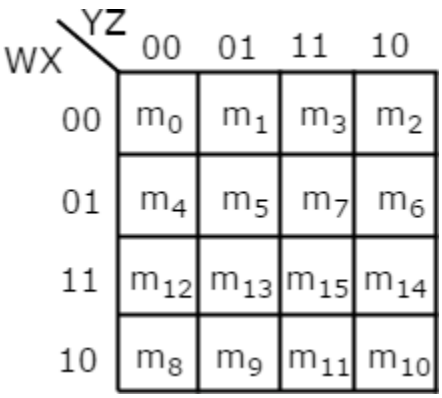
The number of cells in 3 variable K-map is eight, since the number of variables isthree. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3,m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables isfour. The following figure shows **4 variable K-Map**.

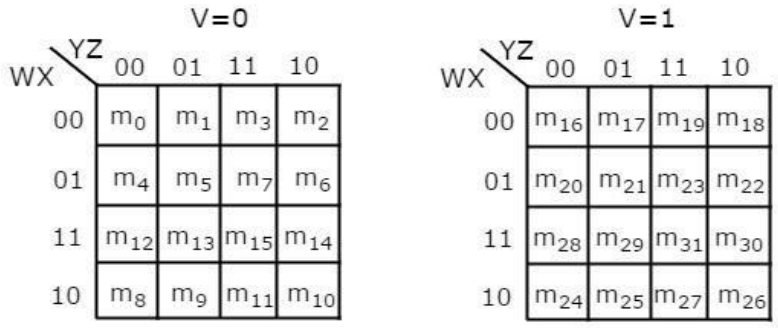


WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are $\{(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)\}$.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variablesis 5. The following figure shows **5 variable K-Map**.



V=0

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

V=1

WX \ YZ	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

- There is only one possibility of grouping 32 adjacent min terms.
- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of minterms from m_0 to m_{15} and m_{16} to m_{31}.
- If v=0, then 5 variable K-map becomes 4 variable K-map.

In the above all K-maps, we used exclusively the min terms notation. Similarly, you can useexclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is in **standard sum of products** form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

Follow these **rules for simplifying K-maps** in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.
- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single '1' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note 1 − If outputs are not defined for some combination of inputs, then those output values will be represented with **don't care symbol 'x'**. That means, we can consider them as either '0' or '1'.

Note 2 − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1'.

Eg

Let us **simplify** the following Boolean function, **f(W, X, Y, Z)= WX'Y' + WY +W'YZ'** using K-map.

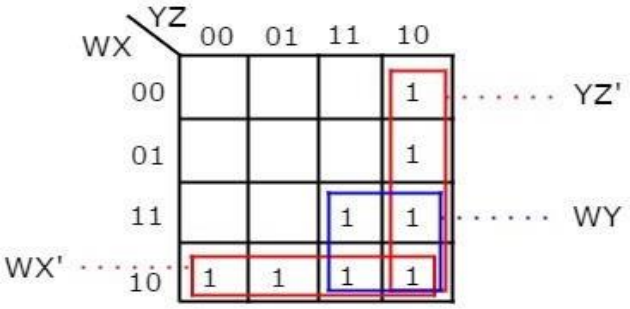
The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require **4 variable K-map**. The **4 variable K-map** with ones corresponding to thegiven product terms is shown in the following figure.

WX \ YZ	00	01	11	10
00				1
01				1
11			1	1
10	1	1	1	1

Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 arecorresponding to the product term, **WX'Y'**.
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4are corresponding to the product term, **WY**.
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 arecorresponding to the product term, **W'YZ'**.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no singleone left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The **4 variable K-map** with these three **groupings** is shown in the following figure.



Here, we got three prime implicants WX', WY & YZ'. All these prime implicants are **essential** because of following reasons.

- Two ones **(m_8 & m_9)** of fourth row grouping are not covered by any other groupings.Only fourth row grouping covers those two ones.
- Single one **(m_{15})** of square shape grouping is not covered by any other groupings.Only the square shape grouping covers that one.
- Two ones **(m_2 & m_6)** of fourth column grouping are not covered by any othergroupings. Only fourth column grouping covers those two ones.

Therefore, the **simplified Boolean function** is

$$f = WX' + WY + YZ'$$

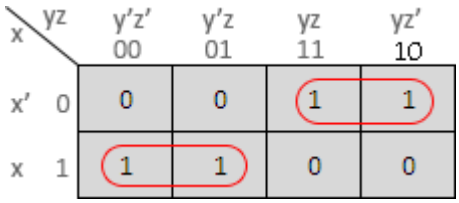
Follow these **rules for simplifying K-maps** in order to get standard product of sums form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given asproduct of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. Itshould be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map andleast power is zero.
- Each grouping will give either a literal or one sum term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single '0' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

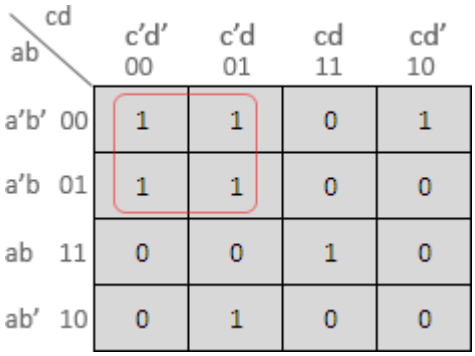
Note − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

Pair, Quad and Octet

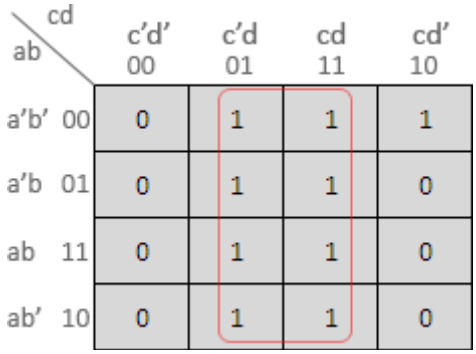
Pair Reduction Rule : Remove the variable which changes its state from complemented to uncomplemented or vice versa.Pair removes one variable only.



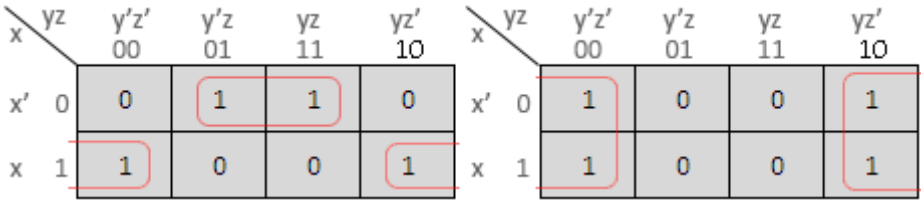
Quad Reduction Rule : Remove the two variables which change their states.A quadremoves two variables.



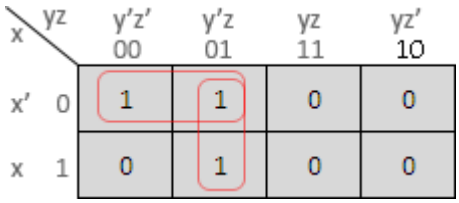
Octet Reduction Rule : Remove the three variables which changes their state.Octet removesthree variables.



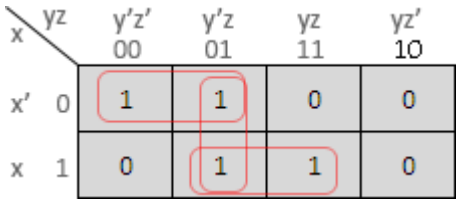
Map Rolling : Map rolling means roll the map considering the map as if its left edges are touching the right edges and top edges are touching bottom edges.While marking the pairsquads and octet, map must be rolled.



Overlapping Groups : Overlapping means same 1 can be encircled more than once.Overlapping always leads to simpler expressions.



Redundant Group : It is a group whose all 1's are overlapped by other groups. Redundantgroups must be removed. Removal of redundant group leads to much simpler expression.

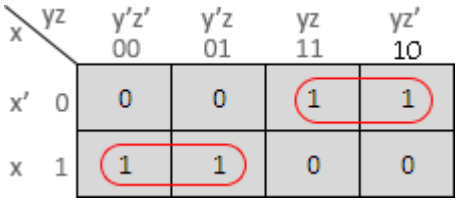


Eg : Represent the following boolean expression in a K-map and simplify.

F = x'yz + x'yz' + xy'z' + xy'z

Solution :

The K-map is as follows :



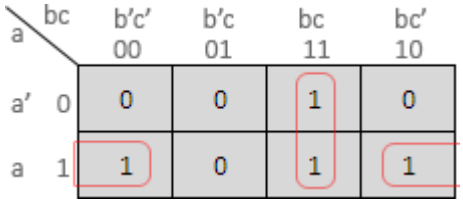
Hence the simplified expression isF = x'y + xy'

Ex. 2 : Simplify the following boolean expression using K-map.

F = a'bc + ab'c' + abc + abc'

Solution :

The K-map is as follows :



Hence the simplified expression is F = bc + ac'

Don’t Care Condition

The “Don’t Care” conditions allow us to replace the empty cell of a K-Map to form a grouping of the variables. While forming groups of cells, we can consider a “Don’t Care” cellas either 1 or 0 or we can simply ignore that cell. Therefore, “Don’t Care” condition can help usto form a larger group of cells.

A Don’t Care cell can be represented by a cross(X) in K-Maps representing a invalid combination. For example, in Excess-3 code system, the states 0000, 0001, 0010, 1101, 1110and 1111 are invalid or unspecified. These are called don’t cares. Also, in design of 4-bit BCD-to-XS-3 code converter, the input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are don’t cares.

A standard SOP function having don’t cares can be converted into a POS expression bykeeping don’t cares as they are, and writing the missing minterms of the SOP form as the maxterm of POS form. Similarly, a POS function having don’t cares can be converted to SOP form keeping the don’t cares as they are and write the missing maxterms of the POS expressionas the minterms of SOP expression.

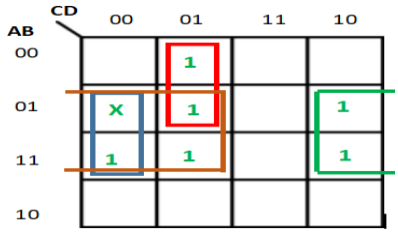
Eg

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

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

Explanation

The SOP K-map for the given expression is:



Therefore, SOP minimal is,

f = BC' + BD' + A'C'D

Eg

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

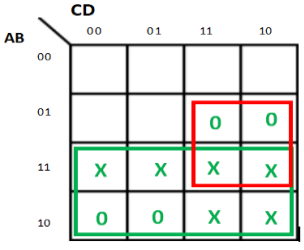
F(A, B, C, D) = m(0, 1, 2, 3, 4, 5) + d(10, 11, 12, 13, 14, 15)

Explanation:

Writing the given expression in POS form:

F(A, B, C, D) = M(6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)

The POS K-map for the given expression is:



Therefore, POS minimal is,

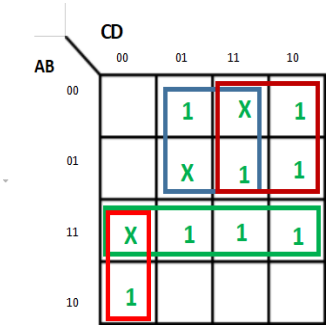
F = A'(B' + C')

Eg

Minimise the following function in SOP minimal form using K-Maps: F(A, B, C, D) = m(1, 2,6, 7, 8, 13, 14, 15) + d(3, 5, 12)

Explanation:

The SOP K-map for the given expression is:



Therefore,

$$f = AC'D' + A'D + A'C + AB$$

Product of Sum Method(POS)

A canonical product of sum is a boolean expression that entirely consists of maxterms. The Boolean function F is defined on two variables X and Y. The X and Y are the inputs of the boolean function F whose output is true when only one of the inputs is set totrue. The truth table for Boolean expression F is as follows:

Inputs		**Output**
X	**Y**	**F**
0	0	0
0	1	1
1	0	1
1	1	0

In our minterm and maxterm section, we learned about how we can form the maxterm from the variable's value. A column will be added for the maxterm in the above table. The complement of the variables is taken whose value is 0, and the variables whose value is 1 willremain the same.

Inputs		**Output**	**Minterm**
X	**Y**	**F**	**M**
0	0	0	X'+Y'
0	1	1	X'+Y
1	0	1	X+Y'
1	1	1	X+Y

Now, we will multiply all the minterms for which the output is false to find the desiredcanonical POS(Product of sum) expression.

$$F=(X'+Y').(X+Y)$$

Converting Product of Sum (POS) to shorthand notation

The process of converting POS form to shorthand notation is the same as the process of finding shorthand notation for maxterms. There are the following steps used to find the shorthand notation of the given POS expression.

- Write the given POS expression.
- Find the shorthand notation of all the maxterms.
- Replace the minterms with their shorthand notations in the given expression.

Eg

F = (X'+Y').(X+Y)

1. Firstly, we will write the POS expression:

$$F = (X'+Y').(X+Y)$$

2. Now, we will find the shorthand notations of the maxterms X'+Y' and X+Y.

$$X'+Y' = (00)_2 = M_0$$

$$X+Y = (11)_2 = M_3$$

3. In the end, we will replace all the minterms with their shorthand notations:

$$F=M_0.M_3$$

Converting shorthand notation to POS expression

The process of converting shorthand notation to POS is the reverse process of converting POS expression to shorthand notation. Let's see an example to understand thisconversion.

Eg

Let us assume that we have a boolean function F, defined on two variables X and Y.The maxterms for the function F are expressed as shorthand notation is as follows:

$$F=\prod(1,2,3)$$

Now, from this expression, we find the POS expression. The Boolean function F has twoinput variables X and Y and the output of F=0 for M1, M2, and M3, i.e., 1st, 2nd, and 3rd combinations. So,

$$F=\prod(1,2,3)$$

$$F= M1.M2.M3$$

$$F= 01.10.11$$

Next, we replace zeros with either X or Y and ones with either X' or Y'. Simply, if the valueof the variable is 1, then we take the complement of that variable, and if the value of the variable is 0, then we take the variable "as is".

$$F = \sum(1,2,3)$$

$$F=01.10.11$$

$$F=(A+B').(A'+B).(A'+B')$$

Product of Sum Simplification

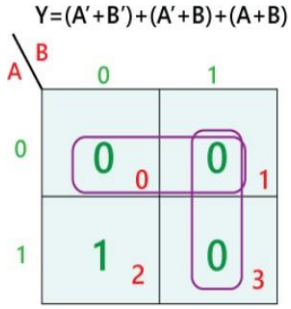
To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

1. We will populate the K-map by entering the value of 0 to each sum-term into the K-map cell and fill the remaining cells with one's.
2. We will make the groups of 'zeros' not for 'ones'.
3. Now, we will define the boolean expressions for each group as sum-terms.
4. At last, to find the simplified boolean expression in the POS form, we will combinethe sum-terms of all individual groups.

Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

Eg

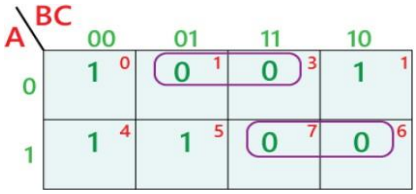
Y=(A'+B')+(A'+B)+(A+B)



Simplified expression: A'B

Eg

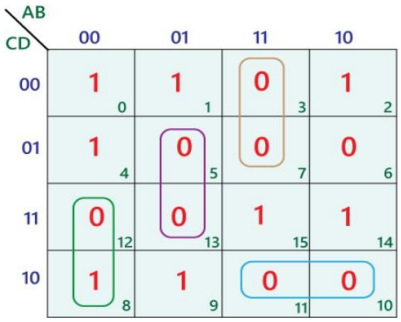
Y=(A + B + C') + (A + B' + C') + (A' + B' + C) + (A' + B' + C')



Simplified expression: Y=(A + C') .(A' + B')

Eg

F(A,B,C,D)=π(3,5,7,8,10,11,12,13)



Simplified expression: Y=(A + C') .(A' + B')

Combinational Circuits

Karnaugh map:

For each symbol of the Excess-3 code, we use 1's to draw the mapfor simplifying Boolean function :

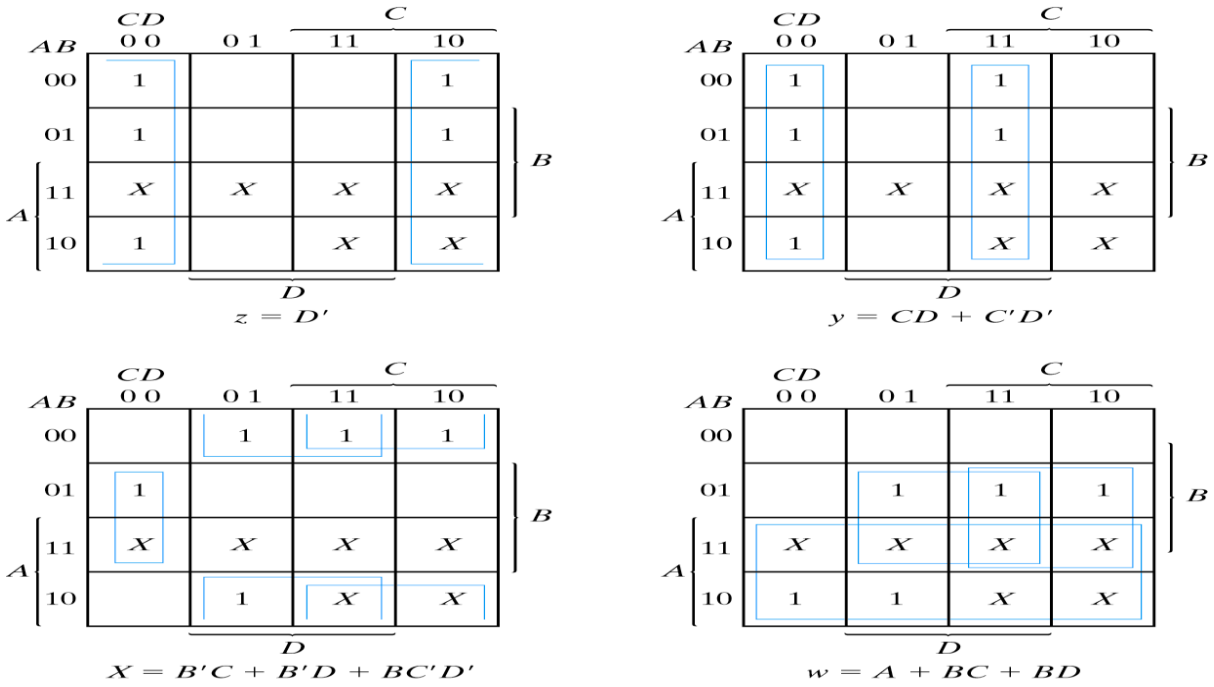


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Circuit implementation:

z = D'; y = CD + C'D' = CD + (C + D)'

x = B'C + B'D + BC'D' = B'(C + D) + B(C + D)' w = A + BC + BD = A + B(C + D)

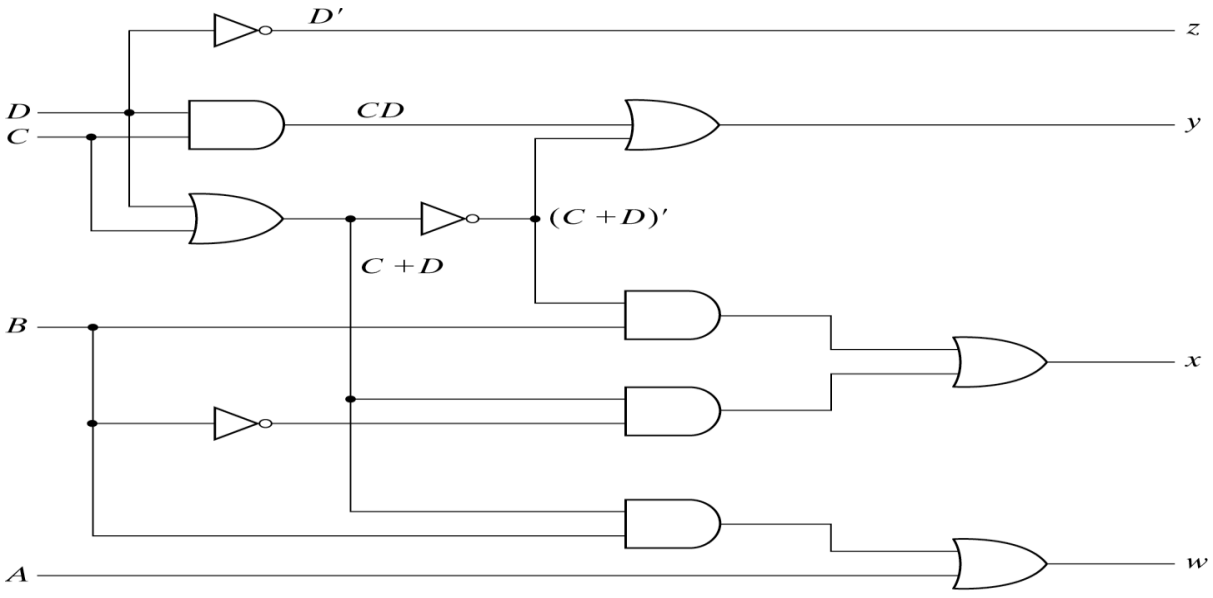
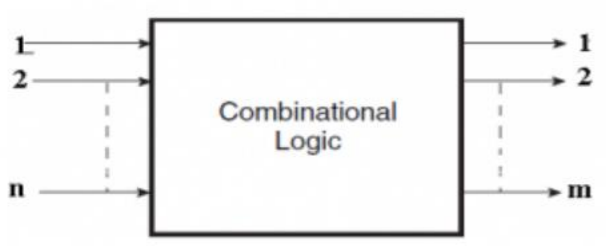


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

Introduction to combinational circuits:

❖ *A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.*

❖ A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.



Sequential circuits:

- ❖ *Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.*
- ❖ Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

Analysis Procedure

Explain the analysis procedure. Analyze the combinational circuit the following logic diagram. (May 2015)

- The analysis of a combinational circuit requires that we determine the function that the circuit implements.
- The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.
- The first step in the analysis is to make that the given circuit is combinational or sequential.
- Once the logic diagram is verified to be combinational, one can proceed to obtain the output Boolean functions or the truth table.

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

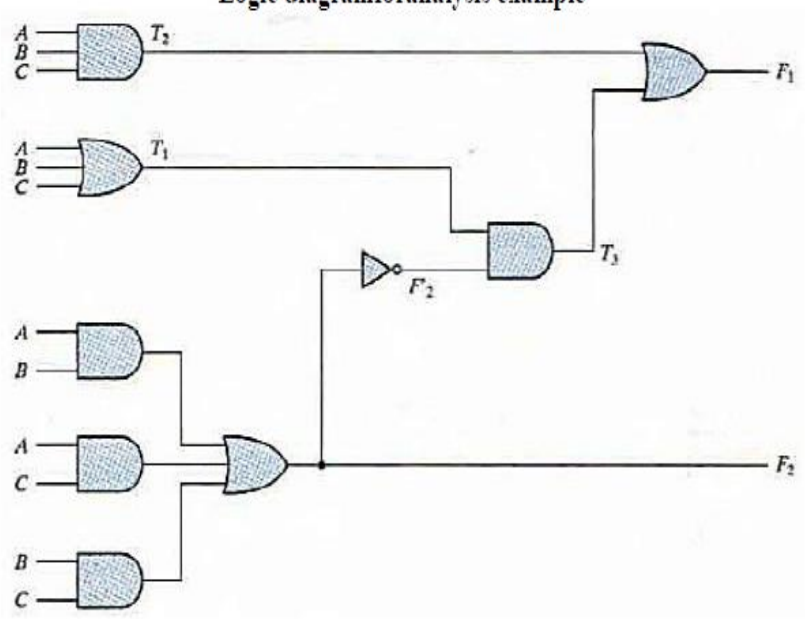
   Example:

   F2 = AB + AC + BC; T1 = A + B + C; T2 = ABC; T3 = F2'T1;

   F1 = T3 + T2

   F1 = T3 + T2 = F2'T1 + ABC = A'BC' + A'B'C + AB'C' + ABC

Logic diagramforanalysis example



The Boolean functions for the above outputs are,

Derive truth table from logic diagram :

- **We can derive the truth table in Table 4-1 by using the circuit of Fig.4-2.**

Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design procedure :

- ❖ The design of combinational circuits starts from the specification of the design objective and culminatesin a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.
- ❖ The procedure involved involves the following steps,
- ✓ From the specifications of the circuit, determine the required number of inputs and outputs and assign asymbol to each.
- ✓ Derive the truth table that defines the required relationship between inputs and outputs.
- ✓ Obtain the simplified Boolean functions for each output as a function of the input variables.
- ✓ Draw the logic diagram and verify the correctness of the design.

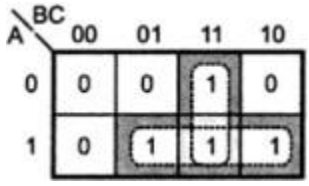
Example : Design a combinational logic circuit with three inputs , the output is at logic 1 when more than one inputs are at logic 1.

Solution: Assume A, B, C are inputs and Y is output .

Truth table

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

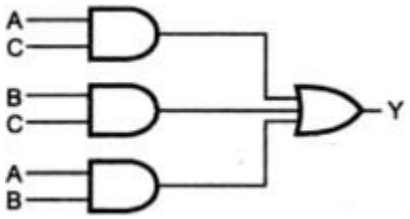
K map Simplification



Boolean Expression

Y=AC + BC + AB

Logic Diagram



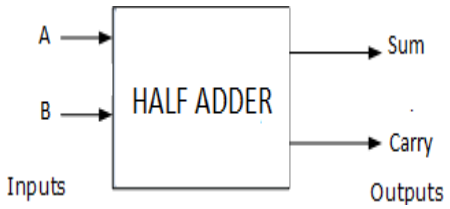
Binary Adder-Subtractor:

- **HALF-ADDER**: a combinational circuit that performs theaddition of two bits is called a half adder.

- Half-adder is used to add two bits. Therefore, half-adder has two inputs and two outputs, with SUM and CARRY. Figure shows the truth table of a half-adder. The Boolean expressions for SUM and CARRY are,

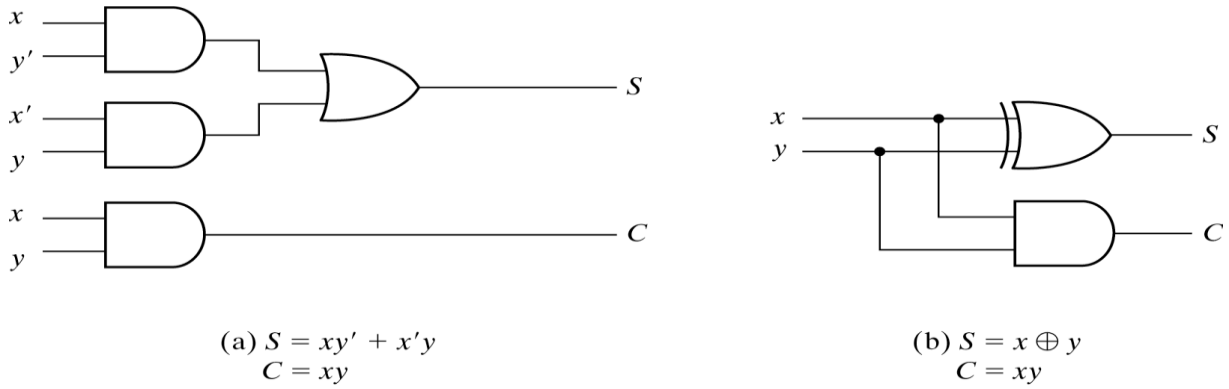
SUM = AB'+A'B
CARRY = AB

- These expressions shows`that, SUM output is EX-OR gate and the CARRY output is ANDgate. Figure shows the be implementation of half-adder with all the combinations including the implementation using NAND gates only.



Truth table

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

Fig. 4-5 Implementation of Half-Adder

- **FULL ADDER :** one that performs the addition of three bits(twosignificant bits and a previous carry) is a full adder.

$S = z \oplus (x \oplus y)$

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

$= xy'z' + x'yz' + xyz + x'y'z$

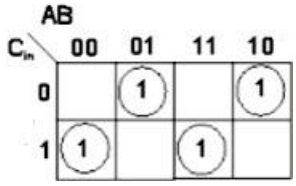
$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$

Truth table

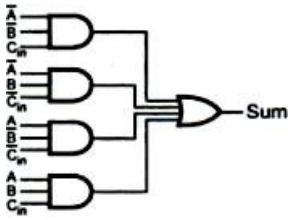
Inputs			Outputs	
A	**B**	**Cin**	**Cout**	**Sum**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-map simplifications

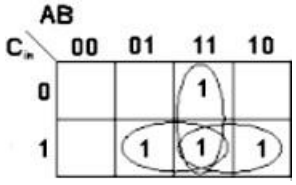
For Sum



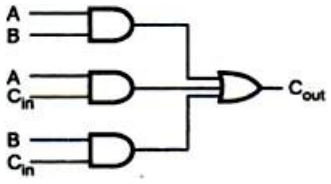
$Sum = \bar{A}\bar{B}Cin + \bar{A}B\bar{C}in + A\bar{B}\bar{C}in + ABCin$



For Cout



$Cout = AB + BCin + ACin$



Carry Propagation :

- The signal from C_i to the output carry C_{i+1}, propagates through an AND and ORgates, so, for an n-bit RCA, there are 2n gate levels for the carry to propagate from input to output.
- Because the propagation delay will affect the output signals on different time, sothe signals are given enough time to get the precise and stable outputs.

The Full Adder can be implement using Two Half Adders and OR gates

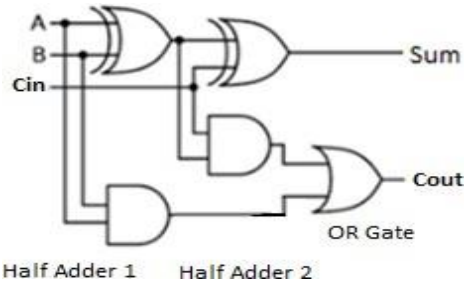
The expression for sum is

$$\begin{aligned}
\text{Sum} &= \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in} \\
&= C_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}\,(\overline{A}\,B + A\,\overline{B}) \\
&= C_{in}\,(A \cdot B) + \overline{C}_{in}\,(A \oplus B) \\
&= C_{in}\,(\overline{A \oplus B}) + \overline{C}_{in}\,(A \oplus B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}$$

The Expression for carry is

$$\begin{aligned}
C_{out} &= AB + A\,C_{in} + BC_{in} \\
&= AB + A\,C_{in} + BC_{in}\,(A + \overline{A}) \\
&= ABC_{in} + AB + A\,C_{in} + \overline{A}\,BC_{in} \\
&= AB\,(C_{in} + 1) + A\,C_{in} + \overline{A}\,BC_{in} \\
&= AB + AC_{in} + \overline{A}\,BC_{in} \\
&= AB + A\,C_{in}\,(B + \overline{B}) + \overline{A}\,B\,C_{in} \\
&= AB\,C_{in} + AB + A\,\overline{B}\,C_{in} + \overline{A}\,BC_{in} \\
&= AB\,(C_{in} + 1) + A\,\overline{B}\,C_{in} + \overline{A}\,BC_{in} \\
&= AB + A\,\overline{B}\,C_{in} + \overline{A}\,BC_{in} \\
&= AB + C_{in}\,(A\,\overline{B} + \overline{A}\,B) \\
&= AB + C_{in}\,(A \oplus B)
\end{aligned}$$

Logic Diagram



Subtractor

Subtractor is the logic circuit which is used to subtract two binary number (digit) and provides Difference and Borrow as a output. In digital electronics we have two types of subtractor, Half Subtractor and Full Subtractor.

Rules for two bit addition

0 - 0 = 0
0 - 1 = 1 with borrow 1
1 - 0 = 1
1 - 1 = 0

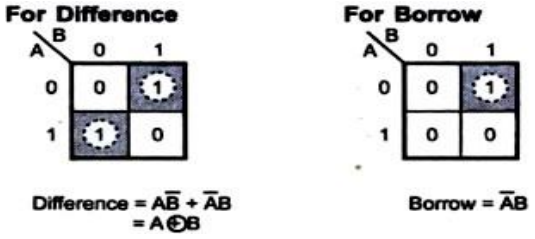
Half Subtractor

Half Subtractor is used for subtracting one single bit binary digit from another single bit binary digit.The truth table of Half Subtractor is shown below.

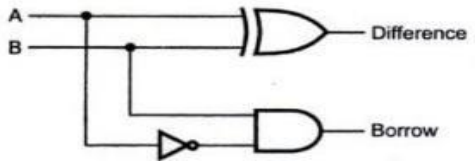
Truth table of Half adder

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map for Difference and Borrow



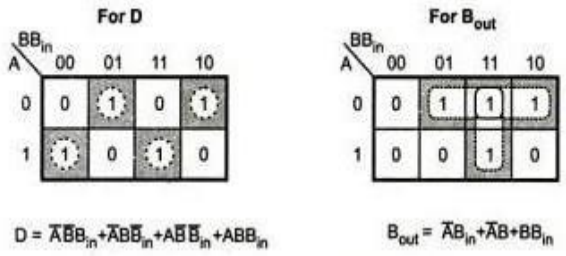
Logic Diagram



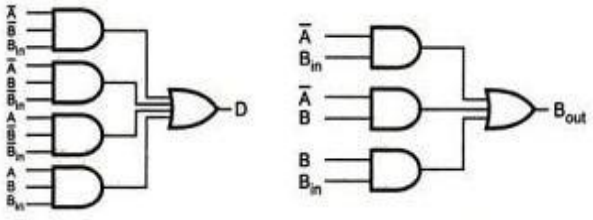
Full Subtractor

A logic Circuit Which is used for Subtracting Three Single bit Binary digit is known as Full Subtractor.The inputs are A,B, Bin and the outputs are D and Bout.

K-map for D and Bout



Logic Diagram



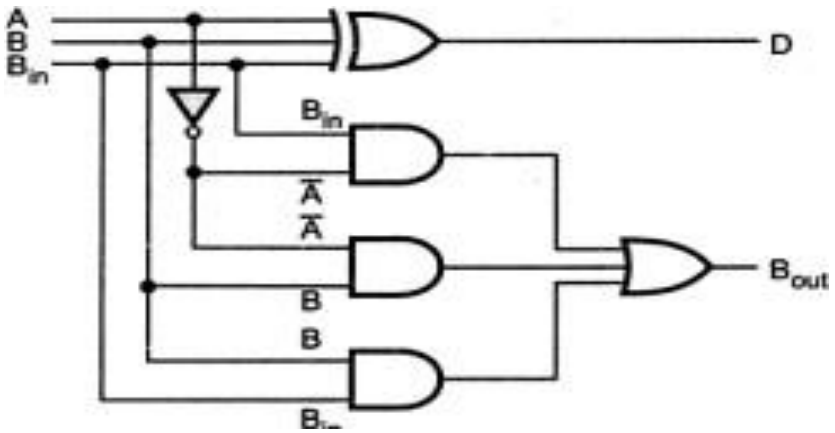
Truth table

Inputs			Outputs	
A	B	Bin	D	Bout
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

We can further simplify the function of the Difference (D)

$$
\begin{aligned}
D &= \bar{A}\,\bar{B}\,B_{in} + \bar{A}\,B\,\bar{B}_{in} + A\,\bar{B}\,\bar{B}_{in} + A\,B\,B_{in} \\
&= B_{in}\,(\bar{A}\,\bar{B} + AB) + \bar{B}_{in}\,(\bar{A}\,B + A\,\bar{B}) \\
&= B_{in}\,(A \odot B) + \bar{B}_{in}\,(A \oplus B) \\
&= B_{in}\,(\overline{A \oplus B}) + \bar{B}_{in}\,(A \oplus B) \\
&= B_{in} \oplus (A \oplus B)
\end{aligned}
$$

Simplified Logic diagram



Magnitude comparator:

- The equality relation of each pair of bits can be expressed logically with anexclusive-NOR function as:

$A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$

$x_i = A_iB_i + A_i'B_i'$ for i = 0, 1, 2, 3

$(A = B) = x_3x_2x_1x_0$

Magnitude Comparator Definition

- A magnitude comparator is a combinational circuit that compares two numbers A & B to determine whether:
- A > B, or A = B, or A < B

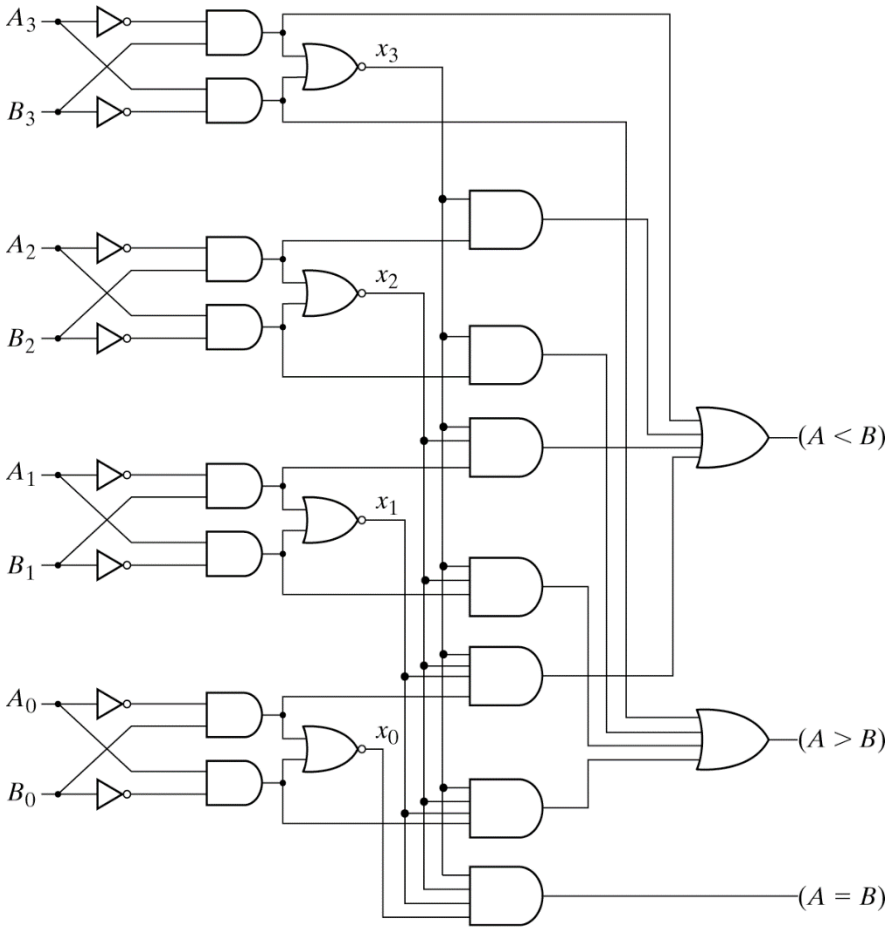


Fig. 4-17 4-Bit Magnitude Comparator

- We inspect the relative magnitudes of pairs of MSB. If equal, we compare thenext lower significant pair of digits until a pair of unequal digits is reached.
- If the corresponding digit of A is 1 and that of B is 0, we conclude that A>B.
- (A>B)=

$$A_3B'_3+x_3A_2B'_2+x_3x_2A_1$$

$$B'_1+x_3x_2x_1A_0B'_0 \text{ (A<B)=}$$

$$A'_3B_3+x_3A'_2B_2+x_3x_2A'_1B_1+x_3x$$

$$_2x_1A'_0B_0$$

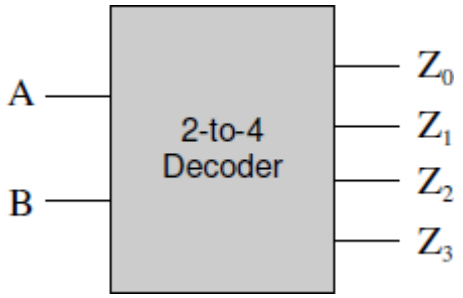
Decoder

- Decoder is a combinational circuit. It has N inputs and 2^N outputs.
- The decoder is called n-to-m-line decoder, where $m \leq 2^n$.
- the decoder is also used in conjunction with other code converters such as aBCD-to-seven_segment decoder.
- 3-to-8 line decoder: For each possible input combination, there are sevenoutputs that are equal to 0 and only one that is equal to 1.

2 to 4 Decoder

It has 2 inputs and $2^2 = 4$ outputs.

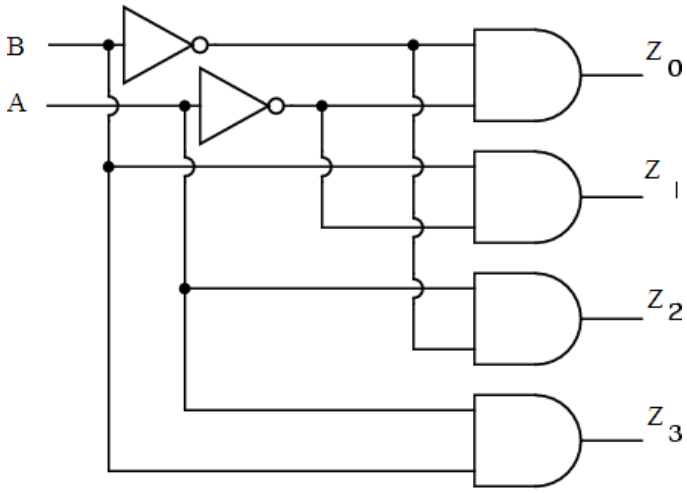
Circuit Diagram



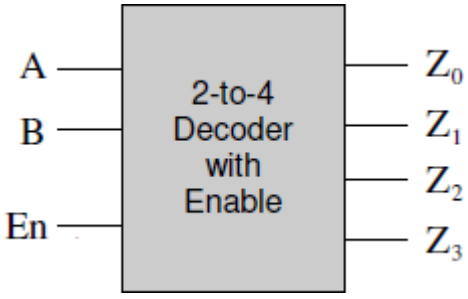
Truth Table

A	B	Z_0	Z_1	Z_2	Z_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Logic Diagram



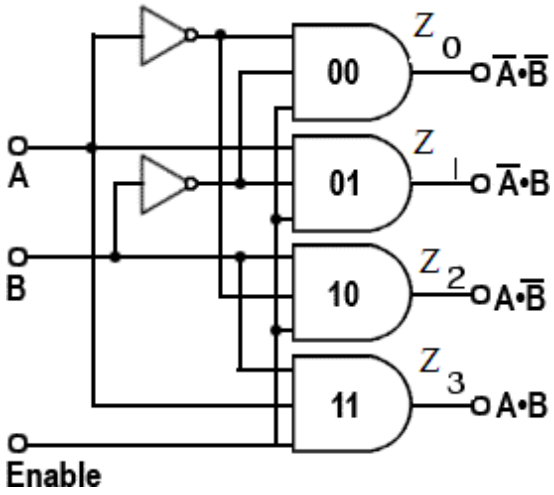
2 to 4 Decoder with Enable input



Truth Table

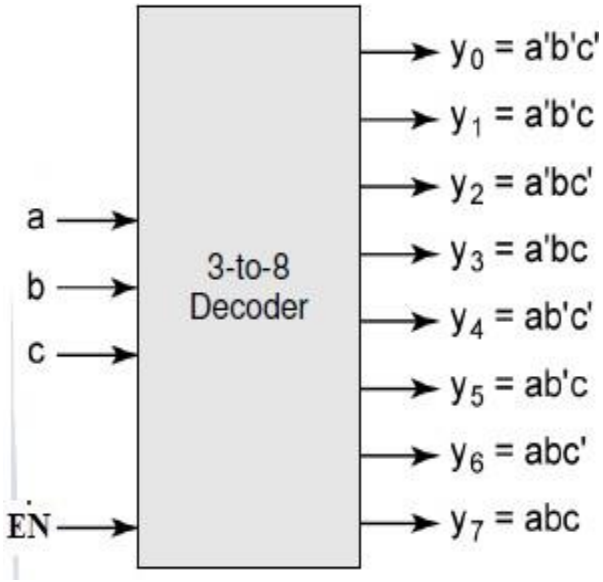
	En	A	B	Z_0	Z_1	Z_2	Z_3
enabled	1	0	0	1	0	0	0
	1	0	1	0	1	0	0
	1	1	0	0	0	1	0
	1	1	1	0	0	0	1
disabled	0	x	x	0	0	0	0

Logic Diagram



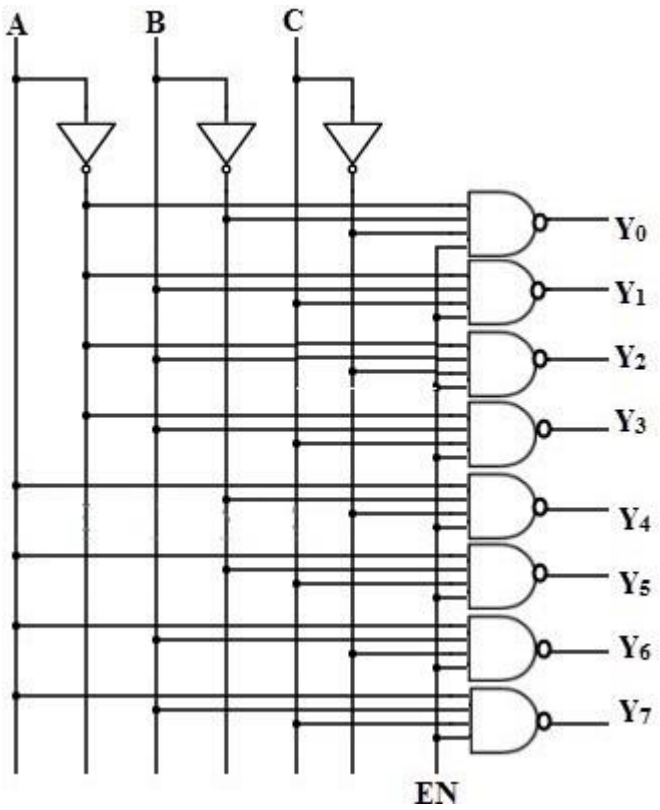
3 to 8 Decoder

It has 3 inputs and $2^3 = 8$ outputs.



Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Logic Diagram

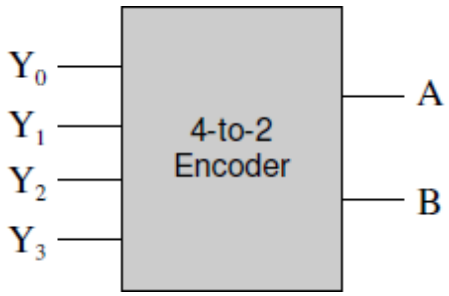


Encoders

Encoders is a combinational circuit which takes 2^N inputs and gives out N outputs, the enable pin should be kept 1 for enabling the circuit.

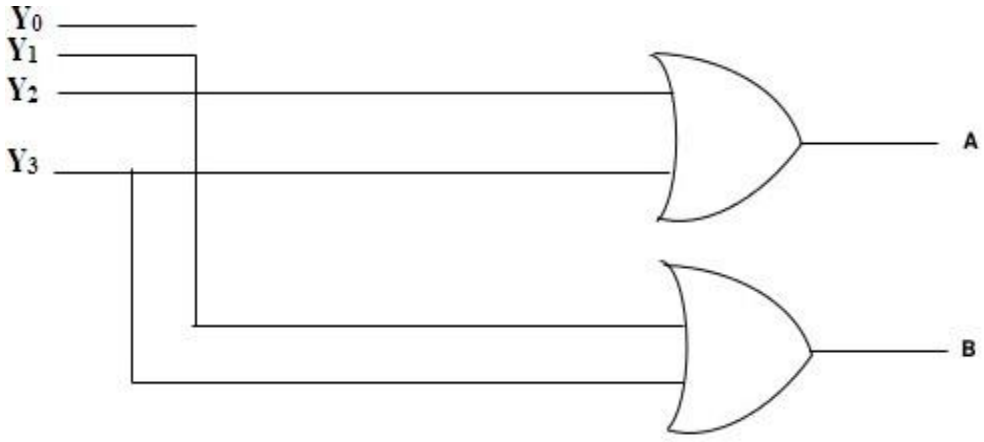
4 to 2 Encoder

It has 2^2 inputs and 2 outputs.



Truth Table

Y_0	Y_1	Y_2	Y_3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



Priority Encoders

A Priority Encoder works opposite of the decoder circuit. If more than one input is active, the higher order input has priority.

4 to 2 Priority Encoders

D0-D3 - inputs A1,A0 – outputs

Active (A)– Valid indicator. It indicates the output is valid or not Output is invalid

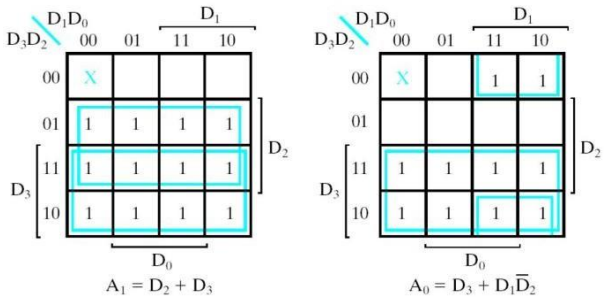
when no inputs are active .i.e, A=0

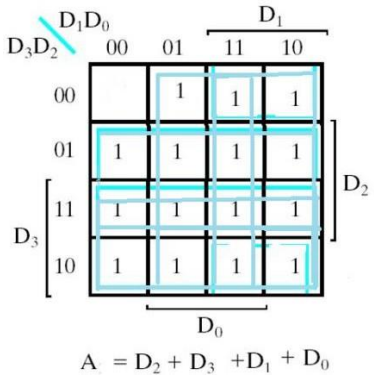
Output is valid when at least one input is active .i,e, A=1

Truth Table

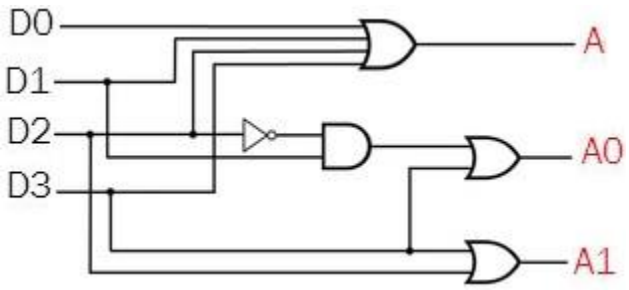
D3	D2	D1	D0	A1	A0	Active
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

K-map simplification

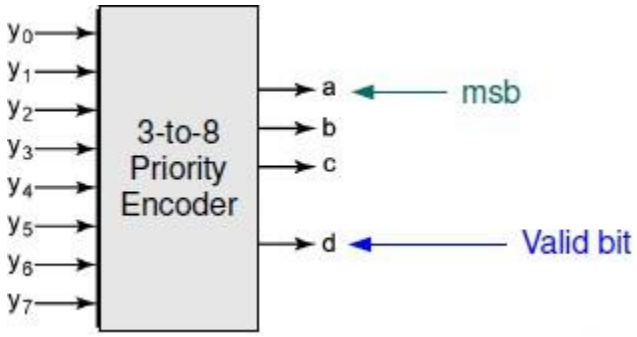




Logic Diagram



3 to 8 Priority Encoder



y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

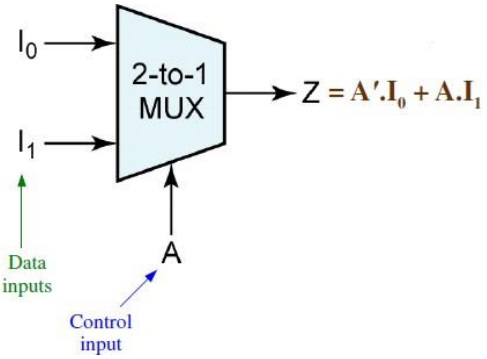
Multiplexer (Mux)

- Multiplexer is a combinational circuit that selects binary information from one of many inputs and directs it into single output.
- The selection of particular input is controlled by a set of selection line Mutliplexer has 2^n inputs, n select line (control input) and one output
- It also called as Data selector.

2 to 1 Multiplexer

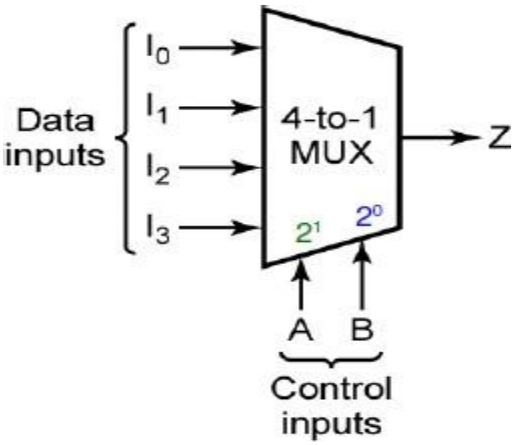
has 2^1 inputs, 1 select line and one output

Circuit diagram



4 to 1 MUX

4 to 1 MUX has $2^2 = 4$ inputs, 2 select line and one output

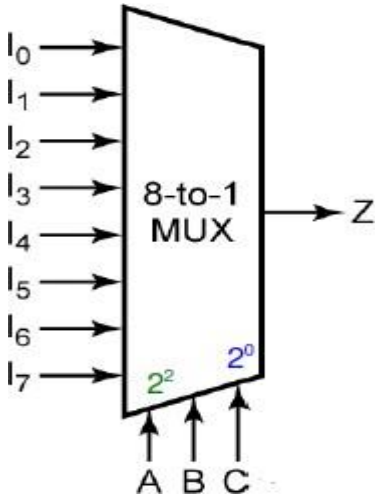


$Z = A'.B'.I_0 + A'.B.I_1 + A.B'.I_2 + A.B.I_3$

A	B	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

8 to1 MUX

8 to1 MUX has $2^3 = 8$ inputs, 3 select line and one output



A	B	C	Z
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$$Z = A'.B'.C'.I_0 + A'.B'.C.I_1 + A'.B.C'.I_2 + A'.B.C.I_3 + A.B'.C'.I_0 + A.B'.C.I_1 + A'.B.C'.I_2 + A.B.C.I_3$$

$$Z = \Sigma\, m_i.I_i$$

MUX as universal combinational modules

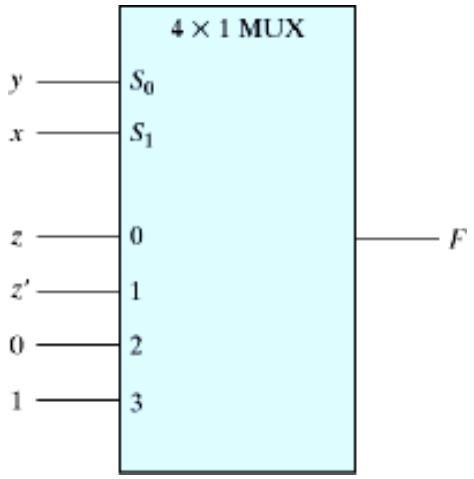
- Each minterm of the function can be mapped to a data input of the multiplexer.
- For each row in the truth table, where the output is 1, set the corresponding data input of the mux to 1.
- Set the remaining inputs of the mux to 0.

Example 1: Implement the following Boolean function using 4:1 MUX F(x,y,z) = Σm(1, 2, 6, 7)

Truth Table

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

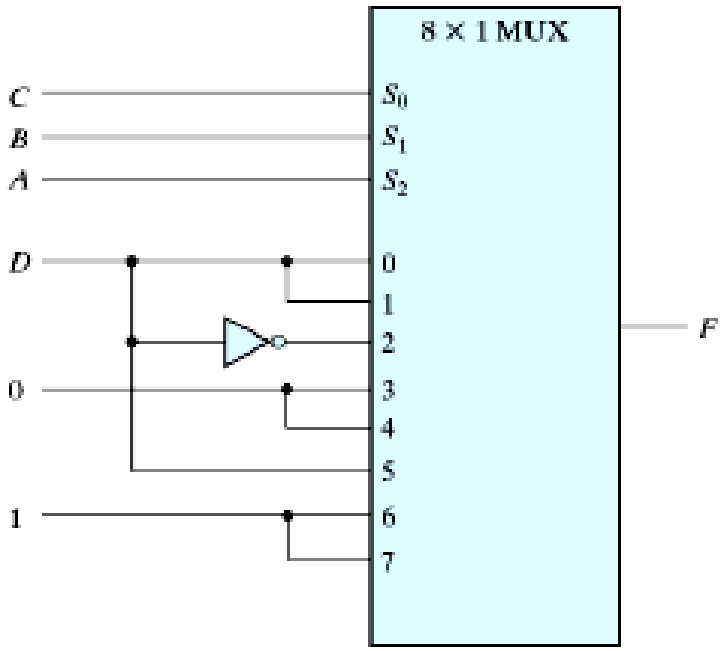
Multiplexer Implementation



Example 2: Implement the following Boolean function using 8:1 MUX

F(A,B,C,D) = Σm(1, 3, 4, 11,12-15)

A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = D'
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



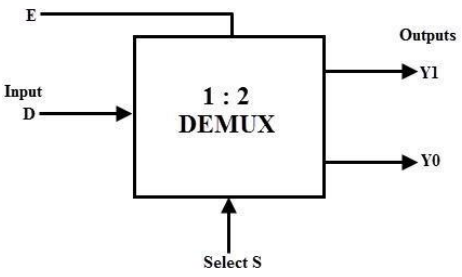
Demultiplexer (DEMUX)

Demultiplexer has 2^n outputs , n select lines, one input. A

demultiplexer is also called a data distributor.

1-to-2 demultiplexer

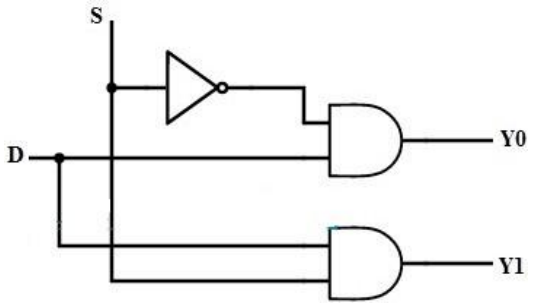
has 2^2 outputs , 2 select lines, one input.



Truth Table

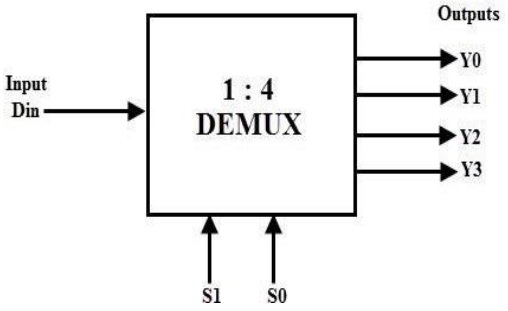
Select	Input	Outputs	
S	D	Y_1	Y_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Logic diagram



1-to-4 Demultiplexer

It has one input,2 select lines,4 outputs



Truth Table

Data Input	Select Inputs		Outputs			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

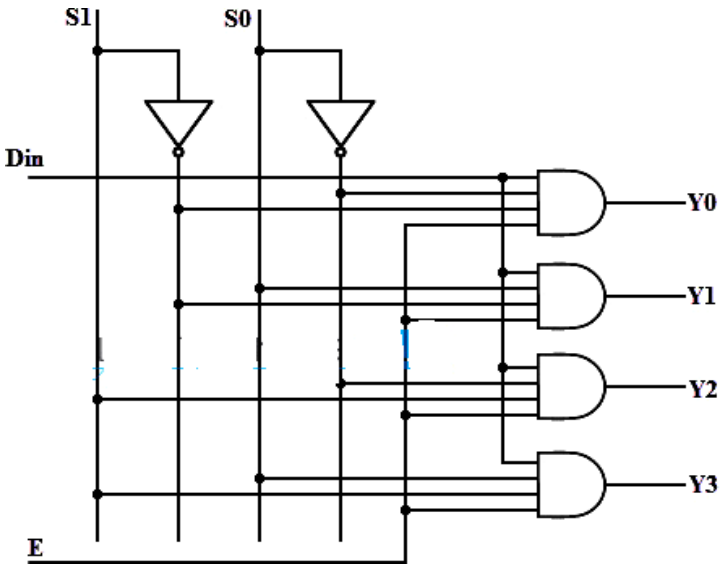
$Y0 = \overline{S1}\ \overline{S0}\ D$

$Y1 = \overline{S1}\ S0\ D$

$Y2 = S1\ \overline{S0}\ D$

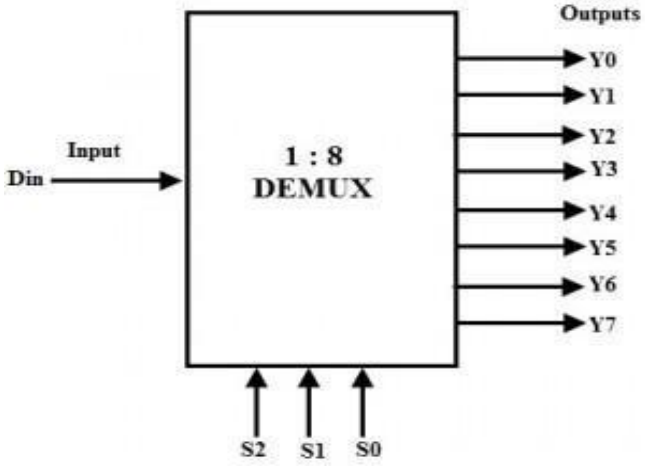
$Y3 = S1\ S0\ D$

Logic Diagram



1-to-8 Demultiplexer

Has one input 3-select lines 8-outputs



Truth Table

Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

$$Y0 = D\,\overline{S2}\,\overline{S1}\,\overline{S0}$$

$$Y1 = D\,\overline{S2}\,\overline{S1}\,S0$$

$$Y2 = D\,\overline{S2}\,S1\,\overline{S0}$$

$$Y3 = D\,\overline{S2}\,S1\,S0$$

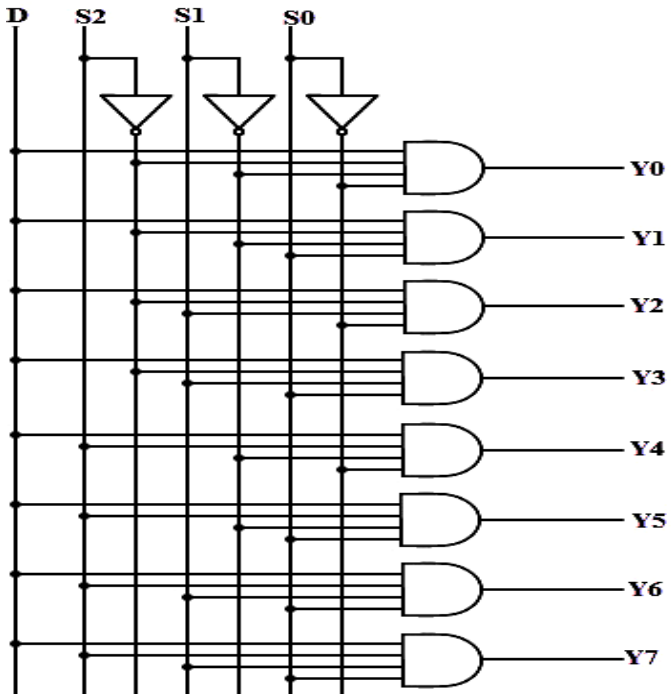
$$Y4 = D\,S2\,\overline{S1}\,\overline{S0}$$

$$Y5 = D\,S2\,\overline{S1}\,S0$$

$$Y6 = D\,S2\,S1\,\overline{S0}$$

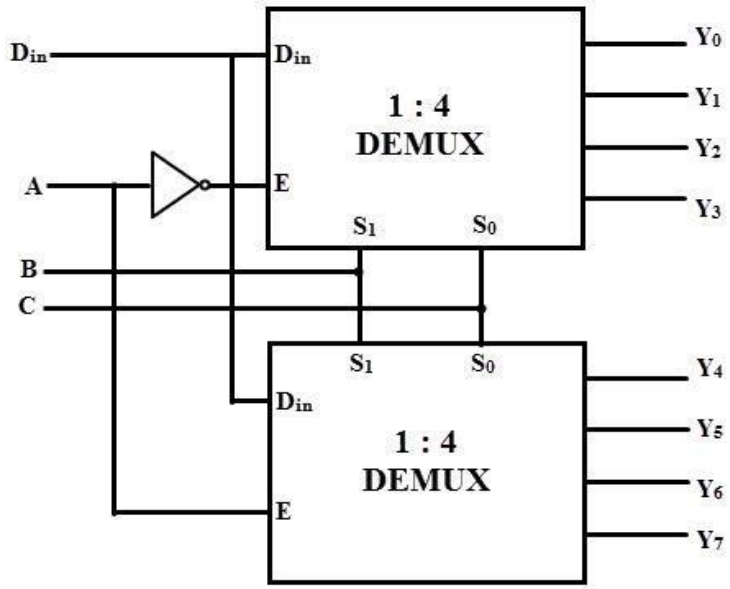
$$Y7 = D\,S2\,S1\,S0$$

Logic Diagram



1-to-8 DEMUX using Two 1-to- 4 Demultiplexers

1-to-8 demultiplexer can be implemented by using two 1-to-4 demultiplexers with a proper cascading.



In the figure, the highest significant bit A of the selection inputs are connected to the enable inputs such that it is complemented before connecting to one DEMUX and to the other it is directly connected. By this configuration, when A is set to zero, one of the output lines from Y0 to Y3 is selected based on the combination of select lines B and C. Similarly, when A is set to one, based on the select lines one of the output lines from Y4 to Y7 will be selected.

Applications of Demultiplexer

- Synchronous data transmission systems
- Boolean function implementation (as we discussed full subtractor function above)
- Data acquisition systems
- Combinational circuit design
- Automatic test equipment systems
- Security monitoring systems (for selecting a particular surveillance camera at a time), etc.