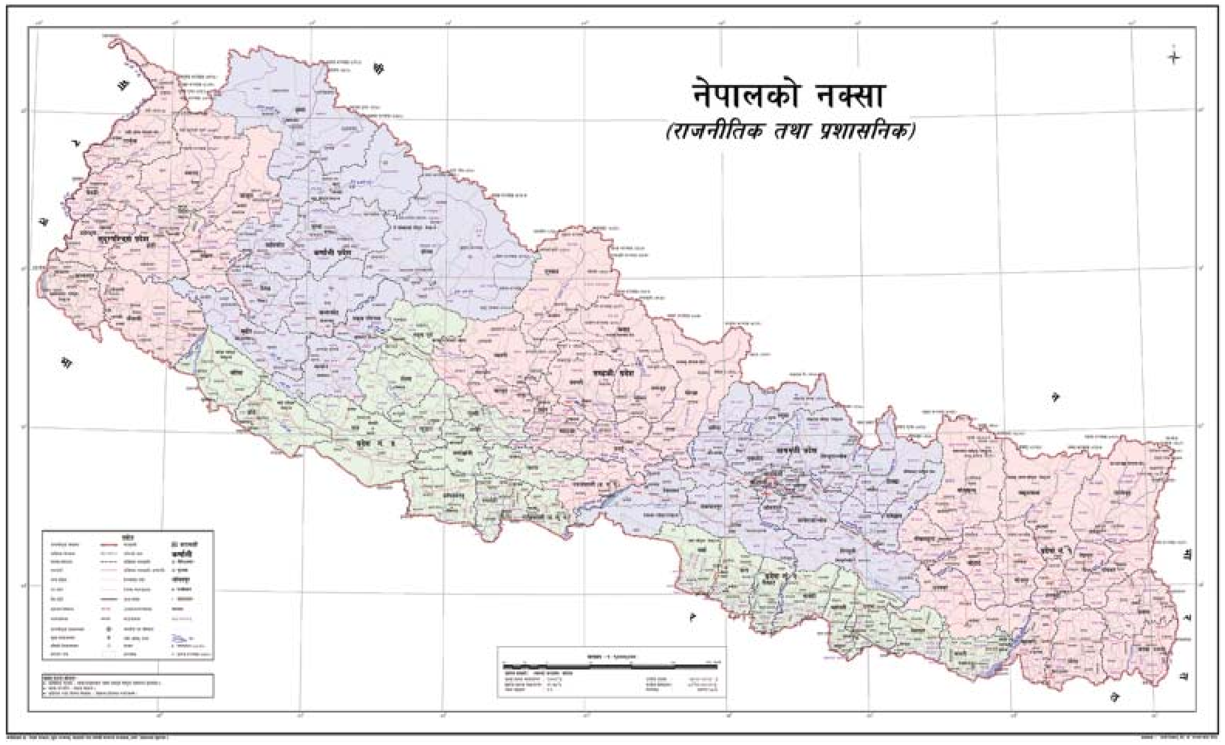
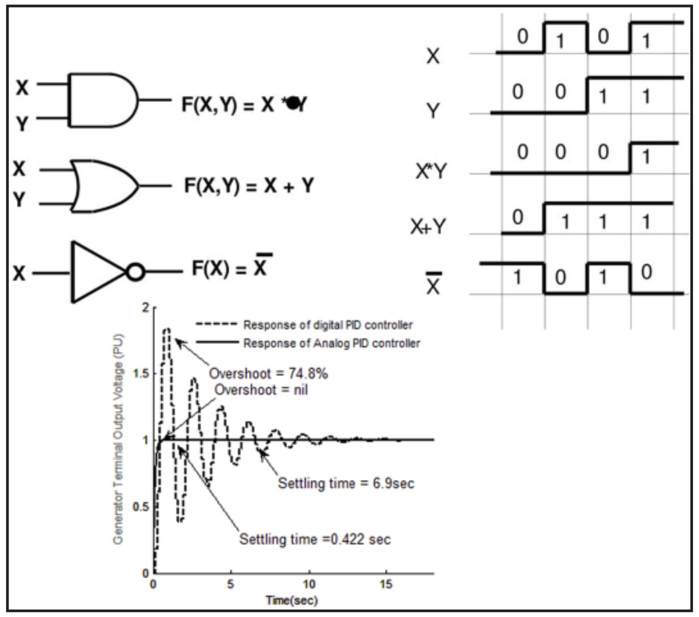
Computer Engineering

9

Fundamentals of Digital System





Government of Nepal
Ministry of Education, Science and Technology
Curriculum Development Centre
Sanothimi, Bhaktapur
Phone : 5639122/6634373/6635046/6630088
Website : www.moecdc.gov.np

Technical and Vocational Stream
Learning Resource Material

Fundamentals of Digital System
(Grade 9)

Secondary Level
Computer Engineering

Government of Nepal
Ministry of Education, Science and Technology
Curriculum Development Centre
Sanothimi, Bhaktapur

Publisher : Government of Nepal
Ministry of Education, Science and Technology
Curriculum Development Centre
Sanothimi, Bhaktapur

Layout by Khados Sunuwar

Preface

The curriculum and curricular materials have been developed and revised on a regular basis with the aim of making education objective-oriented, practical, relevant and job oriented. It is necessary to instill the feelings of nationalism, national integrity and democratic spirit in students and equip them with morality, discipline and self-reliance, creativity and thoughtfulness. It is essential to develop in them the linguistic and mathematical skills, knowledge of science, information and communication technology, environment, health and population and life skills. it is also necessary to bring in them the feeling of preserving and promoting arts and aesthetics, humanistic norms, values and ideals. It has become the need of the present time to make them aware of respect for ethnicity, gender, disabilities, languages, religions, cultures, regional diversity, human rights and social values so as to make them capable of playing the role of responsible citizens with applied technical and vocational knowledge and skills. This Learning Resource Material for Computer Engineering has been developed in line with the Secondary Level Computer Engineering Curriculum with an aim to facilitate the students in their study and learning on the subject by incorporating the recommendations and feedback obtained from various schools, workshops and seminars, interaction programs attended by teachers, students and parents.

In bringing out the learning resource material in this form, the contribution of the Director General of CDC Dr. Lekhnath Poudel, Pro, Dr. Subarna Shakya, Bibha Sthapit, Anil Barma, Bhuwan Panta, Arun Roka, Lina Maharjan, Trimandir Prajapati is highly acknowledged. The book is written by Sankar Kumar Yadav, Bishnuraj Bhandari, Satyaram Suwal and Bimal Thapa and the subject matter of the book was edited by Badrinath Timalsina and Khilanath Dhamala. CDC extends sincere thanks to all those who have contributed in developing this book in this form.

This book is a supplimentary learning resource material for students and teachrs. In addition they have to make use of other relevnt materials to ensure all the learning outcomes set in the curriculum. The teachers, students and all other stakeholders are expected to make constructive comments and suggestions to make it a more useful learning resource material.

2076 BS

Ministry of Education, Science and Technology
Curriculum Development Centre

Table of content

UNIT-1
Number System

Objectives:

At the end of this unit, you will able to

1. to obtain formalism of logic enabling you to analyse logical processes
2. implement simple logical operations using combinational logic circuits
3. understand common forms of number representation in digital electronic circuits and to convert between different representations
4. understand the logical operation of simple arithmetic and other MSI circuits (Medium Scale Integrated Circuits)
5. develop concepts of sequential circuits enabling you to analyse sequential systems in terms of state machines
6. implement synchronous state machines using flip-flops.

Introduction

Number Systems

In a positional number system, there are only a few symbols, called digits, and these symbols represent different values depending on the position they occupy in the number. Data are stored in a binary format, which cannot be easily read by human beings. This is the reason why input and output (I/O) interfaces are required. Every computer stores numbers, character letters, and other special characters in coded form.

The value of each digit in such number is determined by

7. The digit itself
8. The position of the digit in the number and
9. The base of the number system where base is defined as the total number of digits available in the number system.

The number system which we are using in our day-to-day life is called the decimal number system. The Number systems follow the same pattern, the values can be written with single character and then a new column is used to count the highest value in the counting system. The Numerical value is called the base of the system like

Binary has base 2, Octal has 8, Decimal has 10, and Hexadecimal has decimal plus alphabet up to F letters.

For example:

The binary system has 2 numerical characters and so has a base of 2:

0 1

Similarly, the Decimal system has 10 numerical characters and so has a base of 10:

0 1 2 3 4 5 6 7 8 9

For writing numbers greater than 9 a second column is added to the left, and this column has 10 times the value of the column immediately to its right.

Decimal Number System, (base 10)

Decimal has ten values 0 1 2 3 4 5 6 7 8 9. If we need larger values than 9, we have to add extra columns. The columns are derived taking first digit 0 and put the single digit 0 1 2 3 4 5 6 7 8 9 in back like 00 01 02 03 04 05 06 07 08 09 which is also equal to single digit 0 1 2 3 4 5 6 7 8 9and then turn of next column 1 and put the single digit 0 1 2 3 4 5 6 7 8 9 in back similar way 10 11 12 13 14 15 16 17 18 19 when completing of maximum value of that base value then start with the derive number as it is like 10 0, 10 1 … 10 9 is same of 100, 101……. 109 and so on. Each column value is ten times the value of the column to its right. You know that in the decimal system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, etc. However, you may not have given much attention to the fact that each position represents a specific power of the base (10).

For example:

The decimal number 4389 (written as 4389_{10}) consists of the digit 9 in the unit position, 8 in the tens position, 3 in the hundreds position, 4 in the thousands position and its value can be shown as:

$(4\times10^3) + (3\times10^2) + (8\times10^1) + (9\times10^0)$

$= (4 \times 1000) + (3 \times 100) + (8 \times 10) + (9 \times 1)$

$= 4000+300+80+9$

$= 4389$

It may also be observed that the same digit signifies different values, depending on the position it occupies in the number.

For example:

In 4389_{10} the digit 9 signifies $9 \times 10^0 = 9$
In 4389_{10} the digit 9 signifies $9 \times 10^1 = 90$
In 4389_{10} the digit 9 signifies $9 \times 10^2 = 900$
In 4389_{10} the digit 9 signifies $9 \times 10^3 = 9000$

Hence, any number can be represented by using the available digits and arranging them in various positions.

The principles which apply to the decimal number system also apply to any other positional number system. It is important only to keep track of the base of the number system, in which we are working.

1. The value of the base determines the total number of different symbols or digits available in the number system. The first of these choices is always zero. i.e. 0
2. The maximum value of a single digit is always equal to one less than the value of the base. E.g. Base 10 has single digit 9.

Binary Number System (base 2)

The binary number system is exactly like the decimal number system, except that it has base 2, instead of 10. Now we can deduce that Binary number system has only two values 0 and 1. If we need larger values than 1, add extra columns. The columns are derived taking first digit 0 and put the single digit 1

For example:
The Binary number 1101 (written as 1101_2) consists of the digit 1 in the unit position, 0 in the tens position, 1 in the hundreds position, 1 in the thousands position and its value can be shown as:

$(1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$
$= (1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$
$= 8+4+0+1$
$= 13$

For example the decimal value thirteen is written 1101 in binary.

Octal Number System (base 8)

Octal Number System has eight values 0 1 2 3 4 5 6 7. If we need larger values than 7, add extra columns. The columns are derived taking first digit 0 and put the single digit from (0 1 2 3 4 5 6 7) in back like 00 01 02 03 04 05 06 07 is also equal to single digit 0 1 2 3 4 5 6 7 and then turn of next column 1 and put the single digit 0 1 2 3 4 5 6 7 in back similar way 10 11 12 13 14 15 16 17 when completing of maximum value of that base value then start with the derive number as it is like 10 0, 10 1 ……….. 10 7 is same of 100, 101 ………. 107 and so on. Each column value is eight times the value of the column to its right.

For example:

The octal number 437 (written as 4378) consists of the digit 7 in the unit position, 3 in the tens position, 4 in the hundreds position and its value can be shown as:

(4×82) + (3×81) + (7×80)
$= (4 \times 64) + (3 \times 8) + (7 \times 1)$
= 256+24+7
= 287

It may also be observed that the same digit signifies different values, depending on the position it occupies in the number.
For example the decimal value twenty-seven is written 33 in octal (3 eights + 3 ones).

Hexadecimal Number System (base 16)

Hexadecimal Number System has sixteen character 0 1 2 3 4 5 6 7 8 9 A B C D E F. If you need larger values than F, add extra columns. The columns are derived taking first digit 0 and putting the single digit 0 1 2 3 4 5 6 7 8 9 A B C D E F in back like 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F is also equal to single digit 0 1 2 3 4 5 6 7 8 9 A B C D E F and then turn of next column 1 and put the single digit 0 1 2 3 4 5 6 7 8 9 A B C D E F in back similar way 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F when completing of maximum value of that base value then start with the derive number as it is like 10 0, 10 1 ……….. 10 F is same of 100, 101 ………. 10F and so on. Each column value is ten times the value of the column to its right.

For example:

The decimal number 438A (written as 438A 16) consists of the digit A in the unit position, 8 in the tens position, 3 in the hundreds position, 4 in the thousands position and its value can be shown as:

(4×163) + (3×162) + (8×161) + (A×160)
= (4 × 4096) + (3 × 256) + (8 × 16) + (10 × 1)
= 16384+768+128+10
=17290

It may also be observed that the same digit signifies different values, depending on the position it occupies in the number.

For example the decimal value sixty-eight is written as 44 in hexadecimal (4 sixteen + 4 ones).

Table 1.1				
Some column values of different number system				
Decimal	1000	100	10	1
Binary	8	4	2	1
Octal	512	64	8	1
Hexadecimal	4096	256	16	1

Each of these different number systems works in the same way, it is just that each system has a different base, and the column values in each system increase by multiples of the base number as columns are added to the left.

Because this module describes several different number systems, it is important to know which system is being described. Therefore, if there is some doubt which system a number is in, the base of the system, written as a subscript immediately after the value, is used to identify the number system.

For example:

1010 represents the decimal value ten. (1 ten + 0 units)
102 represent the binary value two. (1 two + 0 units)
108 represent the octal value eight. (1 eight + 0 units)
1016 represents the hexadecimal value sixteen. (1 sixteen + 0 units)

Table 1.2			
Decimal	Binary	Octal	Hexadecimal
Radix (Base) 10	Radix (Base) 2	Radix (Base) 8	Radix (Base) 16
0	0	0	0
1	1	1	1
2		2	2
3		3	3
4		4	4
5		5	5
6		6	6
7		7	7
8			8
9			9
			A
			B
			C
			D
			E
			F

Example: Convert 2222 in Hexadecimal number.

```
      |  2222
  ----|---------
   16 |  138        14
  ----|---------                =8AE
   16 |  8          10
  ----|---------
      |  0          8
```

The System Radix

The base of a system, more properly called the RADIX, is the number of different values that can be expressed using a single digit. Therefore the decimal system has a radix of 10, the binary system has a radix of 2, the octal system has a radix of 8, and hexadecimal has a radix of 16. The range of number values in different number systems is shown in Table 1.2; Notice that because the hexadecimal system must express 16 values using only one column, it uses the letters A B C D E & F to represent the numbers 10 to 15.

The Radix Point

When writing a number, the digits used give its value, but the number is 'scaled' by its radix point.

For example, 618.510 are ten times bigger than 61.8510although the digits are the same.

also notice that when using multiple number systems, the term 'Radix Point' is used instead of 'Decimal Point'. When using decimal numbers, a decimal point is used, but if a different system is used, it would be wrong to call the point a decimal point, it would need to be called "Binary point" or "Octal point" etc. The simplest way around this is to refer to the point in any system (which will of course have its value labelled with its radix) as the Radix Point.

Exponents

A decimal number such as 618.510 can be considered as the sum of the values of its individual digits, where each digit has a value dependent on its position within the number (the value of the column):

= 618.510

Table 1.3

Column 2	Column 1	Column 0	Column -1
6 hundreds	+1 tens	+8 units	+5 tenths
(6×102)	(1×101)	(8×100)	(5×10-1)
600	+10	8	+0.5

Each digit in the number is multiplied by the system radix raised to a power depending on its position relative to the radix point. This is called the Exponent. The immediate digit to the left of the radix point has the exponent 0 applied to its radix, and for each place to the left, the exponent increases by one. The first place to the right of the radix point has the exponent -1 and so on, positive exponents to the left of the radix point and negative exponents to the right.

This method of writing numbers is widely used in electronics with decimal numbers, but can be used with any number system. Only the radix is different.

Hexadecimal exponents 79.216 = (7 x 161) + (9 x 160) + (2 x 16-1)
Octal exponents 97.68 = (9 x 81) + (7 x 80) + (6 x 8-1)
Binary Exponents 18.12 = (1 x 21) + (8 x 20) + (1 x 2-1) = 10.5
Use the calculator for answer.

Floating Point Notation

If electronic calculator hasn't radix points other than in decimal, this could be a problem. The radix exponent can also be used to eliminate the radix point, without altering the value of the number.

For example

It is all done by changing the radix exponent.

902.610 = 902.6 x 100 = 90.26 x 101 = 9.026 x 102 = .9026 x 103

The radix point is moved one place to the left by increasing the exponent by one.

It is also possible to move the radix point to the right by decreasing the exponent. In this way the radix point can be positioned wherever it is required, in any number system, simply by changing the exponent. This is called floating point notation and then calculate by calculator or yourself.

Normalised Form

Putting the radix point at the front of the number, and keeping it there by changing the exponent we can make, calculations easier for any radix.

4 Bit Binary Representation

Table 1.4				
Decimal	MSB	4 Bit Binary		LSB
	23=8	22=4	21=2	20=1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

When a number is stored in an electronic system, it is stored in a memory location having a fixed number of binary bits. Some of these memory locations are used for general storage while others, having some special function, are called registers. Wherever a number is stored, it will be held in some form of binary, and must always have a set number of bits. Therefore a decimal number such as 13, which can be expressed in four binary bits as 11012 becomes 000011012 when stored in an eight-bit register. This is achieved by adding four Non Significant Zeros to the left of the most significant ‘1’ digit.

Using this system, a binary register that is n bits wide can hold 2n values.

Therefore an 8 bit register can hold 28 values = 256 values (0 to 255)

A 4 bit register can hold 24 values = 16 values (0 to 15)

Bottom of Form

Filling the register with non-significant zeros is fine - if the number is smaller than the maximum value the register will hold, but how about larger numbers? These must be dealt with by dividing the binary number into groups of 8 bits, each of which can be stored in a one-byte location, and using several locations to hold the different parts of the total value.

Converting Between Number Systems

It is often necessary to convert values written in one number system to another. The purpose of this module is to explain just that, and to get you to carry out some simple conversions so that you can not only convert between number systems, but also understand how the conversion process works. There are various ways to tackle conversions without a calculator; once the conversion methods are learned, the only skills needed are the ability to multiply and divide, and to add together a few numbers.

Conversion from any system to decimal

The number of values that can be expressed by a single digit in any number system is called the system radix and any value can be expressed in terms of its system radix.

Octal to Decimal

For example the system radix of octal is 8, since any of the 8 values from 0 to 7 can be written as a single digit.

Convert 758 to decimal.

Using the values of each column, (which in an octal integer are powers of 8) the octal value 208 can also be written as:

(2x81) + (0 x 80)

As (81 = 8) and (80 =1),

This gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:

2x8 =16   0x1 = 0

Then simply add these results to give the decimal value.

16 + 0 = 1610

Therefore, 208 = 1610

Convert 1268 to decimal.

Using the values of each column, (which in an octal integer are powers of 8) the octal value 1268 can also be written as:

(1x82) + (2x81) + (6 x 80)

As (82 = 64), (81 = 8) and (80 =1),

This gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:

1x64 = 64   2x8 =16   6x1 = 6

Then simply add these results to give the decimal value.

64 + 16 + 6 = 8610

Therefore, 1268 = 8610

Convert 758 to decimal.

Using the values of each column, (which in an octal integer are powers of 8) the octal value 758 can also be written as:

(7x81) + (5 x 80)

As (81 = 8) and (80 =1),

This gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:

7x8 =56   5x1 = 5

Then simply add these results to give the decimal value.

56 + 5 = 6110

Therefore, 758 = 6110

Convert 126_8 to decimal.

Using the values of each column, (which in an octal integer are powers of 8) the octal value 10278 can also be written as:

(1x83) + (0x82) + (2 x 81) + (7x80)
As (83 = 512), (82 = 64), (81 = 8) and (80 =1),
This gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:

1x512 = 512   0x64 =0   2x8 = 16   7x1=7
Then simply add these results to give the decimal value.
512+0+16+7 = 53510
Therefore, 10278 = 53510

Binary to Decimal

Convert 10012 to decimal.

The same method can be used to convert binary number to decimal:

= (1x23) + (0x22) + (0x21) + (1x20)
= 8 + 0 +0 +1
= 910
Therefore, 10012 = 910

Convert 111_2 to decimal.

The same method can be used to convert binary number to decimal:

= $(1x2^2) + (1x2^1) + (1x2^0)$
= 4+ 2 +1
= 7_{10}
Therefore, $111_2 = 7_{10}$

Hexadecimal to Decimal

Convert B2D16 to decimal.

Using the same method to convert hexadecimal to decimal

$= (Bx16^2) + (2x16^1) + (Dx16^0)$

$= (11x16^2) + (2x16^1) + (13x16^0)$

$= 2816 + 32 + 13$

$= 2861_{10}$

Therefore $B2D_{16} = 2861_{10}$.

Convert CD_{16} to decimal.

Using the same method to convert hexadecimal to decimal

$= (Cx16^1) + (Dx16^0)$

$= (12x16^1) + (13x16^0)$

$= 196 + 13$

$= 209_{10}$

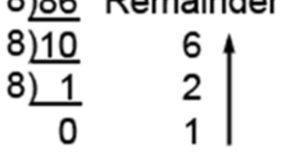
Therefore CD16 = 20910.

The same method (multiplying each digit by its column value) can be applied to convert any system to decimal.

Now the convert Decimal to any radix form.

To convert a decimal integer number (a decimal number in which any fractional part is ignored) to any other radix, all that is needed is to continually divide the number by its radix, and with each division, write down the remainder. When read from bottom to top, the remainder will be the converted result.

Decimal to Octal



Example: Decimal to Octal Conversion

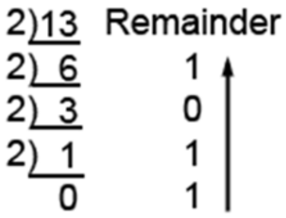
For example, to convert the decimal number 8610 to octal; divide 8610 by the system radix, which when converting to octal is 8. This gives the answer 10, with a remainder of 6.

Continue dividing the answer by 8 and writing down the remainder until the answer = 0

Now simply write out the remainders, starting from the bottom, to give 1268.

Therefore 8610 = 1268

Decimal to Binary



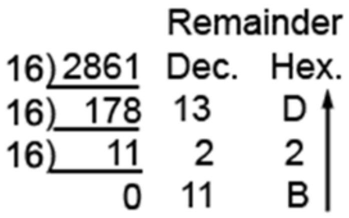
Example: Decimal to Binary Conversion

This process also works to convert decimal to binary, but this time the system radix is 2:

For example, to convert the decimal number 13_{10} to binary:

Therefore $13_{10} = 1101_2$

Decimal to Hexadecimal

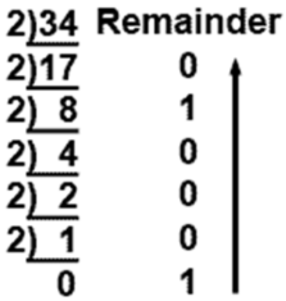


Example : Decimal to Hexadecimal Conversion

It also works to convert decimal to hexadecimal, but now the radix is 16:
When remainders may be greater than 9 replace with the alphabetical value, it easier to use decimal for the remainders, and then convert them to hexadecimal.
Therefore 2861_{10} = $B2D_{16}$

Numbers with Fractions

It is very common in the decimal system to use fractions; that is any decimal number that contains a decimal point, but how can decimal numbers, such as 34.625_{10} be converted to binary fractions?
However, for the sake of completeness, here is a method for converting decimal fractions to binary fractions.
Converting the Decimal Integer to Binary



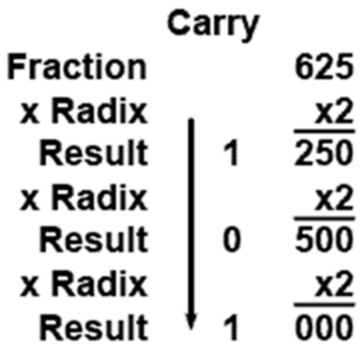
Example: Converting the Decimal Integer to Binary

The radix point splits the number into two parts; the part to the left of the radix point is called the Integer. The part to the right of the radix point is the Fraction. A number such as 34.62510 is therefore split into 3410 (the integer), and .62510 (the fraction).

To convert such a fractional decimal number to any other radix, the method described above is used to convert the integer.

So 3410 = 1000102

Converting the Decimal Fraction to Binary



Example: Converting the Fraction to Binary

To convert the fraction, this must be multiplied by the radix (in this case 2 to convert to binary). Notice that with each multiplication a carry is generated from the third column. The carry will be either 1 or 0 and these are written down at the left hand side of the result. However when each result is multiplied the carry is ignored (don't multiply the carry). Each result is multiplied in this way until the result (ignoring the carry) is 000. Conversion is now complete.

For the converted value just read the carry column from top to bottom

So 0.62510 = .1012

Therefore the complete conversion shows that 34.62510 = 100010.1012

However, with binary, there is a problem in using this method, .625 converted easily but many fractions will not. For example if you try to convert .626 using this method you would find that the binary fraction produced goes on to many, many places without a result of exactly 000 being reached.

Quick Conversions

The most commonly encountered number systems are binary and hexadecimal, and a quick method for converting to decimal is to use a simple table showing the column weights, as shown in tables.

Converting Binary to Decimal

Table								
Bit	27	26	25	24	23	22	21	20
Value of each bit	128	64	32	16	8	4	2	1
8 bit Binary	0	1	0	0	0	0	1	1

To convert from binary to decimal, write down the binary number giving each bit its correct 'weighting' i.e. the value of the columns, starting with a value of one for the right hand (least significant) bit. Giving each bit twice the value of the previous bit as you move left.

Example:

To convert the binary number 010000112 to decimal, write down the binary number and assign a 'weighting' to each bit as in Table

Now simply add up the values of each column containing a 1 bit, ignoring any columns containing 0.

Applying the appropriate weighting to 01000011 gives 64 + 2 + 1 = 67

Therefore: 010000112 = 6710

Converting Hexadecimal to Decimal

Table				
Column	163	162	161	160
Value of each column	4096	256	16	1
Hex value	2	5	C	B

A similar method can be used to quickly convert hexadecimal to decimal, using Table:

The hexadecimal digits are entered in the bottom row and then multiplied by the weighting value for that column.

Adding the values for each column gives the decimal value.

$$\begin{array}{rrr} 2 \times & 4096 = & 8192 \\ 5 \times & 256 = & 1280 \\ C\ (12_{10}) \times & 16 = & 192 \\ B\ (11_{10}) \times & 1 = & \underline{11} \\ & & \underline{9675} \end{array}$$

Therefore: $25CB_{16} = 9675_{10}$

Practice yourself: Conversion

11010011_2 to decimal

10111011_2 to decimal

$34F2_{16}$ to decimal

$FFFF_{16}$ to decimal

Binary to Hexadecimal

Converting between binary and hexadecimal is a much simpler process; hexadecimal is really just a system for displaying binary in a more readable form.

Binary is normally divided into Bytes (of 8 bits) it is convenient for machines but quite difficult for humans to read accurately. Hexadecimal groups each 8-bit byte into two 4-bit nibbles, and assigns a value of between 0 and 15 to each nibble. Therefore each hexadecimal digit (also worth 0 to 15) can directly represent one binary nibble. This reduces the eight bits of binary to just two hexadecimal characters.

Table	
Binary	Hexadecimal
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

For example:

111010012 is split into 4 bits from the LSB (least significant bits) and if there wouldn't have 4 bits on MSB (most significant bits) you can add zero in front of MSB for making 4 bits group.

Otherwise, 111010012 is split into 2 nibbles 11102 and 10012 then each nibble is assigned a hexadecimal value between 0 and F.

The bits in the most significant nibble (11102) add up to 8+4+2+0 = 1410 = E16

The bits in the least significant nibble (10012) add up to 8+0+0+1 = 910 = 916

Therefore 111010012 = E916

Converting hexadecimal to binary of course simply reverses this process.

For example: BAD16 to binary

Here, Binary value of B (11) = 10112

Binary value of A (10) = 10102

Binary value of D (13) = 11012

Then concatenate the nibble value as like B A D has 1011 1010 11012

Therefore, BAD16= 1011101011012

Binary to Octal conversion

Table	
Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

For example:

111010012 is split into 3 bits from the LSB (least significant bits) and if there wouldn't have 3 bits on MSB (most significant bits) you can add zero in front of MSB for making 3 bits group as 112 1012 0012 is equal to 0112 1012 0012 then each group is assigned a octal value between 0 and 7.

The bits in the most significant 3 bits group (0112) add up to 0+2+1 = 310 and next 3 bits group (1012) add up to 4+0+1 = 510

The bits in the least significant nibble (0012) add up to 0+0+1 = 110

Therefore 111010012 = 3518

Converting hexadecimal to binary of course simply reverses this process.

For example: 7018to binary

Here, Binary value of 7 in octal = 1112

Binary value of 0in octal = 0002

Binary value of 1in octal = 0012

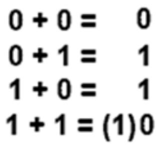
Then concatenate the 3 bit group value as like 7 0 1 has 111 000 0012

Therefore, 7018= 1110000012

Binary Arithmetic

Binary Addition Rules

Arithmetic rules for binary numbers are quite straightforward, and similar to those used in decimal arithmetic. The rules for addition of binary numbers

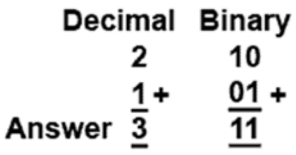


```
0 + 0 =   0
0 + 1 =   1
1 + 0 =   1
1 + 1 = (1)0
```

Fig. 2.1 Rules for Binary Addition

Notice that in Fig. 2.1, 1+1 = (1)0 requires a 'carry' of 1 to the next column. Remember that binary 102 = 210 decimal

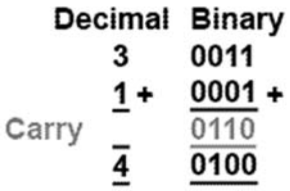
Example:



```
        Decimal  Binary
            2      10
            1 +    01 +
Answer      3      11
```

Fig. 2.2 Simple Binary Addition

Binary addition is carried out just like decimal, by adding up the columns, starting at the right and working column by column towards the left.



```
      Decimal  Binary
          3    0011
          1 +  0001 +
Carry     _    0110
          4    0100
```

Fig. 2.2 Binary Addition with Carry

Just as in decimal addition, it is sometimes necessary to use a 'carry', and the carry is added to the next column. For example, in Fig. 2.2 when two ones in the right-most column are added, the result is 210 or 102, the least significant bit of the answer is therefore 0 and the 1 becomes the carry bit to be added to the 1 in the next column.

Binary Subtraction

The rules for subtraction of binary numbers are again similar to decimal. When a large digit is to be subtracted from a smaller one, a ‘borrow’ is taken from the next column to the left. In decimal subtractions the digit ‘borrowed in’ is worth ten, but in binary subtractions the ‘borrowed in’ digit must be worth 210 or binary 102.

0 - 0 = 0
0 - 1 = 1*
1 - 0 = 1
1 - 1 = 0

*After 10_2 is borrowed from next column on left.

Fig. 2.3 Rules for Binary Addition

Binary Subtraction Rules

The rules for binary subtraction are quite straightforward except that when 1 is subtracted from 0, a borrow must be created from the next most significant column. This borrow is then worth 210 or 102 because a 1 bit in the next column to the left is always worth twice the value of the column on its right.

Fig 2.3 shows how binary subtraction works by subtracting 510 from 1110 in both decimal and binary. Notice that in the third column from the right (22), a borrow from the (23) column is made and then paid back in the MSB (23) column.

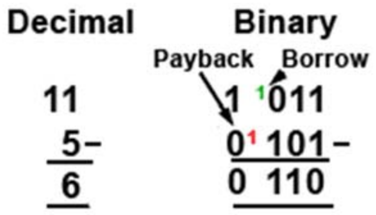


Fig. 2.3 Binary Subtraction

Subtraction Exercise

Give some question here:

$$\begin{array}{rrrr} \multicolumn{2}{l}{\textbf{4-bit Binary}} & \multicolumn{2}{l}{\textbf{8-bit Binary}} \\ 15 & 12 & 27 & 56 \\ \underline{\ 9}\ - & \underline{\ 3}\ - & \underline{17}\ - & \underline{31}\ - \\ \_\_ & \_\_ & \_\_ & \_\_ \end{array}$$

$$\begin{array}{rl} & \textbf{Binary} \\ & 0111 \\ & \underline{0011}\ + \\ \text{Carry} & \underline{1110} \\ & \underline{1010} \end{array}$$

Fig. 2.4 Limits of 4 Bit Arithmetic

Limitations of Binary Arithmetic

Now back to ADDITION to illustrate a problem with binary arithmetic. In Fig. 2.4 notice how the carry goes right up to the most significant bit.

This is not a problem with this example as the answer 10102 (1010) still fits within 4 bits, but what would happen if the total was greater than 1510?

$$\begin{array}{rl} & \textbf{Binary} \\ & 1111 \\ & \underline{0001}\ + \\ \text{Carry (1)} & \underline{1110} \\ (1) & \underline{0000} \end{array}$$

Fig. 2.5 The Overflow Problem

As shown in Fig 2.5 there are cases where a carry bit is created that will not fit into the 4-bit binary word. When arithmetic is carried out by electronic circuits, storage locations called registers are used that can hold only a definite number of bits. If the register can only hold four bits, then this example would raise a problem. The final carry bit is lost because it cannot be accommodated in the 4-bit register; therefore the answer will be wrong.

To handle larger numbers more bits must be used, but no matter how many bits are used, sooner or later there must be a limit. How numbers are held in a computer system depends largely on the size of the registers available and the method of storing data in them; however any electronic system will have a way of overcoming this 'overflow' problem, but will also have some limit to the accuracy of its arithmetic.

Signed Binary

Signed Binary Notation

There are a number of ways in which binary numbers can represent both positive and negative values, 8 bit systems for example normally use one bit of the byte to represent either + or − and the remaining 7 bits to give the value. One of the simplest of these systems is SIGNED BINARY, also often called 'Sign and Magnitude', which exists in several similar versions, but is commonly an 8 bit system that uses the most significant bit (MSB) to indicate a positive or a negative value. By convention, a 0 in this position indicates that the number given by the remaining 7 bits is positive, and a most significant bit of 1 indicates that the number is negative.

For example:

$+45_{10}$ in signed binary is $(0)0101101_2$

-45_{10} in signed binary is $(1)0101101_2$

Table 2.3.1

Binary	Decimal	Signed Binary
11111111	255	-127
11111110	254	-126
11111101	253	-125
↑	↑	↑
10000010	130	-2
10000001	129	-1
10000000	128	-0
01111111	127	+127

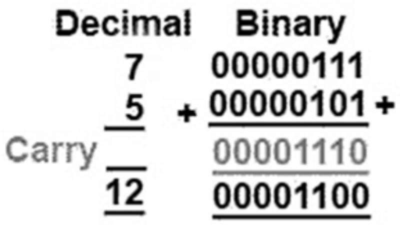
01111110	126	+126
↑	↑	↑
00000011	3	+3
00000010	2	+2
00000001	1	+1
00000000	0	+0

Note:

The brackets around the MSB (the sign bit) are included here for clarity but brackets are not normally used. Because only 7 bits are used for the actual number, the values the system can represent range from are -127_{10} or 11111111_2, to $+127_{10}$.

A comparison between signed binary, pure binary and decimal numbers is shown in Table2.3.1 Notice that in the signed binary representation of positive numbers between $+0_{10}$ and $+127_{10}$, all the positive values are just the same as in pure binary. However the pure binary values equivalents of $+128_{10}$ to $+255_{10}$ are now considered to represent negative values −0 to −127.

This also means that 0_{10} can be represented by 00000000_2 (which is also 0 in pure binary and in decimal) and by 10000000_2 (which is equivalent to 128 in pure binary and in decimal).

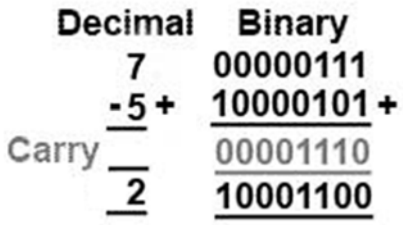


```
      Decimal   Binary
          7     00000111
          5   + 00000101 +
Carry           00001110
         12     00001100
```

Fig. 2.3.2 Adding Positive Numbers in Signed Binary

Signed Binary Arithmetic

Because the signed binary system now contains both positive and negative values, calculation performed with signed binary arithmetic should be more flexible.



	Decimal	Binary
	7	00000111
	-5 +	10000101 +
Carry		00001110
	2	10001100

Fig. 2.3.3 Adding Positive & Negative Numbers in Signed Binary

In Fig. 2.3.1 two positive (MSB = 0) numbers are added and the correct answer is obtained. This is really not different to adding two numbers in pure binary as described Number Systems above.

In Fig. 2.3.3 however, the negative number −5 is added to +7, the same action in fact as SUBTRACTING 5 from 7, which means that subtraction should be possible by merely adding a negative number to a positive number. Although this principle works in the decimal version the result using signed binary is 10001100_2 or -12_{10} which of course is wrong, the result of 7 − 5 should be +2.

Although signed binary can represent positive and negative numbers, if it is used for calculations, some special action would need to be taken, depending on the sign of the numbers used, and how the two values for 0 are handled, to obtain the correct result. Whilst signed binary does solve the problem of representing positive and negative numbers in binary, and to some extent carrying out binary arithmetic, there are better sign and magnitude systems for performing binary arithmetic.

Summary:

- Electronic systems may use a variety of different number systems, (e.g. Decimal, Hexadecimal, Octal, and Binary).
- The number system in use can be identified by its radix (10, 16, 8, 2).
- The individual digits of a number are scaled by the Radix Point.
- The Exponent is the system radix raised to a power dependent on the column value of a particular digit in the number.
- In Floating Point Notation the radix point can be moved to a new position without changing the value of the number if the Exponent of the number is also changed.

- In Normalized form, the radix point is always placed to the left of the most significant digit.
- When numbers are stored electronically they are stored in a register holding a finite number of digits; if the number stored has less digits than the register, non-significant zeros are added to fill spaces to the left of the stored number. Numbers containing more digits than the register can hold are broken up into register sized groups and stored in multiple locations
- Number systems are basically of two types: non-positional and positional.
- In a non – positional system, each symbol represents same value, regardless of its position in the number, and the symbols are simply added to find out the value of a particular number. It is very difficult to perform arithmetic with such a number system.
- In a positional number system, there are only few symbols called digits, and these symbols represent different values, depending on the position, they occupy in the number. The value of each digit in such a number is determined by three considerations:
    - The digit itself
    - The position of the digit in the number, and
    - The base of the number system.
- The number system that we use in our day-to –day life is called decimal number system. In this system, the base is equal to 10, because there are altogether ten symbols or digits(0,1,2,3,4,5,6,7,8,9).
- Some of the positional numbers system, which are used in the computer professionals, are binary (for which base is 2), octal (for which base is 8), and hexadecimal (for which base is 16).
- Students must know the techniques for the following types of conversions to convert numbers from one base to another.
    - Converting to decimal from another base.
    - Converting from decimal to another base.
    - Converting from a base other than 10, to a base other than 10.
    - Shortcut method for binary to octal conversion.
    - Shortcut method for octal to binary conversion.
    - Shortcut method for binary to hexadecimal conversion.

- Shortcut method to hexadecimal to binary conversion.

Self evaluation:

(Short Answer Questions)

1. What is the difference between positional and non-positional number systems? Give examples of both types of numbers systems.
2. What is meant by the base of number system? Give examples to illustrate the role of base in positional numbers system.
3. What is the value of the base for decimal, hexadecimal, binary and octal number systems?
4. Why are octal and /or hexadecimal number systems used as shortcut notations?
5. Find out the decimal equivalent of the following binary numbers:

   1101011 b) 11011101

   11101101 d) 1000

   10110001100 f) 110001

   10110001100 h) 111
6. Find out the octal equivalent of the binary numbers of the question 5.
7. Find out the hexadecimal equivalent of the binary number of question 5.
8. Convert the following numbers to decimal numbers:

   1101102 b) 25736 c) 2A3B16 d) 12349
9. Convert the following decimal numbers to binary numbers.

   34510 b) 169410 c) 3210 d) 13510
10. Convert the decimal number of question 9 to octal numbers.
11. Convert the decimal numbers of question 9 to hexadecimal numbers.
12. Carry out the following conversions:

    1256 =?4 b) 249 =?3 c) ABC16 =?8
13. Convert the following numbers to their binary equivalent:

    2AC16 b) FAB16 c) 26148 d) 5628
14. Find the decimal equivalent of the following numbers:

    111.012 b) 1001.0112 c) 247.658 d) A2B.D416

UNIT-2
Binary Arithmetic Operations

Objectives:

After competitions of this unit students will be able to

1. Understand the concept of One's complement and two's complements.
2. Perform addition and subtraction processor using One's complement and Two's Complements.
3. Perform the multiplication, division and memory overflow problem.
4. To operate the Binary Decimal Code.

One's and Two's Complement

One's Complement

The complement (or opposite) of +5 is −5. When representing positive and negative numbers in 8-bit ones complement binary form, the positive numbers are the same as in signed binary notation described in number systems i.e. the numbers 0 to +127 are represented as 000000002 to 011111112. However, the complement of these numbers, that is their negative counterparts from −128 to −1, are represented by 'complementing' each 1 bit of the positive binary number to 0 and each 0 to 1.

For example:

+510 is 000001012

−510 is 111110102

Notice in the above example, that the most significant bit (MSB) in the negative number −510 is 1, just as in signed binary. The remaining 7 bits of the negative number however are not the same as in signed binary notation. They are just the complement of the remaining 7 bits, and these give the value or magnitude of the number.

The problem with signed the binary arithmetic described in number system was that it gave the wrong answer when adding positive and negative numbers. Does ones complement notation give better results with negative numbers than signed binary?

```
Decimal   Binary
   +6     00000110
   -4 +   11111011 +
Carry __ (1)11111100
   +2     00000001
```

Figure. Adding Positive & Negative Numbers in Ones Complement

Figure. 1.5.1 shows the result of adding −4 to +6, using ones complement, (this is the same as subtracting +4 from +6, and so it is crucial to arithmetic).

The result, 000000012 is 110 instead of 210.

This is better than subtraction in signed binary, but it is still not correct. The result should be +210 but the result is +1 (notice that there has also been a carry into the none existent 9th bit).

Figure. shows another example, this time adding two negative numbers −4 and −3.

Because both numbers are negative, they are first converted to ones complement notation.

+410 is 00000100 in pure 8 bit binary, so complementing gives 11111011.

```
Decimal   Binary
   -4     11111011
   -3 +   11111100 +
Carry __ (1)11110000
   - 7    11110111
```

Figure. Adding Positive & Negative Numbers in Ones Complement

This is −410 in ones complement notation.
+310 is 00000011 in pure 8 bit binary, so complementing gives 11111100.
This is −310 in ones complement notation.

The result of 111101112 is in its complemented form so the 7 bits after the sign bit (1110111), should be re-complemented and read as 0001000, which gives the value 810. As the most significant bit (MSB) of the result is 1 the result must be negative, which is correct, but the remaining seven bits give the value of −8. This is still wrong by 1, it should be −7.

End around Carry

There is a way to correct this however. Whenever the ones complement system handles negative numbers, the result is 1 less than it should be, e.g. 1 instead of 2 and −8 instead of −7, but another thing that happens in negative number ones complement calculations is that a carry is 'left over' after the most significant bits are added. Instead of just disregarding this carry bit, it can be added to the least significant bit of the result to correct the value. This process is called 'end around carry' and corrects for the result -1 effect of the ones complement system.

However, there are still problems with both ones complement and signed binary notation. The ones complement system still has two ways of writing 010 (000000002 = +010 and 111111112 = −010); additionally there is a problem with the way positive and negative numbers are written. In any number system, the positive and negative versions of the same number should add to produce zero. As can be seen from Table, adding +45 and −45 in decimal produces a result of zero, but this is not the case in either signed binary or ones complement.

Table

	Decimal	Signed Binary	1's complement
	+45	00101101	00101101
	-45	10101101	11010010
Binary sum		11011010	11111111
Decimal	010	-9010	-12710

This is not good enough, however there is a system that overcomes this difficulty and allows correct operation using both positive and negative numbers. This is the Twos Complement system.

Two's Complement Notation

Twos complement notation solves the problem of the relationship between positive and negative numbers, and achieves accurate results in subtractions.

To perform binary subtraction, the twos complement system uses the technique of complementing the number to be subtracted. In the ones complement system this produced a result that was 1 less than the correct answer, but this could be corrected by using the 'end around carry' system. This still left the problem that positive and negative versions of the same number did not produce zero when added together.

The two's complement system overcomes both of these problems by simply adding one to the ones complement version of the number before addition takes place. The process of producing a negative number in Two's Complement Notation is illustrated in Table

Table	
Producing a Two's complement Negative Number	
+5 in 8 bit binary (or 8-bit signed Binary) is	00000101
Complementing to produce the One's complement	11111010
With 1 added	1
So -5 in two's complement is	11111011

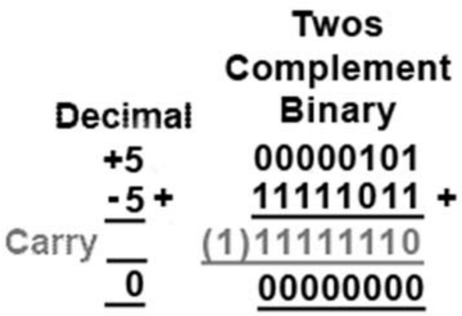


Figure. Adding a Number to its Two's Complement Produces Zero

This version of −5 now, not only gives the correct answer when used in subtractions but is also the additive inverse of +5 i.e. when added to +5 produces the correct result of 0, as shown in Figure.

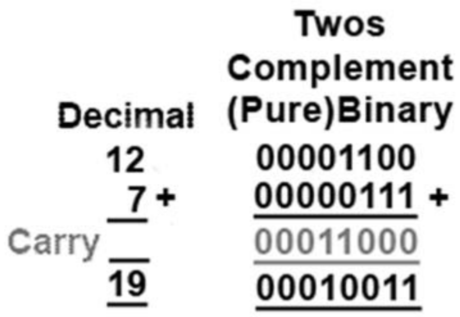
Note that in two's complement the (1) carry from the most significant bit is discarded as there is no need for the ‘end around carry’ fix.

With numbers electronically stored in their two's complement form, subtractions can be carried out more easily (and faster) as the microprocessor has simply to add two numbers together using nearly the same circuitry as is used for addition.

6 − 2 = 4 is the same as (+6) + (−2) = 4

Two's Complement Examples

Note: When working with two's complement it is important to write numbers in their full 8 bit form, since complementing will change any leading 0 bits into 1 bits, which will be included in any calculation. Also during addition, carry bits can extend into leading 0 bits or sign bits, and this can affect the answer in unexpected ways.



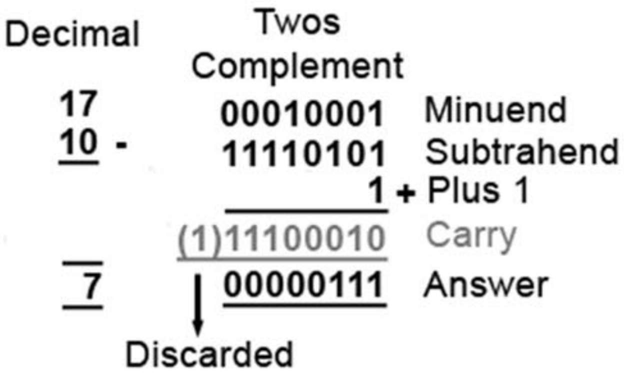
	Decimal	Twos Complement (Pure)Binary
	12	00001100
	7 +	00000111 +
Carry		00011000
	19	00010011

Figure. Adding Positive Numbers in Two's Complement

Two's Complement Addition

Fig shows an example of addition using 8 bit twos complement notation. When adding two positive numbers, there the sign bits (msb) will both be 0, so the numbers are written and added as a pure 8-bit binary addition.

Two's Complement Subtraction



Subtracting a Positive Number from a Larger Positive Number

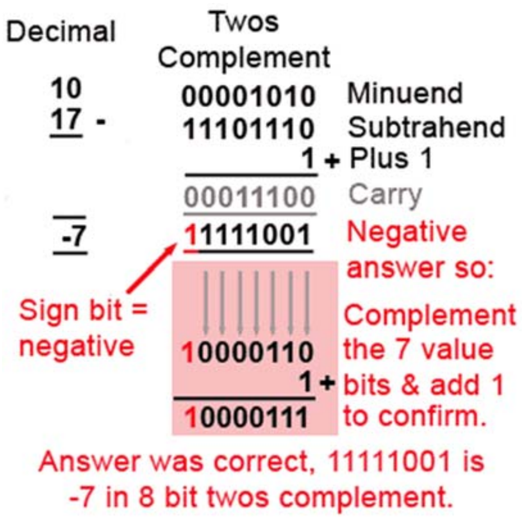
Figure. shows the simplest case of twos complement subtraction where one positive number (the subtrahend) is subtracted from a larger positive number (the minuend). In this case the minuend is 1710 and the subtrahend is 1010.

Because the minuend is a positive number its sign bit (MSB) is 0 and so it can be written as a pure 8 bit binary number.

The subtrahend is to be subtracted from the minuend and so needs to be complemented (simple ones complement) and 1 added to the least significant bit (1SB) to complete the twos complement and turn +10 into −10.

When these three lines of digits and any carry 1 bits are added, remembering that in twos complement, any carry from the most significant bit is discarded. The answer (the difference between 17 and 10) is 000001112 = 710 which is correct. Therefore the twos complement method has provided correct subtraction by using only addition and complementing, both operations that can be simply accomplished by digital electronic circuits.

Subtraction with a negative result


Decimal
Twos Complement
10
17 -
00001010 Minuend
11101110 Subtrahend
1 + Plus 1
00011100 Carry
-7
11111001 Negative answer so:
Sign bit = negative
Complement the 7 value bits & add 1 to confirm.
10000110
1 +
10000111
Answer was correct, 11111001 is -7 in 8 bit twos complement.

Subtraction Producing a Negative Result

Some subtractions will of course produce an answer with a negative value. In Figure. 1.5.6 the result of subtracting 17 from 10 should be −710 but the twos complement answer of 111110012 certainly doesn't look like −7. However the sign bit is indicating correctly that the answer is negative, so in this case the 7 bits indicating the value of the negative answer need to be 'twos complemented' once more to see the answer in a recognisable form.

When the 7 value bits are complemented and 1 is added to the least significant bit however, like magic, the answer of 100001112 appears, which confirms that the original answer was in fact −7 in 8 bit twos complement form.

It seems then, that two's complement will get the right answer in every situation?

Well guess what – it doesn't! There are some cases where even twos complement will give a wrong answer. In fact there are four conditions where a wrong answer may crop up:

1. When adding large positive numbers.
2. When adding large negative numbers.
3. When subtracting a large negative number from a large positive number.
4. When subtracting a large positive number from a large negative number.

The problem seems to be with the word ‘large’. What is large depends on the size of the digital word the microprocessor uses for calculation. As shown in Table, if the microprocessor uses an 8−bit word, the largest positive number that can appear in the problem OR THE RESULT is +12710 and the largest negative number will be −12810. The range of positive values appears to be 1 less than the negative range because 0 is a positive number in twos complement and has only one occurrence (000000002) in the whole range of 25610 values.

With a 16-bit word length the largest positive and negative numbers will be +3276710 and -3276810, but there is still a limit to the largest number that can appear in a single calculation.

Table	
2's complement (8 bits)	Decimal values
01111111	+127
01111110	+126
011111101	+125
↓	↓
00000010	+2
00000001	+1
00000000	0
11111111	-1
11111110	-2
↑	↑
1000010	-126
1000001	-127
1000000	-128

Example-1: Perform 2’s complement subtraction of (7)10 − (11)10.

Solution:

2’s Complements Subtraction of (7)10 – (11)10

First convert the decimal numbers 7 and 11 to its binary equivalents.

(7)10 = (0111)2

(11)10 = (1011)2 in 4-bit system

Then find out the 2’s complement for 1011 i.e,

1’s Complement of 1011 is 0100

2’s Complement of 1011 is 0101

So, (7)10 – (11)10 =

```
 0111
0101
---------
11002
---------
```

Example-2: Perform the following subtractions using 2’s complement of binary method.

i. 01000 – 01001
ii. (ii) 01100 – 00011
iii. (iii) 0011.1001 – 0001.1110

Solution:

i) Subtraction of 01000-01001: 1’s complement of 01001 is 10110 and 2’s Complement is

10110+ 1 =10111.

Hence

```
01000 = 01000
- 01001 = +10111 (2's complement)
-------------------------
11111 (Summation)
-------------------------
```

Since the MSB of the sum is 1, which means the result is negative and it is in 2's

Complement form. So, 2's complement of 1111 =00001= (1)10. Therefore, the result is – 1.

ii) Subtraction of 01100-00011: 1’s complement of 00011 is 11100 and 2’s complement is

11100 + 1 = 11101. Hence

01100 = 01100

– 00011 = + 11101 (2's complement)

----

1 01001 = + 9

Ignore

If a final carry is generated discard the carry and the answer is given by the remaining bits

Which is positive i.e., (1001)2 = (+ 9)10

Subtraction of 0011.1001 – 0001.1110: 1’s complement of 0001.1110 is 1110.0001

And

Its 2’s complement is 1110.0010.

0011.1001 = 0011.1001

- 0001.1110 = + 1110.1011 (2’s complement)

----

1 0001.101I = + 1 .68625

↑

Ignore

If a final carry is generated discard the carry and the answer is given by the remaining

Bits which is positive i.e., (0001.1011)2 = (+ 1.68625)10

Example-3 Perform the following additions using 2’s complement.

-20 to +26

(ii) +25 to -15

Solution:

(i) First convert the two numbers 20 and 26 into its 8-bit binary equivalent and find out the

2's complement of 20, then add -20 to +26.

20 = 0 0 0 1 0 1 0 0 (8-bit binary equivalent of 20)

20 = 1 1 1 0 1 0 1 1 (1's complement)

+1

----

20 = -20 = 1 1 1 0 1 1 0 0 (2's complement of 20)

+26 = 0 0 0 1 1 0 1 0 (8-bit binary equivalent of 26)

Addition of -20 to +26

= +6 = 0 0 0 0 0 1 1 0

Hence -20 to +26 = (6)10 = (0110)2.

(ii) First convert the two numbers 25 and 15 into its 8-bit binary equivalent and find out

the 2's complement of 15, then add +25 to -15.

15 = 0 0 0 0 1 1 1 1 (8-bit binary equivalent of 15)

15 = 1 1 1 1 0 0 0 0 (1's complement)

+1

----

15 = -15 = 1 1 1 1 0 0 0 1 (2's complement of 15)

+25 = 0 0 0 1 1 0 0 1 (8-bit binary equivalent of 25)

Addition of -15 to +25

= +10 = 0 0 0 0 1 0 1 0

Hence -15 to +25 = (10)10 = (1010)2.

Example-4 Perform following subtraction

(i) 11001-10110 using 1's complement

(ii) 11011-11001 using 2's complement

Solution:

(i) 11001 - 10110

1's Complement of 10110 = 01001

```
1 1 0 0 1
+ 0 1 0 0 1
-----------------
1 0 0 0 1 0
```

Add 1 and ignore carry.

Ans is 00011 = 3.

(ii) 11011 – 11001 = A – B

2's complement of B = 00111

```
1 1 0 1 1
+ 0 0 1 1 1
―――――――――
1 0 0 0 1 0
```

Ignore carry to get answer as 00010 = 2.

Example-5 Perform the following operations using the 2's complement method:

(i) 23 – 48

(ii) – 48 – 23

Solution:

(i) 23 – 48   add them

```
 23                    0 1 0 1 1 1
 - (- 48)            + 0 1 0 0 0 0
―――――――              ―――――――――――
 71                    1 0 0 1 1 1
```

– 48 - 23 = - 48 + (-23)

-48 = 1 1 0 1 0 0 0 0

-23 = 1 1 1 0 1 0 0 1

1 1 0 1 1 1 0 0 1 = -71

$\downarrow$

Carry is discarded

Overflow Problems.

Steps can be taken to accommodate large numbers, by breaking a long binary word down into byte sized sections and carrying out several separate calculations before assembling the final answer. However this doesn't solve all the cases where errors can occur.

A typical overflow problem that can happen even with single byte numbers is illustrated in Figure.

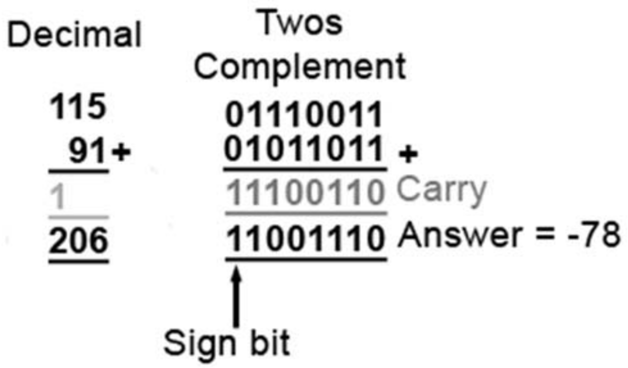


Figure. Carry Overflows into Sign Bit

In this example, the two numbers to be added (11510 and 9110) should give a sum of 20610 and at first glance 110011102 looks like the correct answer of 20610, but remember that in the 8 bit twos complement system the most significant bit is the sign of the number, therefore the answer appears to be a negative value and reading just the lower 7 bits gives 10011102 or -7810. Although twos complement negative answers are not easy to read, this is clearly wrong as the result of adding two positive numbers must give a positive answer.

According to the information in Fig 1.5.6, as the answer is negative, complementing the lower 7 bits of 110011102 and adding 1 should reveal the value of the correct answer, but carrying out the complement+1 on these bits and leaving the msb unchanged gives 101100102which is -5010. This is nothing like the correct answer of 20610 so what has happened?

The 8 bit twos complement notation has not worked here because adding 115 + 91 gives a total greater than +127, the largest value that can be held in 8-bit two's complement notation.

What has happened is that an overflow has occurred, due to a 1 being carried from bit 6 to bit 7 (the most significant bit, which is of course the sign bit), this changes the sign of the answer. Additionally it changes the value of the answer by 12810 because that would be the value of the msb in pure binary. So the original answer of 7810 has 'lost' 12810 to the sign bit. The addition would have been correct if the sign bit had been part of the value, however the calculation was done in twos complement notation and the sign bit is not part of the value.

Of course in real electronic calculations, a single byte overflow situation does not usually cause a problem; computers and calculators can fortunately deal with larger numbers than 12710. They achieve this because the microprocessors used are programmed to carry out the calculation in a number of steps, and although each step must still be carried out in a register having a set word length, e.g. 8 bits or 16 bits, corrective action can also be taken if an overflow situation is detected at any stage.

Multiplication and Division

While addition and subtraction can be achieved by adding positive and negative numbers as described above, this does not include the other basic forms of mathematics, multiplication and division. Multiplication in its simplest form can however be achieved by adding a number to itself a number of times, for example, starting with a total of 0, if 5 is added to the total three times the new total will be fifteen (or 5 x 3). Division can also be accomplished by repeatedly subtracting (using add) the divisor from the number to be divided until the remainder is zero, or less than the divisor. Counting the number of subtractions then gives the result, for example if 3 (the divisor) is repeatedly subtracted from 15, after 5 subtractions the remainder will be zero and the count will be 5, indicating that 15 divided by 3 is exactly 5.

Need the example.

Binary Coded Decimal (BCD)

Representing Decimal Numbers

A binary number with its bits representing values of 1, 2, 4, 8, 16 etc. presents problems. It would be better if a particular number of binary bits could represent the numbers 0 to 9, but this doesn't happen in pure binary, a 3 bit binary number represents the values 0 to 7 and 4 bit represents 0 to 15. What is needed is a system where a group of binary digits can represent the decimal numbers 0-9, and the next group 10-90 etc.

To make this possible, binary codes are used that have ten values, but where each value is represented by the 1s and 0s of a binary code. These special 'half way' codes are called Binary Coded Decimal or BCD. There are several different BCD codes, but they have a basic similarity. Each of the ten decimal digits 0 to 9 is represented by a group of 4 binary bits, but in codes the binary equivalents of the 10 decimal numbers do not necessarily need to be in a consecutive order.

There can be advantages in some specialist applications in using some particular variation of BCD. For example it may be useful to have a BCD code that can be used for calculations, which means having positive and negative values, similar to the twos complement system, but BCD codes are most often used for the display of decimal digits. The most commonly encountered version of BCD binary code is the BCD8421 code. In this version the numbers 0 to 9 are represented by their pure binary equivalents, 4 bits per decimal number, in consecutive order.

BCD Codes

The BCD8421 code is so called because each of the four bits is given a 'weighting' according to its column value in the binary system. The least significant bit (lsb) has the weight or value 1, the next bit, going left, the value 2. The next bit has the value 4, and the most significant bit (msb) the value 8, as shown in following Table.

Table				
	MSB	BCD8421		LSB
Decimal	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

So the 8421BCD code for the decimal number 610 is 01108421. Check this from Table.

For numbers greater than 9 the system is extended by using a second block of 4 bits to represent tens and a third block to represent hundreds etc.

2410 in 8 bit binary would be 00011000 but in BCD8421 is 0010 0100.

99210 in 16 bit binary would be 00000011111000002 but in BCD8421 is 1001 1001 0010.

Therefore BCD acts as a half way stage between binary and true decimal representation, often preparing the result of a pure binary calculation for display on a decimal numerical display. Although BCD can be used in calculation, the values are not the same as pure binary and must be treated differently if correct results are to be obtained. The facility to make calculations in BCD is included in some microprocessors.

One of the main drawbacks of BCD is that, because sixteen values are available from four bits, but only ten are used, there are several redundant values whichever BCD system is used. This is wasteful in terms of circuitry, as the fourth bit (the 8s column) is under used.

Summary:

- The one's complement of a binary number is defined as the value obtained by inverting all the bits in the binary representation of the number (swapping 0s for 1s and vice versa). The ones' complement of the number then behaves like the negative of the original number in some arithmetic operations.
- Two's complement is a mathematical operation on binary numbers, as well as a binary signed number representation based on this operation. Its wide use in computing makes it the most important example of a radix complement.
- The two's complement of an N-bit number is defined as the complement with respect to 2N; This is also equivalent to taking the ones' complement and then adding one
- Two’s complement is found by first finding 1’s complement and then adding 0001 b.
- Two’s complement gives negative of a given number
- Adding a number with its two’s complement gives all bits = 0s
- In subtraction by 1’s complement we subtract two binary numbers using carried by 1’s complement.

The steps to be followed in subtraction by 1’s complement are:

- To write down 1’s complement of the subtrahend.
- To add this with the minuend.
- If the result of addition has a carryover then it is dropped and an 1 is added in the last bit.
- If there is no carry over, then 1’s complement of the result of addition is obtained to get the final result and it is negative.
- With the help of subtraction by 2’s complement method we can easily subtract two binary numbers.

The operation is carried out by means of the following steps:

- At first, 2's complement of the subtrahend is found.
- Then it is added to the minuend.
- If the final carryover of the sum is 1, it is dropped and the result is positive.
- If there is no carry over, the two's complement of the sum will be the result and it is negative.

Self-Evaluation:

(Short Answer Questions)

- Add the binary numbers 1011 and 101 in both decimal and binary forms.
- Add the binary numbers 1010110 and 1011010.
- Add the binary numbers 10111 and 1011.
- Find the complements of the following numbers:

  495_{10} b)29_{10} c) 4_8 d)c_{16} e) 2_5 32_4
- Find the complements of the following binary numbers:

  a) 10 b) 101 c)101101 d) 011100

  e) 10110001 f) 001101001110
- Subtract 01101112 from 11011102.
- Subtract 0110112 from 100002.
- Subtract 2510 from 5010 using complementary method.
- Subtract 23410 from 58810 using complementary method.
- Subtract 010102 from 100002 using complementary method.
- Multiply the binary numbers 1100 and 1010.
- Multiply the binary numbers 101111 and 111.
- Divide 110012 by 1012.
- Divide 01101112 by 01112.
- What are the primary advantages of performing subtraction by the complementary method in digital computers?

UNIT-3

Logic Gate Concepts

Learning Outcomes

After completion of this units students will be able to

- Demonstrate the concept of basics of different types of logic gates.
- Explain Clear Concept of De Morgan's Theorems.

Introduction

Digital Logic Gates

The Digital Logic Gate is the basic building block from which all digital electronic circuits and microprocessor based systems are constructed from. Basic digital logic gates perform logical operations of AND, OR and NOT on binary numbers.

In digital logic design only two voltage levels or states are allowed and these states are generally referred to as Logic "1" and Logic "0", High and Low, or True and False. These two states are represented in Boolean Algebra and standard truth tables by the binary digits of "1" and "0" respectively.

A good example of a digital state is a simple light switch as it is either "ON" or "OFF" but not both at the same time. Then we can summarize the relationship between these various digital states as being:

Boolean Algebra	Boolean Logic	Voltage State
Logic "1"	True (T)	High (H)
Logic "0"	False (F)	Low (L)

table 1

Most digital logic gates and digital logic systems use "Positive logic", in which a logic level "0" or "LOW" is represented by a zero voltage, 0v or ground and a logic level "1" or "HIGH" is represented by a higher voltage such as +5 volts, with the switching from one voltage level to the other, from either a logic level “0” to a “1” or a “1” to a “0” .

There are 7 different gates, they are :-

i) OR ii) AND iii) NOT iv) NOR v) NAND vi) XOR vii) XNOR

- NOR and XNOR gates are Known as universal gates.

Truth Table

Truth table is the representation of inputs and outputs of Boolean variables and calculation by using Boolean operators in tabular format. It simply takes possible combination of inputs (in T and F or 1 and 0 formats) and generates only one corresponding output by using Boolean operators. For example:

A	B	O=A.B
F	F	F
F	T	F
T	F	F
T	T	T

Table 2

Where A and B are inputs variables which only has one of the two possible values T or F(1 or 0) and only one output 0.

Notations

Gates are identified by their function: AND, OR, NOT, NAND, NOR, X-OR and X-NOR. Capital letters are normally used to make it clear that the term refers to a logic gate. The above said logic gates can be classified into following categories:

1. Basic Logic Gates

a. AND Gate

b. OR Gate

c. NOT Gate

2. Universal Gates

a. NAND Gate
b. NOR Gate

3. Combinational Gates

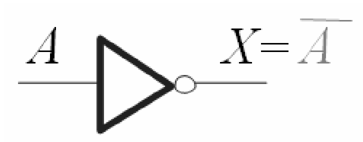
a. X-OR Gate
b. X-NOR Gate

Inverter (NOT gate)

NOT gate / Inverter

NOT gate is also called inverter. Operation of NOT gate takes place like this: if input is high output is low; and if input is low, output is high. In other words output is always inverse or toggle of input. So it is called inverter. Hence, a NOT gate always has a single input. Fig 1 also shows that two NOT gates connected in series to give an output equal to the input.

Circuit diagram of inverter is given below



Mathematical representation

Fig 1

x=$\bar{A}$

NOT gates connected in series

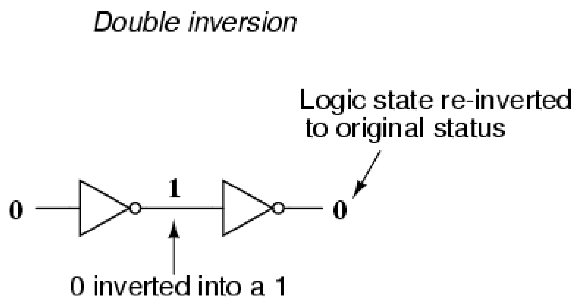


Fig 2

Here input is A output is x=A'

Truth table of NOT gate:

A	B'
0	1
1	0

Table 2

Electric diagram of NOT gate

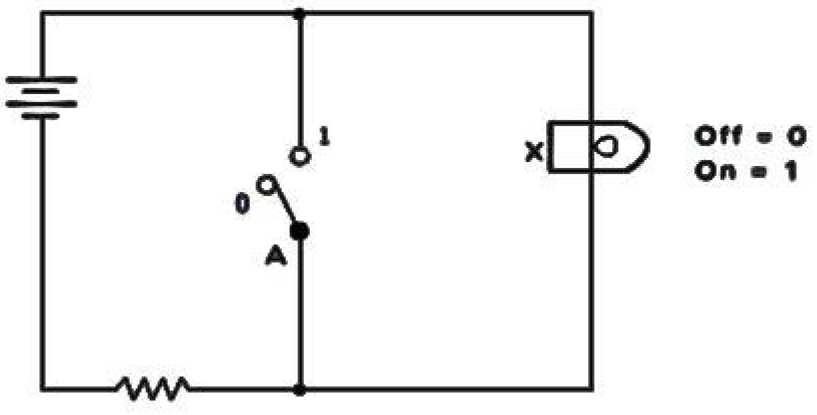


Fig 3

Venn Diagram of NOT Gate

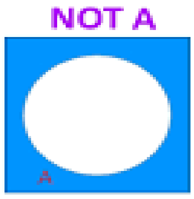


Fig 4

AND Gate

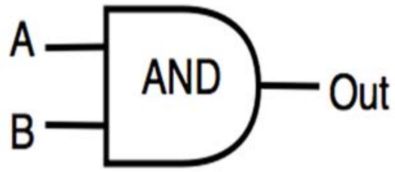
AND gate is the physical realization of logical multiplication. It is an electronic circuit that generates an output signal of 1, only if all input signals are also 1. Two or more switches connected in series behave as an AND gate.

Mathematically it is denoted as

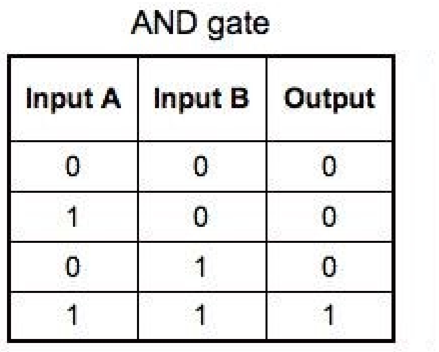
$$A.B=C$$

Where. represent AND gate

Logical Diagram



Truth Table



AND gate

Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

Electrical Circuit Design of AND Gate

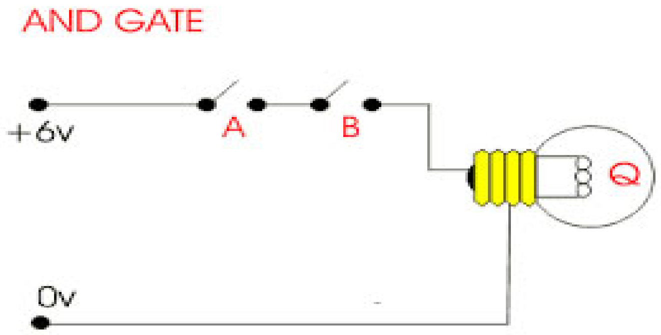


fig. Electrical Circuit Design

Ven diagram of AND gate

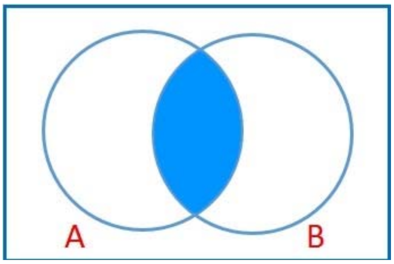


fig. Ven diagram of AND gate.

OR Gate:

An OR gate is the physical realization of logical addition. It is an electronic circuit that generates an output signal of 1, if any of the input signals is also 1.

Two or more switches connected in parallel behave as an OR gate. Observe from Figure that the input current will reach the output point when any one or both switches (A and B) are in ON(1) state . There will be no output only when both the switches are in OFF (0) state.

Mathematically it is denoted as

$$\mathbf{A+B= C}$$

Where + denote as OR Gate

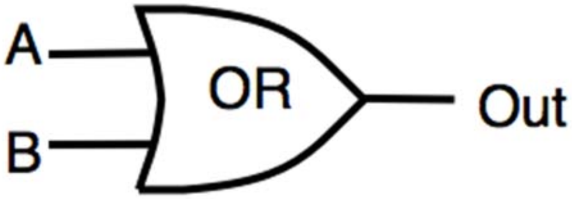


fig: logical circuit Design

Truth Table of OR gate:

OR gate

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	1

Electrical Circuit Design of OR Gate

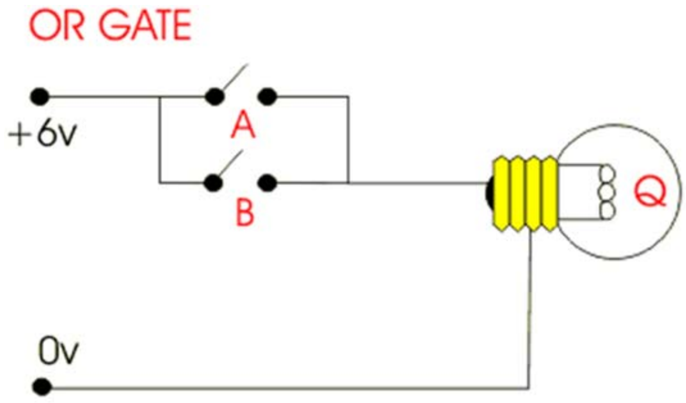


fig: Electrical Circuit Design of OR Gate

Venn diagram of OR gate

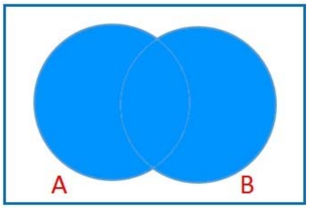


Fig. Venn diagram of OR gate

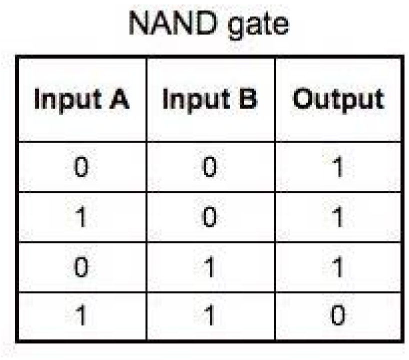
NAND Gate

NAND Gate is constructed by attaching NOT gate at the output of AND Gate, hence NAND Gate is called NOT-AND Gate. NAND Gate has two or more inputs and only one output. The output of NAND Gate is 0 (False) only when both or all inputs are 1(True) otherwise output are 1(True) .

Mathematically it is denoted as

$$\overline{A.B}=C$$

Logical Diagram of NAND



NAND gate

Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

Truth Table of NAND

Electrical Circuit Design of NAND Gate

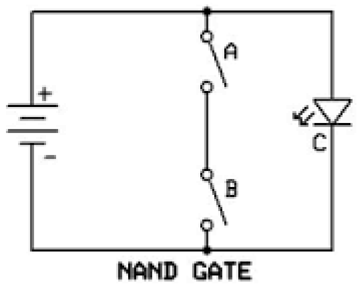


fig: Electrical Circuit Design of NAND

Ven Diagram of NAND

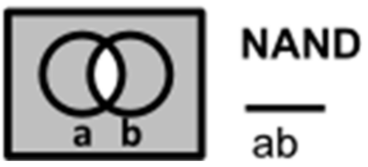


Fig. Ven Diagram of NAND

NOR Gate

NOR Gate is the combination of NOT gate at the output of OR Gate, hence NOR Gate is a type of NOT - OR Gate. NOR Gate has two or more input and only one output.

The output of NOR Gate is 1 (True) only when both or all inputs are 0 (False) otherwise outputs are 0 (False).

Mathematically it is denoted as

$$\overline{A + B} = C$$

Logical diagram

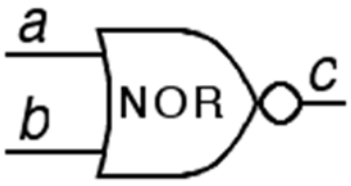
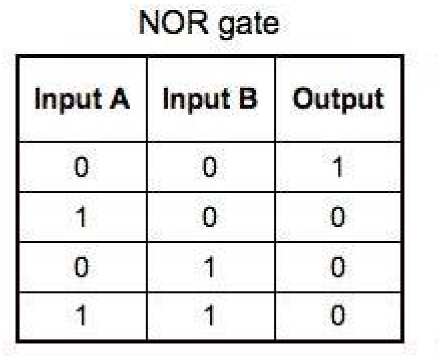


Fig: Logical diagram

Truth Table



NOR gate

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	0

Truth Table

Ven Diagram of NOR

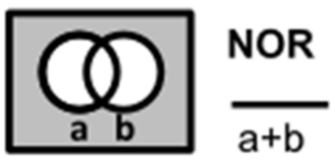


Fig. Ven Diagram of NOR

XOR Gate

It stands for exclusive OR Gate. XOR Gate has two inputs and only one output. The output of XOR Gate is 1 (true) only when both inputs are different otherwise output is 0 (false). XOR Gate is represented by $\oplus$

Mathematically it is represented as

$$\mathbf{A \oplus B = C}$$

Logical Diagram

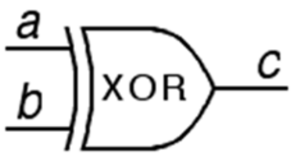
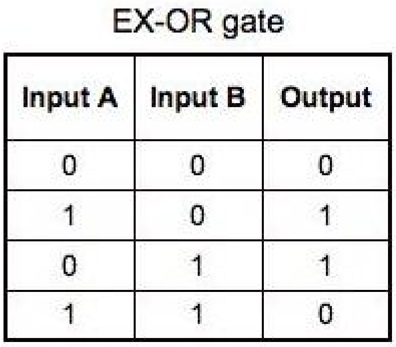


Fig: Logical Diagram of XOR

Truth Table



EX-OR gate

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

Truth Table of EX-OR

Ven Diagram of XOR Gate

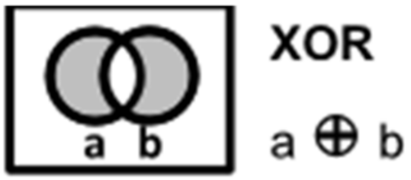


Fig. Ven Diagram of XOR

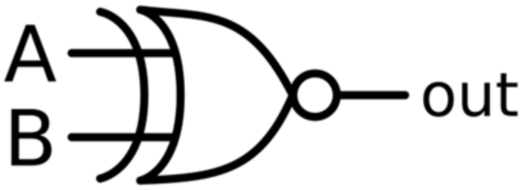
XNOR Gate

XNOR stands for exclusive NOR Gate. XNOR Gate may have two or more outputs but gives only one output. The output of XNOR Gate is 1 (True) only when both inputs are same otherwise outputs are 0 (False).

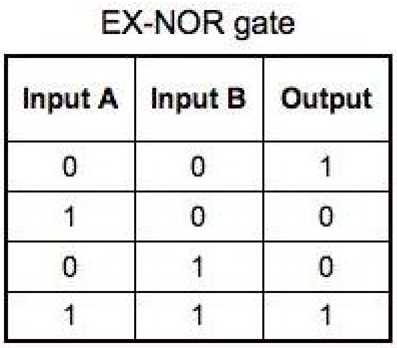
Mathematically it is denoted as

$$\overline{\boldsymbol{A} \oplus \boldsymbol{B}} = \mathrm{C}$$

Logical Diagram



Truth Table



EX-NOR gate

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	1

fig: Truth Table

Ven Diagram of XNOR Gate

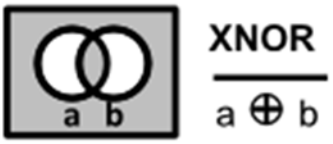


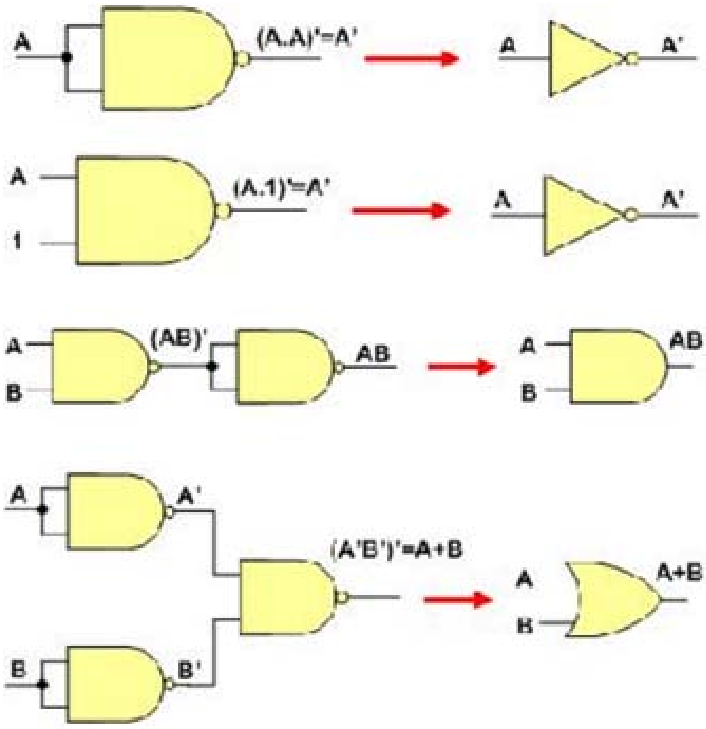
Fig. Ven Diagram of XNOR

Universal Gate

A universal gate is a gate which can implement any Boolean function without using any other gate type. The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families. In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not the other way around Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter not the other way around!!

NAND Gate is a Universal Gate:

To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

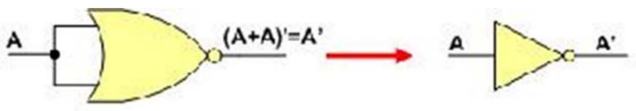


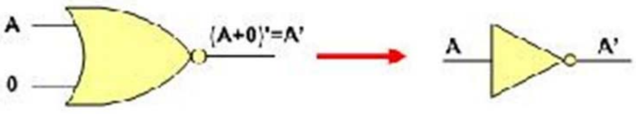
NOR Gate is a Universal Gate.

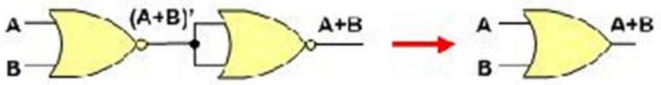
To prove that any Boolean function can be implemented using only NOR gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

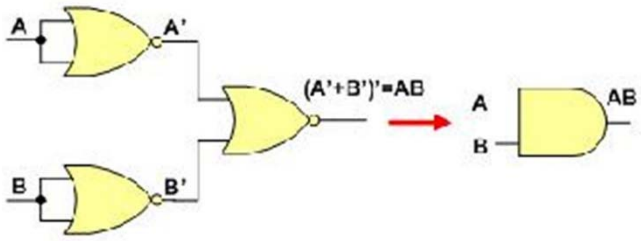
Implementing an Inverter Using only NOR Gate

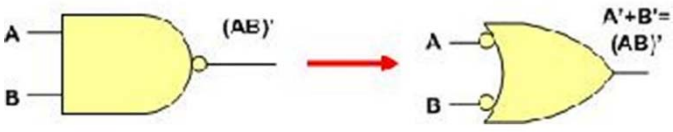
The figure shows two ways in which a NOR gate can be used as **an inverter (NOT gate)**.











De Morgan's Theorems Concept

De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

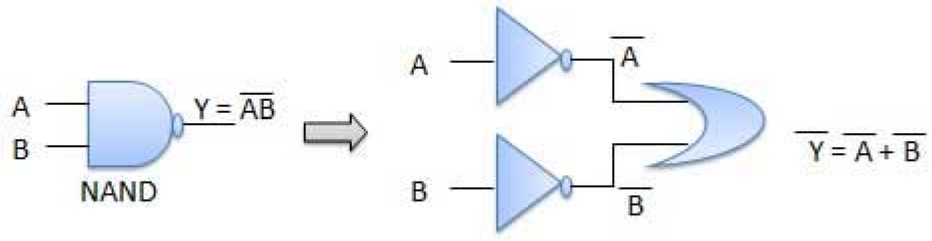
Theorem 1

The complements of the sum of the Boolean variables are equal to the product of their individual complements. i.e.

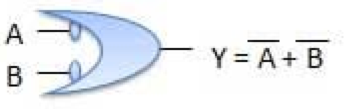
$$\overline{A.B} = \overline{A} + \overline{B}$$

NAND = Bubbled OR

- The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inputs inverted inputs.
- This OR gate is called as **Bubbled OR**.

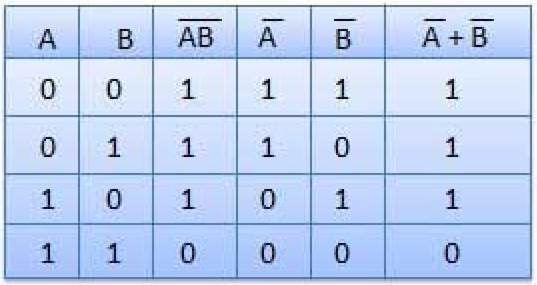


NAND $\equiv$ Bubbled OR



Bubbled OR

Table showing verification of the De Morgan's first theorem −



A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Table: verification of the De Morgan's

since the values in the $\overline{A.B}$ and $\bar{A}+\bar{B}$ are the same for all input combination, we can conclude that the two expression are logically equivalent. Hence $\overline{A.B} = \bar{A}+\bar{B}$, theorem 1 is proved.

Theorem 2

The complements of the product of the Boolean variables are equal to the sum of their individual complements

i.e

$$\overline{A+B} = \overline{A}.\overline{B}$$

NOR = Bubbled AND

- The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.
- This AND gate is called as **Bubbled AND**.

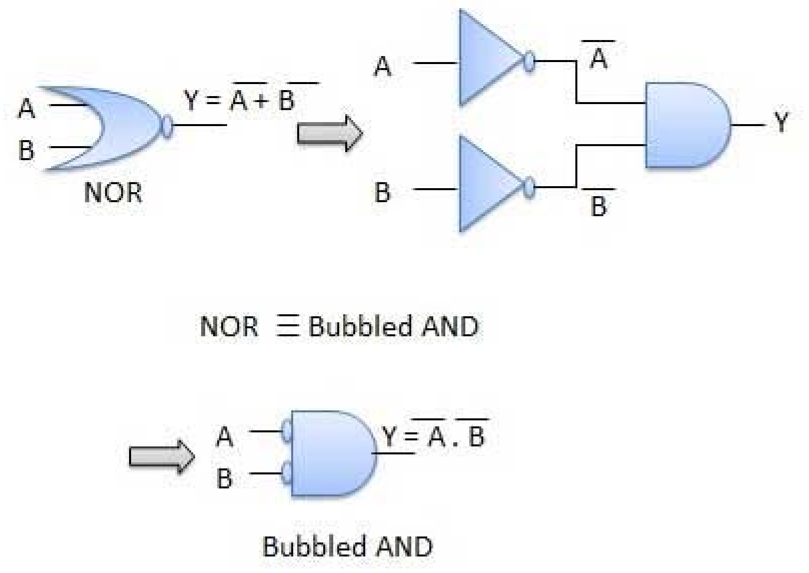
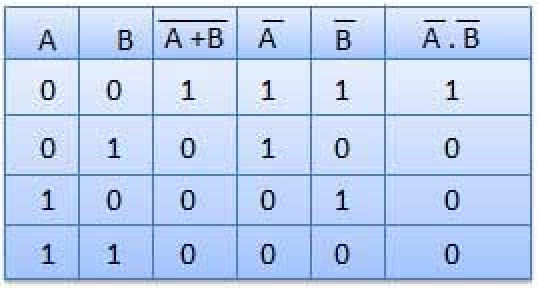


Table showing verification of the De Morgan's second theorem −



A	B	$\overline{A+B}$	$\bar{A}$	$\bar{B}$	$\bar{A}.\bar{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Table: verification of the De Morgan's

Since the values in the $\overline{A+B}$ and $\bar{A}.\bar{B}$ are the same for all input combination .We can conclude that the two expressions are logically equivalent. Hence $\overline{A+B} = \bar{A}.\bar{B}$

Summary

- In logic gates 'OR' operator used for logical addition is represented by the symbol '+' ;
- AND operator used for logical multiplication is represented by the symbol '.'
- NOT operator used for complement is represented by the symbol " - ".
- NOR operator is the combine form of OR and NOT Gates.

- NAND operator is the combine form of AND and NOT Gates.
- XOR operator is the exclusive Gate and represented by symbol $\oplus$.
- XNOR operator is combine form of XOR and NOT Gates.
- NAND and NOR Gates are known as Universal Gates.

Self-Evaluation

Very short question

what will be the output of NOT gate when input is 1 ?

The output of an AND gate with three inputs, A, B, and C, is HIGH when ________.

A. A = 1, B = 1, C = 0

B. A = 0, B = 0, C = 0

C. A = 1, B = 1, C = 1

D. A = 1, B = 0, C = 1

1. If a 3-input NOR gate has eight input possibilities, how many of those possibilities will result in a HIGH output?

A. 1

B. 2

C. 7

D. 8

2. If a signal passing through a gate is inhibited by sending a LOW into one of the inputs, and the output is HIGH, the gate is a(n):

A. AND

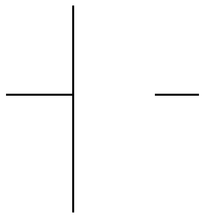
B. NAND

C. NOR

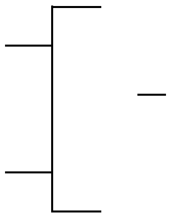
D. OR

3. This question is about NOT and AND logic gates.

   Complete the truth tables for the two gates.



input	output
P	X
0	
1	



inputs		output
P	Q	X
0	0	
0	1	
1	0	
1	1	

Short questions

1. Define logic gates .
2. What is truth table?
3. Draw the logical diagram of XOR Gate.
4. Construct the truth table of NAND Gate.
5. List the Universal Gates.
6. State De Morgan's first theorem.
7. State De Morgan's second theorem.

Long Questions

1. Explain OR Gate with its truth table.
2. Describe the AND Gate with its electrical diagram.
3. Explain the statement "NOT Gate is inverter".
4. Define Truth table, operators and operands with examples.
5. Construct truth table, logic diagram, Venn diagram and use of AND, OR and NOT gates.
6. State and prove the De-Morgan's theorems.

Web Resources

There are a number of resources available on the Internet and the Websites that support this book and help readers keep up with developments in this field. https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm

For students, this Website includes a list of relevant links, organized by chapter, and an extra list for the book. For instructors, this Website provides links to course pages by professors teaching from this book and one can get pdf file of the course.

References

Computer Fundeamentals- sixth Edition
Pradeep Kr. Sinha
Priti Sinha
http://www.tutorialspoint.com
http://www.allaboutcircuits.com/textbook/digital/chpt-3/digital-signals-gates/

Glossary

- Digital =It is a signal or data expressed as series of the digits 0 and 1, typically represented by values of a physical quantity such as voltage or magnetic polarization.
- Boolean Algebra = A division of mathematics which deals with operations on logical values. Boolean algebra traces its origins to an 1854 book by mathematician George Boole.
- Digital Logic= It is the representation of signals and sequences of a digital circuit through numbers
- electronic circuit= It is a complete course of conductors through which current can travel. Circuits provide a path for current to flow.

Practical on Digital Logic gates

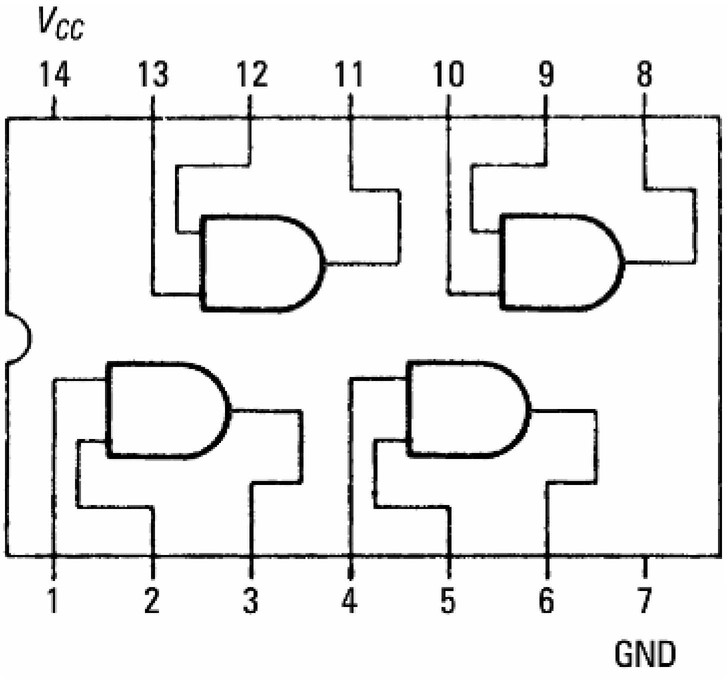
Verification and interpretation of truth tables for AND, OR, NOT, NAND, NOR Exclusive OR (EX-OR), Exclusive NOR (EX-NOR) Gates.

Apparatus: Logic trainer kit, logic gates / ICs, wires.

Theory: Logic gates are electronic circuits which perform logical functions on one or more inputs to produce one output. There are seven logic gates. When all the input combinations of a logic gate are written in a series and their corresponding outputs written along them, then this input/ output combination is called Truth Table. Various gates and their working is explained here.

AND Gate

AND gate produces an output as 1, when all its inputs are 1; otherwise the output is 0. This gate can have minimum 2 inputs but output is always one. Its output is 0 when any input is 0.

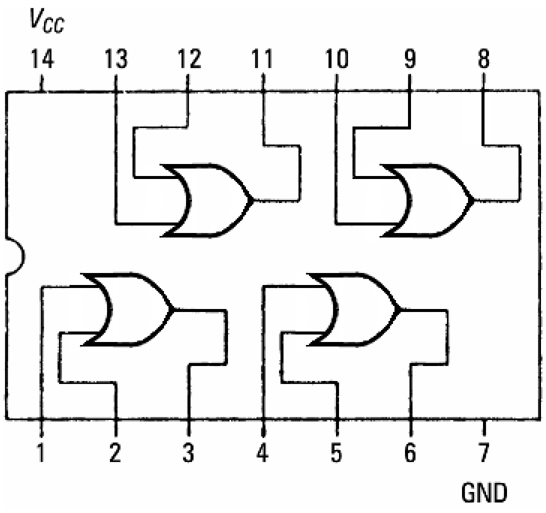


Y=A.B

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

IC 7408

OR Gate
OR gate produces an output as 1, when any or all its inputs are 1; otherwise the output is 0. This gate can have minimum 2 inputs but output is always one. Its output is 0 when all input are 0.



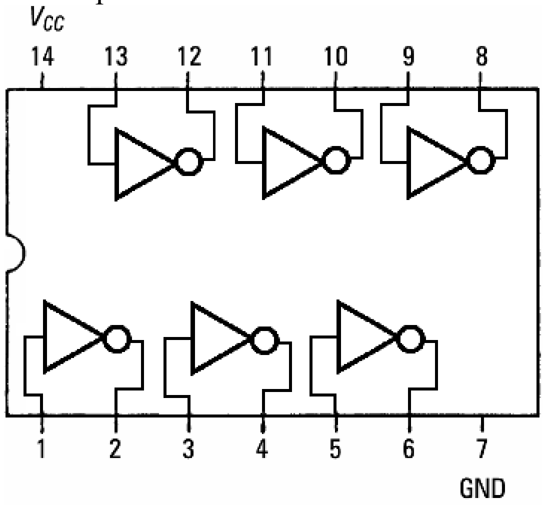
$Y=A+B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

IC 7432

NOT Gate

NOT gate produces the complement of its input. This gate is also called an INVERTER. It always has one input and one output. Its output is 0 when input is 1 and output is 1 when input is 0.



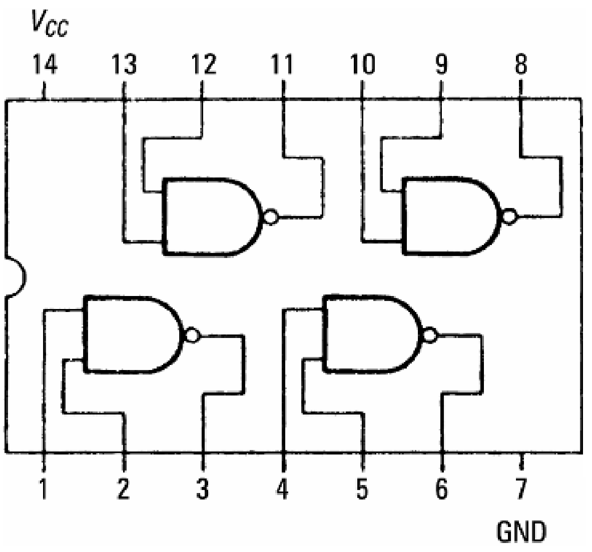
$Y=\overline{A}$

A	Y
0	1
1	0

IC 7404

NAND Gate

NAND gate is actually a series of AND gate with NOT gate. If we connect the output of an AND gate to the input of a NOT gate, this combination will work as NOT-AND or NAND gate. Its output is 1 when any or all inputs are 0, otherwise output is 1.



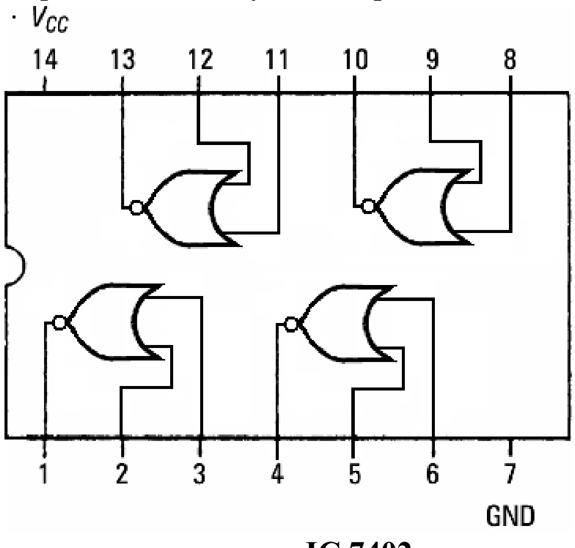
$Y=\overline{A.B}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

IC 7400

NOR Gate

NOR gate is actually a series of OR gate with NOT gate. If we connect the output of an OR gate to the input of a NOT gate, this combination will work as NOT-OR or NOR gate. Its output is 0 when any or all inputs are 1, otherwise output is 1.

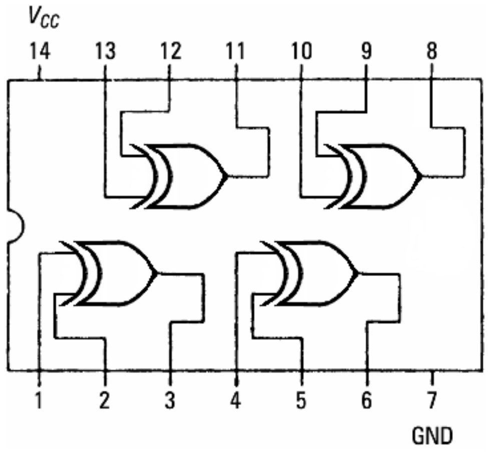


$Y=\overline{A+B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

IC 7402

Exclusive OR (X-OR) Gate
X-OR gate produces an output as 1, when number of 1's at its inputs is odd, otherwise output is 0. It has two inputs and one output.

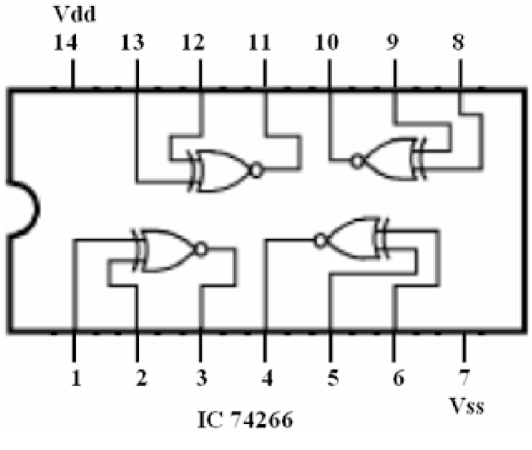


Y=A⊕B

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

IC 7486

Exclusive NOR (X-NOR) Gate
X-NOR gate produces an output as 1, when number of 1's at its inputs is not odd, otherwise output is 0. It has two inputs and one output.



IC 74266

Y = A Θ B

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Procedure:

1. Connect the trainer kit to ac power supply.
2. Connect the inputs of any one logic gate to the logic sources and its output to the logic indicator.
3. Apply various input combinations and observe output for each one.
4. Verify the truth table for each input/ output combination.
5. Repeat the process for all other logic gates.
6. Switch off the ac power supply.

UNIT 4

Boolean Algebra and Karnaugh Map

Learning Outcomes

After completion of this unit students will be able to

- Define boolean expressions and their simplification.
- Define Karnaugh and establish the correspondence between Karnaugh maps and truth tables and logical expressions.
- Demonstrate how to use Karnaugh maps to derive minimal sumof- products and product-of-sums expressions.
- Introduce the concept of "don't care" entries

Introduction

Boolean Algebra, which was invented by George Boole in 1854 is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as Binary Algebra or logical Algebra. The Karnaugh map provides a simple and straight forward method of minimizing Boolean expressions. With the Karnaugh map Boolean expressions having up to four and even six variables can be simplified.

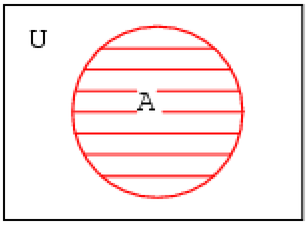
Venn Diagram

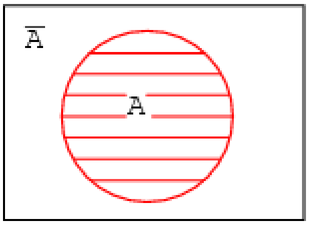
A Venn diagram is a representation of a Boolean operation using shaded overlapping regions. There is one region for each variable, all circular in the examples here. The interior and exterior of region x corresponds respectively to the values 1 (true) and 0 (false) for variable x. The shading indicates the value of the operation for each combination of regions, with dark denoting 1 and light 0. We will adopt the terms OR and AND instead of union and intersection since that is the terminology used in digital electronics.

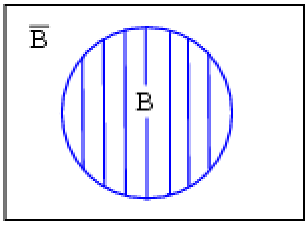
Boolean Relationship on Venn Diagram

The Venn diagram bridges the Boolean algebra to the Karnaugh Map. We will relate what you already know about Boolean algebra to Venn diagrams, then transition to Karnaugh maps.

A set is a collection of objects out of a universe as shown below. The members of the set are the objects contained within the set. The members of the set usually have something in common; though, this is not a requirement. Out of the universe of real numbers, for example, the set of all positive integers {1,2,3…} is a set. The set {3,4,5} is an example of a smaller set, or subset of the set of all positive integers. Another example is the set of all males out of the universe of college students. Can you think of some more examples of sets?

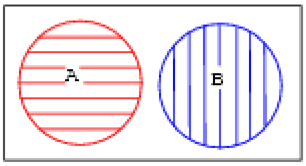


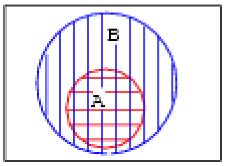


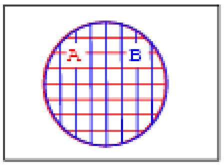


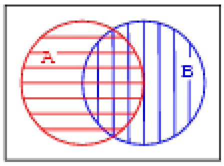
Above left, we have a Venn diagram showing the set A in the circle within the universe U, the rectangular area. If everything inside the circle is A, anything outside of the circle is not A. Thus, above center, we label the rectangular area outside of the circle A as A-not instead of U. We show B and B-not in a similar manner.

What happens if both A and B are contained within the same universe? We show four possibilities.

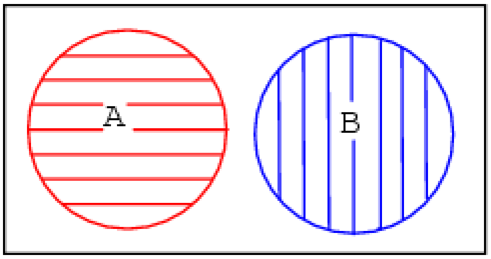








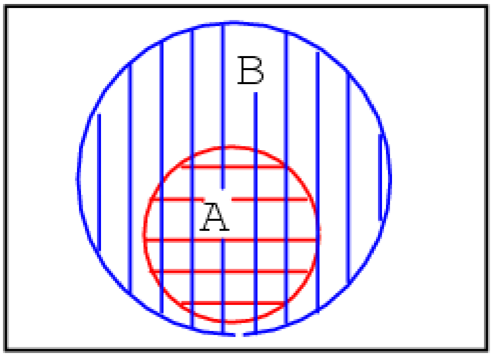
Let's take a closer look at each of the four possibilities as shown above.



The first example shows that set A and set B have nothing in common according to the Venn diagram. There is no overlap between the A and B circular hatched regions. For example, suppose that sets A and B contain the following members:

set A = {1,2,3,4}
set B = {5,6,7,8}

None of the members of set A are contained within set B, nor are any of the members of B contained within A. Thus, there is no overlap of the circles.

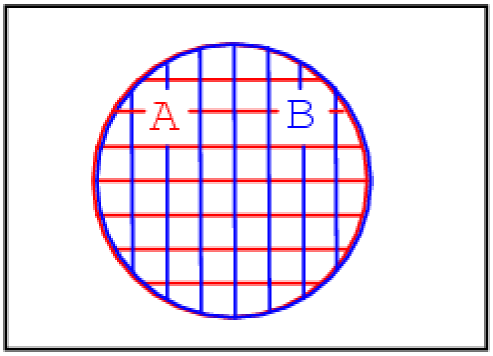


In the second example in the above Venn diagram, set A is totally contained within set B How can we explain this situation? Suppose that sets A and B contain the following members:

set A = {1,2} set B = {1,2,3,4,5,6,7,8}

All members of set A are also members of set B. Therefore, set A is a subset of Set B. Since all members of set A are members of set B, set A is drawn fully within the boundary of set B.

There is a fifth case, not shown, with the four examples. Hint: it is similar to the last (fourth) example. Draw a Venn diagram for this fifth case.



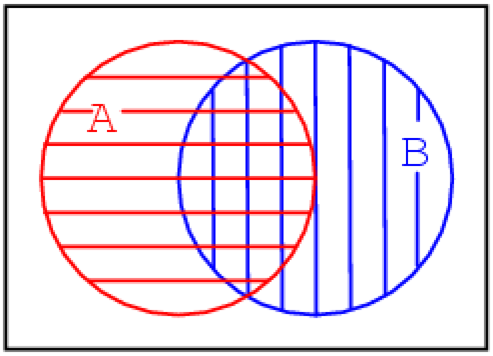
The third example above shows perfect overlap between set A and set B. It looks like both sets contain the same identical members. Suppose that sets A and B contain the following:

set A = {1,2,3,4} set B = {1,2,3,4}

Therefore,

Set A = Set B

Sets A and B are equal because they both have the same identical members. The A and B regions within the corresponding Venn diagram above overlap completely. If there is any doubt about what the above patterns represent, refer to any figure above or below to be sure of what the circular regions looked like before they were overlapped.



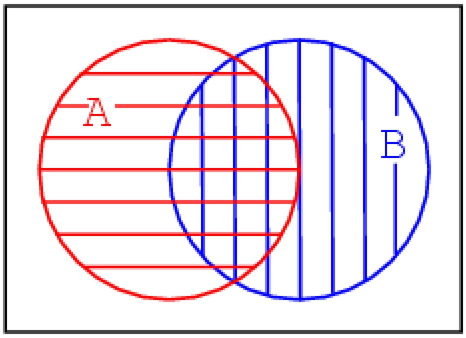
The fourth example above shows that there is something in common between set A and set B in the overlapping region. For example, we arbitrarily select the following sets to illustrate our point:

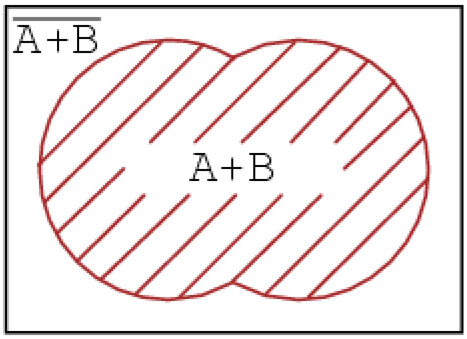
set A = {1,2,3,4}

set B = {3,4,5,6}

Set A and Set B both have the elements 3 and 4 in common. These elements are the reason for the overlap in the center common to A and B. We need to take a closer look at this situation.

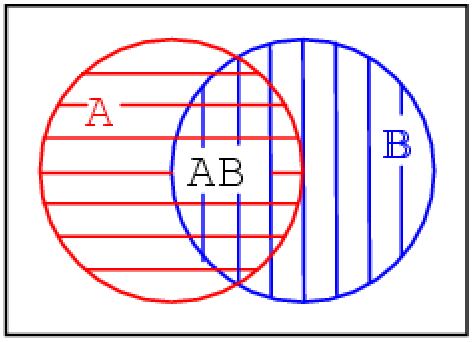
The fourth example has A partially overlapping B. Though, we will first look at the whole of all hatched area below, then later only the overlapping region. Let's assign some Boolean expressions to the regions above as shown below. Below left there is a red horizontal hatched area for A. There is a blue vertical hatched area for B.

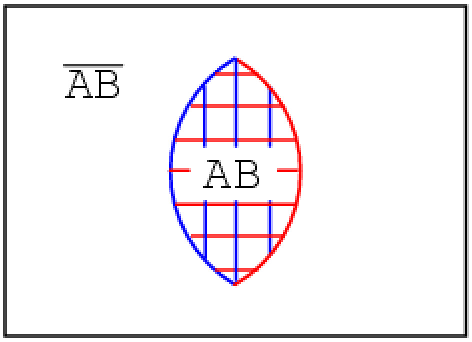




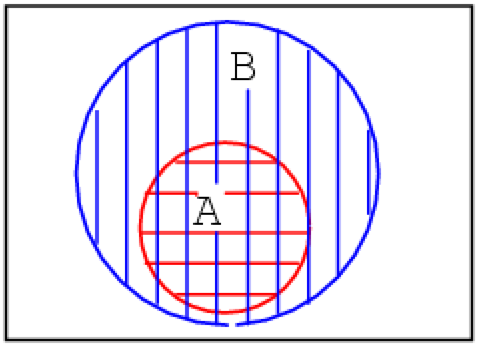
If we look at the whole area of both, regardless of the hatch style, the sum total of all hatched areas, we get the illustration above right which corresponds to the inclusive OR function of A, B. The Boolean expression is A+B. This is shown by the 45o hatched area. Anything outside of the hatched area corresponds to (A+B)-not as shown above. Let's move on to next part of the fourth example.

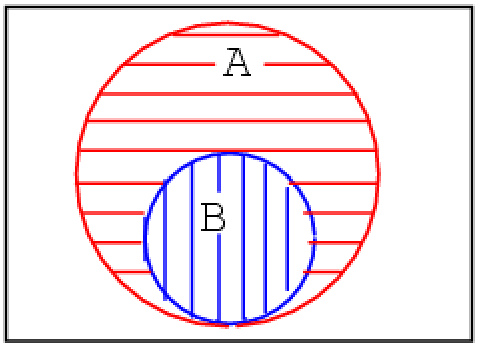
The other way of looking at a Venn diagram with overlapping circles is to look at just at the part common to both A and B, the double hatched area below left. The Boolean expression for this common area corresponding to the AND function is AB as shown below right. Note that everything outside of double hatched AB is AB-not.





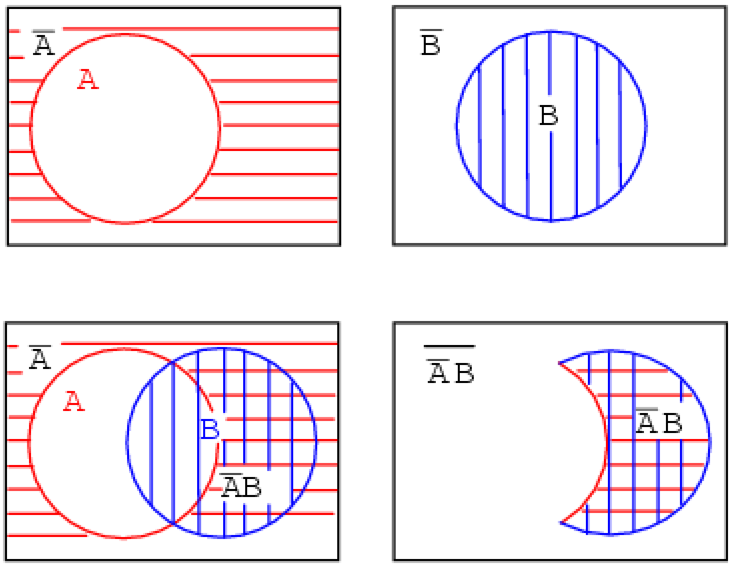
Note that some of the members of **A**, above, are members of **(AB)'**. Some of the members of **B** are members of **(AB)'**. But, none of the members of **(AB)'** are within the doubly hatched area **AB**.





We have repeated the second example above left. Your fifth example, which you previously sketched, is provided above right for comparison. Later we will find the occasional element, or group of elements, totally contained within another group in a Karnaugh Map.

Next, we show the development of a Boolean expression involving a complemented variable below.



Simplification

- Canonical Forms

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

- The sum of minterms (SOM) OR Sum of Product (SOP) form
- The product of maxterms (POM) OR Product of Sum (POS) form

(Note : the product term in a canonical SOP expression is called a minterm)

A Boolean expression consisting entirely either of minterm or maxterm is called canonical expression.

Example :

If we have two variables X and Y then,

Following is a canonical expression consisting of minterms XY + X'Y' and

Following is a canonical expression consisting of maxterm (X+Y) . (X' + Y')

Sum of Products (SOP)

A boolean expression consisting purely of Minterms (product terms) is said to be in canonical sum of products form.

Example:

suppose, we have a boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, output of F is true i.e., F = 1 when any one of the input is true or 1. Now we draw the truth table for F.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Now we will create a column for the minterm using the variables A and B. If input is 0 we take the complement of the variable and if input is 1 we take the variable as it is.

A	B	F	Minterm
0	0	0	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB

To get the desired canonical SOP expression we will add the minterms (product terms) for which the output is 1.

F = A’B + AB’ + AB

Converting Sum of Products (SOP) to Shorthand Notation

From the previous example we have

F = A'B + AB' + AB

Now, lets say we want to express the SOP using shorthand notation.

we have F = A'B + AB' + AB

First we need to denote the minterms in shorthand notation.

A'B = (01)2 = m1

AB' = (10)2 = m2

AB = (11)2 = m3

We saw the conversion of SOP to shorthand notation. Lets check the conversion of shorthand notation to SOP.

Converting Shorthand Notation to Sum of Products (SOP)

Suppose, we have a Boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, the minterms are expressed as shorthand notation given below.

F = ∑(1, 2, 3) our task is to get the SOP.

F has two input variables A and B and output of F = 1 for m1, m2 and m3 i.e., 2nd, 3rd and 4th combination.

We have,

F = ∑(1, 2, 3)
= m1 + m2 + m3
= 01 + 10 + 11

To convert from shorthand notation to SOP we follow the given rules. If the variable is 1 then it is taken "as is" and if the variable is 0 then we take its "complement".

F = ∑(1, 2, 3)
= A'B + AB' + AB

And we have the required SOP.

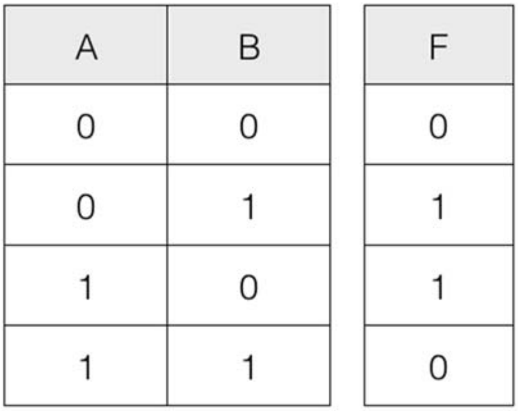
Product of Sums (POS)

A boolean expression consisting purely of Maxterms (sum terms) is said to be in canonical product of sums form.

Example

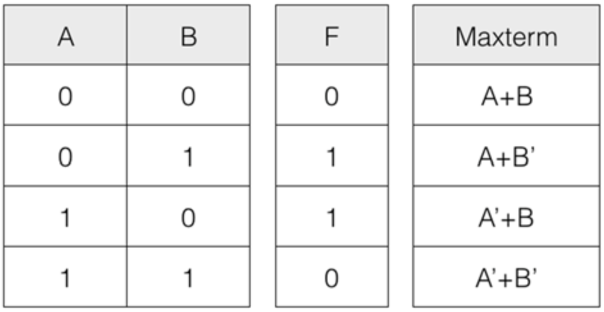
Suppose, we have a boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, output of F is true i.e., F = 1 when only one of the input is true or 1.

Now we draw the truth table for F.



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Now we will create a column for the maxterm using the variables A and B. If input is 1, we take the complement of the variable and if input is 0, we take the variable as is.



A	B	F	Maxterm
0	0	0	A+B
0	1	1	A+B'
1	0	1	A'+B
1	1	0	A'+B'

To get the desired canonical POS expression we will multiply the maxterms (sum terms) for which the output is 0.

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

Converting Product of Sums (POS) to Shorthand Notation

From the previous example we have

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

Now, lets say we want to express the POS using shorthand notation.

we have F = (A+B) . (A'+B')

First we need to denote the maxterms in shorthand notation.

A+B = $(00)_2$ = M_0

A'+B' = $(11)_2$ = M_3

Now we express F using shorthand notation.

F = M_0 . M_3

This can also be written as F = ∏(0, 3)

We saw the conversion of POS to shorthand notation. Lets check the conversion of shorthand notation to POS.

Converting Shorthand Notation to Product of Sums (POS)

Lets say, we have a boolean function F defined on two variables A and B so, A and B are the inputs for F and lets say, the maxterm are expressed as shorthand notation given below.

F = ∏(1, 2, 3)

Our task is to get the POS.

F has two input variables A and B and output of F = 0 for M1, M2 and M3 i.e., 2nd, 3rd and 4th combination.

we have, F = ∏(1, 2, 3)

= M1 . M2 . M3

= 01 . 10 . 11

To convert from shorthand notation to POS we follow the given rules. If the variable is 0 then it is taken as is and if the variable is 1 then we take its complement.

we have, F = ∏(1, 2, 3)

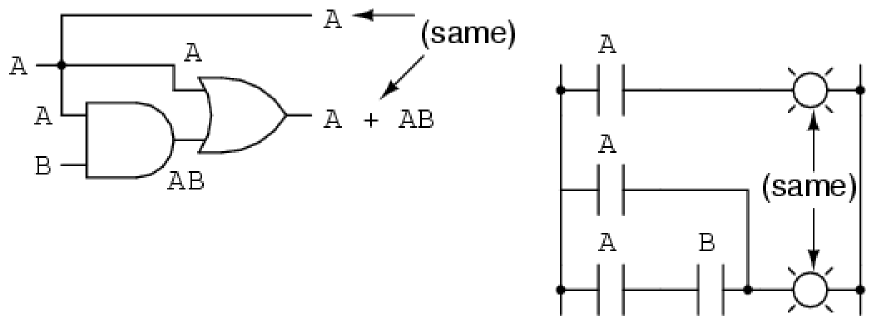
= (A+B') . (A'+B) . (A'+B') And we have the required POS.

Algebraic Simplification

Boolean algebra finds its most practical use in the simplification of logic circuits. If we translate a logic circuit's function into symbolic (Boolean) form, and apply certain algebraic rules to the resulting equation to reduce the number of terms and/or arithmetic operations, the simplified equation may be translated back into circuit form for a logic circuit performing the same function with fewer components. If equivalent function may be achieved with fewer components, the result will be increased reliability and decreased cost of manufacture.

To this end, there are several rules of Boolean algebra presented in this section for use in reducing expressions to their simplest forms. The identities and properties already reviewed in this chapter are very useful in Boolean simplification, and for the most part bear similarity to many identities and properties of "normal" algebra. However, the rules shown in this section are all unique to Boolean mathematics.

$$A + AB = A$$



This rule may be proven symbolically by factoring an "A" out of the two terms, then applying the rules of A + 1 = 1 and 1A = A to achieve the final result:

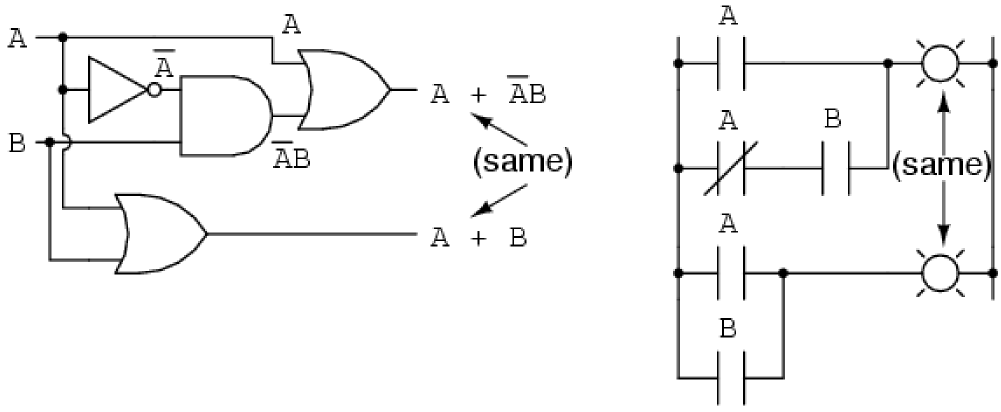
$$A + AB \xrightarrow{\text{Factoring } A \text{ out of both terms}} A(1 + B) \xrightarrow{\text{Applying identity } A + 1 = 1} A(1) \xrightarrow{\text{Applying identity } 1A = A} A$$

Please note how the rule A + 1 = 1 was used to reduce the (B + 1) term to 1. When a rule like "A + 1 = 1" is expressed using the letter "A", it doesn't mean it only applies to expressions containing "A". What the "A" stands for in a rule like A + 1 = 1 is any Boolean variable or collection of variables.

.For instance, the Boolean expression ABC + 1 also reduces to 1 by means of the "A + 1 = 1" identity. In this case, we recognize that the "A" term in the identity's standard form can represent the entire "ABC" term in the original expression.

The next rule looks similar to the first one shown in this section, but is actually quite different and requires a more clever proof:

$$\mathbf{A + \overline{A}B = A + B}$$

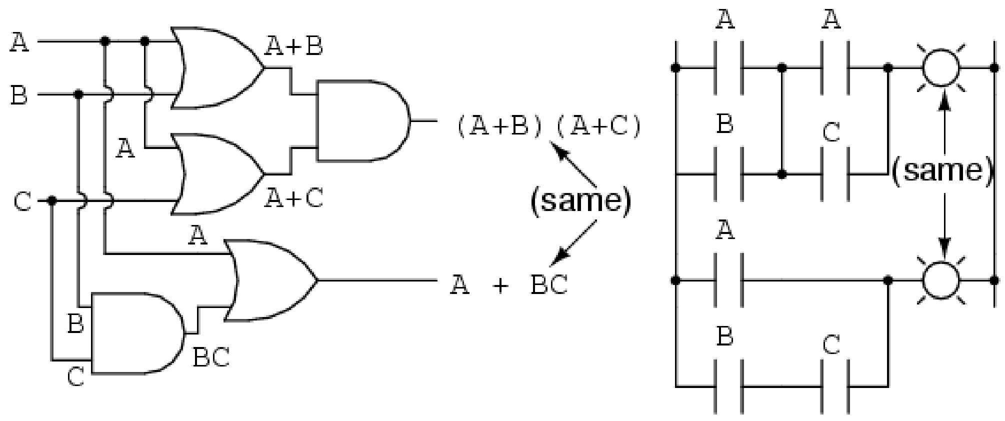


$$\begin{array}{ll} A + \overline{A}B & \\ \downarrow & \text{Applying the previous rule to expand } \mathbf{A} \text{ term} \\ & \mathbf{A + AB = A} \\ A + AB + \overline{A}B & \\ \downarrow & \text{Factoring } \mathbf{B} \text{ out of 2}^{nd} \text{ and 3}^{rd} \text{ terms} \\ A + B(A + \overline{A}) & \\ \downarrow & \text{Applying identity } \mathbf{A + \overline{A} = 1} \\ A + B(1) & \\ \downarrow & \text{Applying identity } \mathbf{1A = A} \\ A + B & \end{array}$$

Note how the last rule (A + AB = A) is used to "un-simplify" the first "A" term in the expression, changing the "A" into an "A + AB". While this may seem like a backward step, it certainly helped to reduce the expression to something simpler! Sometimes in mathematics we must take "backward" steps to achieve the most elegant solution. Knowing when to take such a step and when not to is part of the art-form of algebra, just as a victory in a game of chess almost always requires calculated sacrifices.

Another rule involves the simplification of a product-of-sums expression:

(A + B) (A + C) = A + BC



And, the corresponding proof:

(A + B)(A + C)

↓ Distributing terms

AA + AC + AB + BC

↓ Applying identity **AA = A**

A + AC + AB + BC

↓ Applying rule **A + AB = A** to the A + AC term

A + AB + BC

↓ Applying rule **A + AB = A** to the A + AB term

A + BC

To summarize, here are the three new rules of Boolean simplification expounded in this section:

Useful Boolean rules for simplification

$$A + AB = A$$

$$A + \overline{A}B = A + B$$

$$(A + B)(A + C) = A + BC$$

Here are some examples of Boolean algebra simplifications. Each line gives a form of the expression, and the rule or rules used to derive it from the previous one. Generally, there are several ways to reach the result. Here is the list of simplification rules.

- Simplify: C + BC:

Expression	Rule(s) Used
C + BC	Original Expression
C + (B + C)	DeMorgan's Law
(C + C) + B	Commutative, Associative Laws
T + B	Complement Law
T	Identity Law

- Simplify: AB(A + B)(B + B):

Expression	Rule(s) Used
AB(A + B)(B + B)	Original Expression
AB(A + B)	Complement law, Identity law
(A + B)(A + B)	DeMorgan's Law
A	Complement, Identity
A + BB	Distributive law. This step uses the fact that or distributes over and. It can look a bit strange since addition does not distribute over multiplication.

- Simplify: (A + C)(AD + AD) + AC + C:

Expression	Rule(s) Used

(A + C)(AD + AD) + AC + C	Original Expression
(A + C)A(D + D) + AC + C	Distributive
(A + C)A + AC + C	Complement, Identity
A((A + C) + C) + C	Commutative, Distributive
A(A + C) + C	Associative, Idempotent
AA + AC + C	Distributive
A + (A + T)C	Idempotent, Identity, Distributive
A + C	Identity, twice

- You can also use distribution of or over and starting from A(A+C)+C to reach the same result by another route.
- Simplify: A(A + B) + (B + AA)(A + B):

Expression	Rule(s) Used
A(A + B) + (B + AA)(A + B)	Original Expression
AA + AB + (B + A)A + (B + A)B	Idempotent (AA to A), then Distributive, used twice
AB + (B + A)A + (B + A)B	Complement, then Identity. (Strictly speaking, we also used the Commutative Law for each of these applications.)
AB + BA + AA + BB + AB	Distributive, two places
AB + BA + A + AB	Idempotent (for the A's), then Complement and Identity to remove BB
AB + AB + AT + AB	Commutative, Identity; setting up for the next step
AB + A(B + T + B)	Distributive
AB + A	Identity, twice (depending how you count it).
A + AB	Commutative
(A + A)(A + B)	Distributive
A + B	Complement, Identity

The Concept of K-Maps

The Karnaugh map (KM or K-map) is a method of simplifying Boolean algebra expressions. Maurice Karnaugh introduced it in 1953 as a refinement of Edward Veitch's 1952 Veitch chart. A Karnaugh map (K-map) is a pictorial method used to minimize Boolean expressions without having to use Boolean algebra theorems and equation manipulations. A K-map can be thought of as a special version of a truth table .

The Karnaugh map reduces the need for extensive calculations by taking advantage of humans' pattern-recognition capability. It also permits the rapid identification and elimination of potential race conditions.

This is all well and good for something simple like the 2-input multiplexer. But using Boolean algebra to reduce circuits can be difficult. Thus, we would like a method or procedure that is easier. Therefore, we use Karnaugh maps.The Karnaugh map (or K-map) is a visual way of detecting redundancy in the SSoP.

The K-map can be easily used for circuits with 2, 3, or 4 inputs.

It consists of an array of cells, each representing a possible combination of inputs.

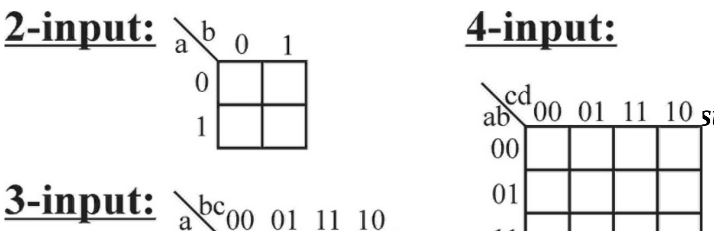
- The cells are arranged so that each cell's input combination differs from adjacent cells by only a single bit.
- This is called Gray code ordering – it ensures that physical neighbours in the array are logical neighbours as well. (In other words, neighbouring bit patterns are nearly the same, differing by only 1 bit).

Consider the following arrangements of cells:

2-input

a'. b' 00	a'. b 01
a .b' 10	a . b 11

The cells are arranged as above, but we write them empty, like this:



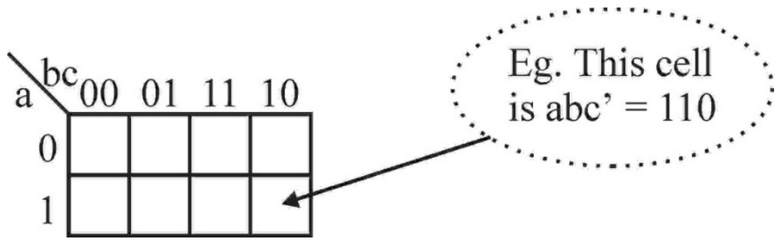
The no. of boxes in the table = 2n, where n = no. of variables.

Note: The numbers are not in binary order, but are arranged so that only a single bit changes between neighbours.

This one-bit change applies at the edges, too.

So cells in the same row on the left and right edges of the array also only differ by one bit.

Note: The value of a particular cell is found by combining the numbers at the edges of the row and column.



Also, in general, it is easier to order the inputs to a K-map so that they can be read like a binary number.

(Show example.)

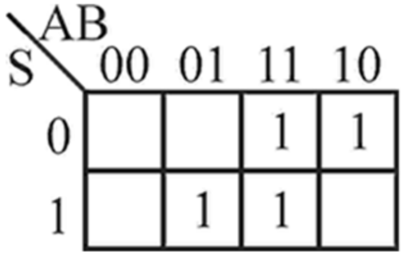
So, we have this grid. What do we do with it?

- We put 1's in all the cells that represent minterms in the SSoP .
  (In other words, we find the 1's in the truth table output, and put 1's in the cells corresponding to the same inputs.)

Let's do this in relation to the 2-input multiplexer example:

S	A	B	Y

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



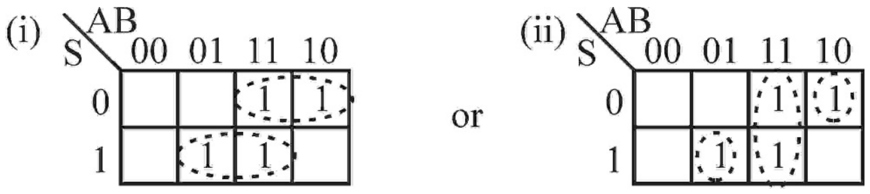
If there are two neighboring 1's in the grid, it means that the input bit change between the two cells has no effect on the output, and thus there is redundancy. This leads to a basic strategy.

Basic Strategy:

Group adjacent 1's together in square or rectangular groups of 2, 4, 8, or 16, such that the total number of groups and isolated 1's is minimized, while using as large groups as possible. Groups may overlap, so that a particular cell may be included in more than one group.

(Recall that adjacency wraps around edges of grid.)

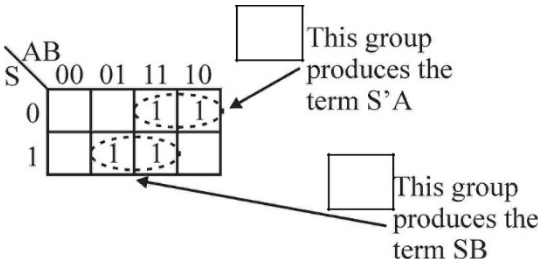
Applying this to the multiplexer example:



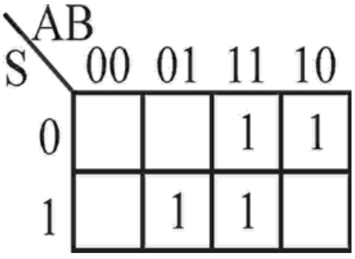
Option (i) is best since only 2 groups vs. 3

So, considering the best option above (i), notice the following:

- B changes but the output doesn't, so B is redundant in this group (See comment 1, below).
- A changes but the output doesn't, so A is redundant in this group (See comment 2, below).


AB
S
00 01 11 10
0
1
1 1
1 1
This group produces the term S'A
This group produces the term SB

(1)



(2)

So, we write out Boolean expressions for each group, leaving out the redundant elements. That is, for each group, we write out the inputs that don't change. The multiplexer example, with two groups, gives us two terms, Y = S.B + S'.A
which is the same as what we achieved through using Boolean algebra to reduce the circuit. So, we can summarize this process into a basic set of rules:

Rules for K-Maps

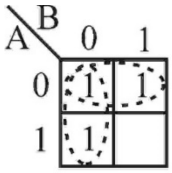
1) Each cell with a 1 must be included in at least one group.
2) Try to form the largest possible groups.
3) Try to end up with as few groups as possible.
4) Groups may be in sizes that are powers of 2: 2 0 = 1, 21 = 2, 22 = 4, 23 = 8, 24 = 16, ...
5) Groups may be square or rectangular only (including wrap-around at the grid edges). No diagonals or zigzags can be used to form a group.
6) The larger a group is, the more redundant inputs there are:
   - A group of 1 has no redundant inputs.
   - A group of 2 has 1 redundant input.
   - A group of 4 has 2 redundant inputs.
   - A group of 8 has 3 redundant inputs.
   - A group of 16 has 4 redundant inputs.

The following simple examples illustrate rule 6 above.

Examples

2-input Example

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



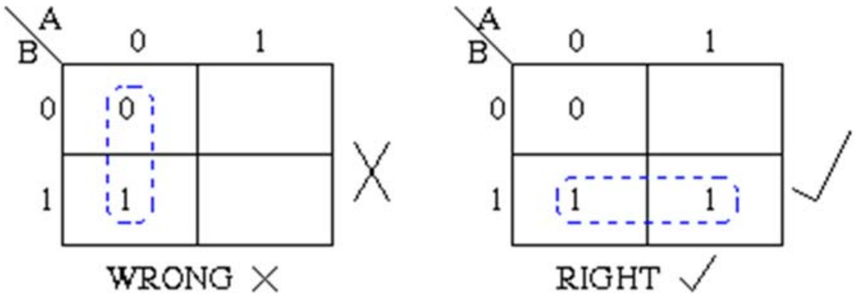
Y = B' + A'
(This is NAND)

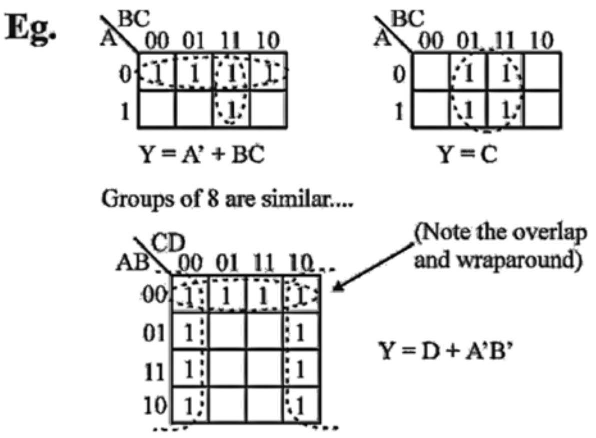
Direct from truth table: Y = A'B' + A'B + AB'

Karnaugh Maps - Rules of Simplification

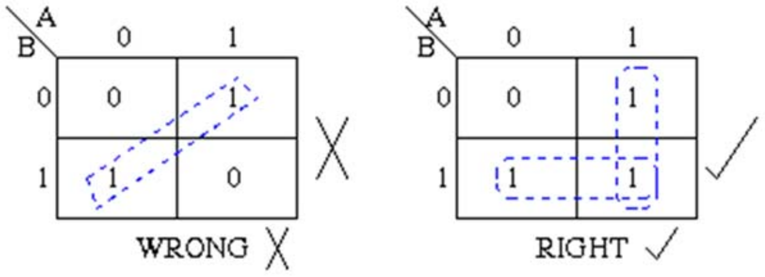
The Karnaugh map uses the following rules for the simplification of expressions by grouping together adjacent cells containing ones

- Groups may not include any cell containing a zero

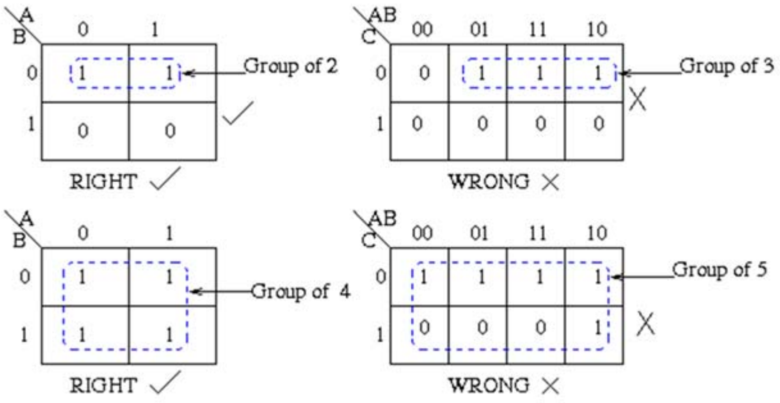



Eg.
Y = A' + BC
Y = C
Groups of 8 are similar....
(Note the overlap and wraparound)
Y = D + A'B'

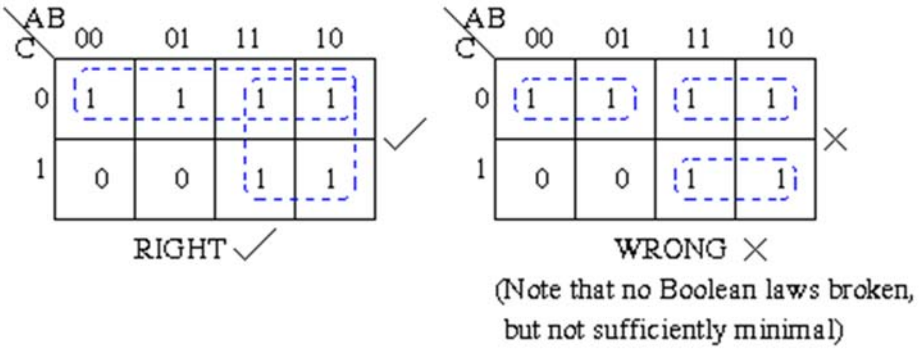
* Groups may be horizontal or vertical, but not diagonal.



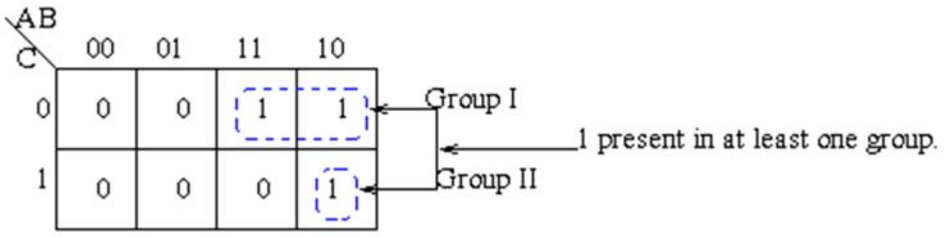
* Groups must contain 1, 2, 4, 8, or in general 2n cells.
  That is if n = 1, a group will contain two 1's since 21 = 2.
  If n = 2, a group will contain four 1's since 22 = 4.



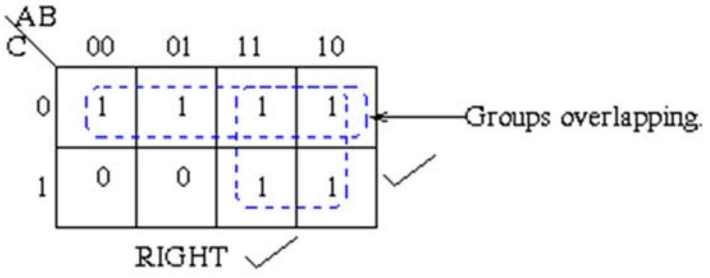
* Each group should be as large as possible.

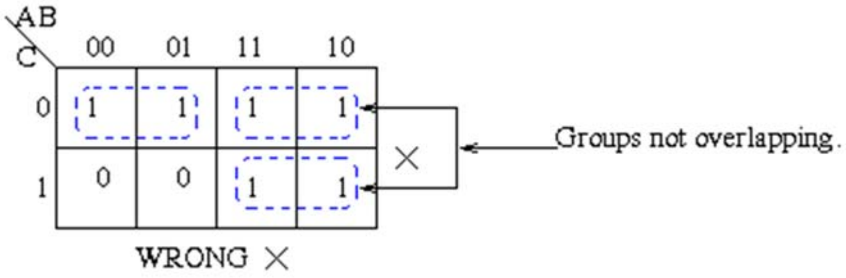

AB
C
00
01
11
10
RIGHT ✓
WRONG ×
(Note that no Boolean laws broken, but not sufficiently minimal)

- Each cell containing a **1** must be in at least one group.

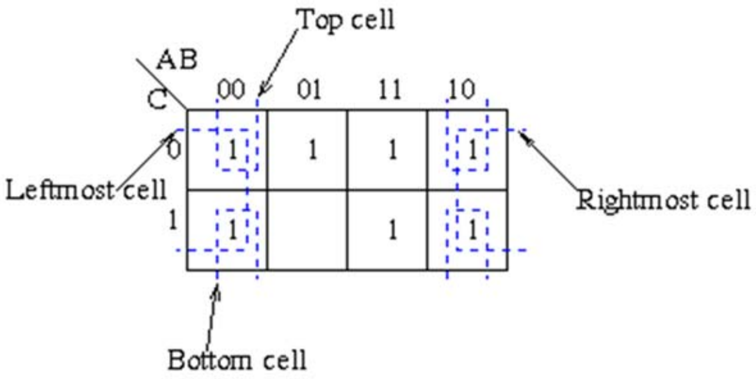


- Groups may overlap.

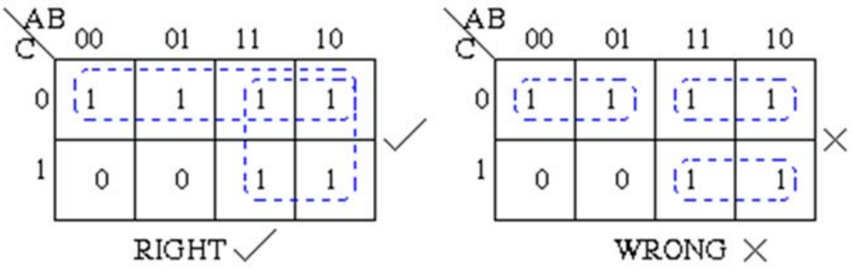




- Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



- There should be as few groups as possible, as long as this does not contradict any of the previous rules.

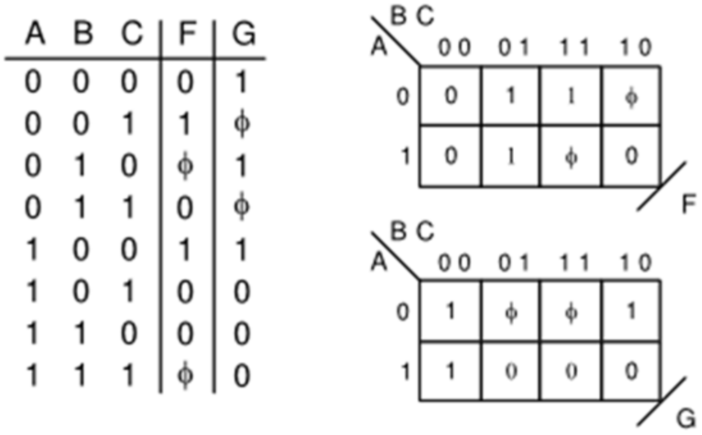


Summary for Rules of Simplification

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 number of cells in each group.
4. Groups should be as large as possible.
5. Everyone must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups possible.

Don't Care Condition

Situations can arise where a circuit has N input signals, but not all 2N combinations of inputs are possible. Or, if all 2N combinations of inputs are possible, some combinations might be irrelevant. For example, consider a television remote control unit that can switch between control of a television, VCR, or DVD. Some remotes might have operational modes where buttons like “fast forward” are physically switched out of the circuit; other remotes may use modes where such buttons are left in the circuit, but their functions are irrelevant. In either case, some combinations of input signals are completely inconsequential to the proper operation of the circuit. It is possible to take advantage of these situations to further minimize logic circuits.



A	B	C	F	G
0	0	0	0	1
0	0	1	1	ϕ
0	1	0	ϕ	1
0	1	1	0	ϕ
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	ϕ	0

F:

A \ BC	00	01	11	10
0	0	1	1	ϕ
1	0	1	ϕ	0

G:

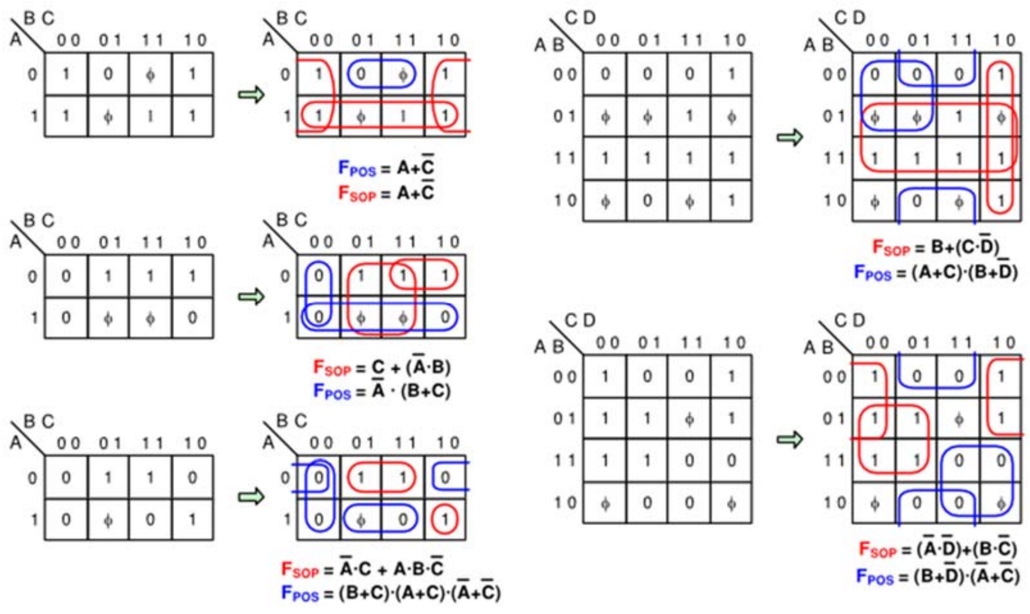
A \ BC	00	01	11	10
0	1	ϕ	ϕ	1
1	1	0	0	0

Figure 1. Truth table and K-Maps with Don't Care.

Input combinations that cannot possibly effect the proper operation of a logic system can be allowed to drive circuit outputs high or low—literally. The designer doesn't care what the circuit response is to these impossible or irrelevant inputs. This information is encoded by using a special “don't care” symbol in truth tables and K-maps to indicate that the signal can be a '1' or a '0' without effecting circuit operation. Some sources use an 'X' to indicate a don't care, but this can be confused with a signal named 'X'. It is perhaps a better practice to use a symbol that is not normally associated with signal names—here, we choose the 'Φ' symbol. The “don't care” can be used on the map to provide simplification of the function.

The truth table above shows two output functions (F and G) for the same three inputs. Both outputs have two rows where the output is a don't care. This same information is also shown in the associated K-maps. Clearly, looping cell 7 as a '1' and cell 2 as a '0' results in a more minimal logic circuit. In this case, both an SOP and POS looping would result in identical circuits.

In the 'G' K-map, the don't cares in cells 1 and 3 can be looped as either a '1' or a '0'. In an SOP looping, both don't cares would be looped as 1's, giving a logic function of G=A¯+B¯·C¯G=A¯+B¯·C¯. In a POS looping, however, cells 1 and 3 would be looped as 0's, giving the logic function G=C¯·(A¯+B¯)G=C¯·(A¯+B¯). A little Boolean algebra reveals these two equations are not algebraically equal. Often, the SOP and POS forms of equations looped from K-maps that contain don't cares are not algebraically equal (although they would perform identically in the circuit). The following examples in Fig. 2 illustrate the use of don't cares in K-maps.



$F_{POS} = A+\overline{C}$
$F_{SOP} = A+\overline{C}$
$F_{SOP} = C + (\overline{A}\cdot B)$
$F_{POS} = \overline{A} \cdot (B+C)$
$F_{SOP} = \overline{A}\cdot C + A\cdot B\cdot\overline{C}$
$F_{POS} = (B+C)\cdot(A+C)\cdot(\overline{A}+\overline{C})$
$F_{SOP} = B+(C\cdot\overline{D})$
$F_{POS} = (A+C)\cdot(B+\overline{D})$
$F_{SOP} = (\overline{A}\cdot\overline{D})+(B\cdot\overline{C})$
$F_{POS} = (B+\overline{D})\cdot(\overline{A}+\overline{C})$

Figure 2. Looping K-Maps with Don't Care.

If all combination of 2N inputs are possible, some combinations may be irrelevant. Some combinations of input signals can be completely inconsequential to the operation of the circuit. It is possible to take advantage of the situations where input combinations are not required for the function of the circuit to further minimize the logic circuit. Information for the irrelevant inputs are noted by the 'Φ' symbol and are known as “Don't Cares”.

Summary

SOP and POS –useful forms of Boolean equations

Design of a comb. Logic circuit

- construct its truth table, (2) convert it to a SOP, (3)
- simplify using Boolean algebra or K mapping, (4)
- SOP and POS –useful forms of Boolean equations
- Design of a comb. Logic circuit
- (1) construct its truth table, (2) convert it to a SOP, (3)
- simplify using Boolean algebra or K mapping, (4)
- Implement
- K map: a graphical method for representing a circuit’s
- truth table and generating a simplified expression
- “Don’t cares” entries in K map can take on values of 1
- or 0. Therefore can be exploited to help simplification

Self Evaluation

Very short question

Q.1 Define Venn diagram.

Q.2 Draw venn diagram of A'

Q.3 List out different canonical forms.

Q.4 What do you mean by sum of product

Q.5 Define minterms.

Q.6 What is maxterms mean ?

Q.7 Complete following boolean expression

A+AB=

A+A'B=

(A+B)(A+C)=

Q.8 Define K-maps in short.

Short Question

Q.1 What is boolean algebra ? Give example
Q.2 Explain Product of sum in detail
Q.3 Prove boolean expression A+AB=A
Q.4 Simplify the expression AB(A+B)(B+B)
Q.5 Explain karnauf Map with table.
Q.6 List out the rules to use in K-maps.
Q.7 Explain Don't care condition with example.

Long question

Q.1 Explain Sum of Product in detail
Q.2 Express the Boolean function F=A+B'.C in sum-of -minterms(products) form.
Q.3 Explain Product of Sum in detail with example
Q.4 Explain Karnaugh Maps rules of simplification with example.

Glossary

- **Boolean Expression:-** a Boolean expression is an expression in a programming language that produces a Boolean value when evaluated, i.e. one of true or false. A Boolean expression may be composed of a combination of the Boolean constants true or false.
- **Gray code:-** a numerical code used in computing in which consecutive integers are represented by binary numbers differing in only one digit.
- **Multiplexer:-** a multiplexer (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. A multiplexer of 2n inputs has n select lines, which are used to select which input line to send to the output.

Resources

- 1 www.wikipedia.com
- Computer Fundeamentals- sixth Edition
- Pradeep Kr. Sinha Priti Sinha
- http://www.tutorialspoint.com
- http://www.allaboutcircuits.com/textbook/digital/chpt-3/digital-signals-gates/

UNIT- 5

Binary Arithmetic Logic

Learning Outcomes

After completion of this unit you will be able to

- To explain/describe the formalism of binary arithmetic logic
- To explain/describe binary adder, half adders, full adders, half subtractors, full subtractors.

Introduction

Binary arithmetic is essential part of all the digital computers and many other digital systems. Binary logic deals with variables that assume discrete values and with operators that assume logical meaning.

While each logical element or condition must always have a logical value of either "0" or "1", we also need to have ways to combine different logical signals or conditions to provide a logical result.

For example, consider the logical statement: "If I move the switch on the wall up, the light will turn on." At first glance, this seems to be a correct statement. However, if we look at a few other factors, we realize that there's more to it than this. In this example, a more complete statement would be: "If I move the switch on the wall up and the light bulb is good and the power is on, the light will turn on."

If we look at these two statements as logical expressions and use logical terminology, we can reduce the first statement to:

Light = Switch

This means nothing more than that the light will follow the action of the switch, so that when the switch is up/on/true/1 the light will also be on/true/1. Conversely, if the switch is down/off/false/0 the light will also be off/false/0.

Looking at the second version of the statement, we have a slightly more complex expression:

Light = Switch and Bulb and Power

When we deal with logical circuits (as in computers), we not only need to deal with logical functions; we also need some special symbols to denote these functions in a logical diagram. There are three fundamental logical operations, from which all other functions, no matter how complex, can be derived. These functions are named and, or, and not. Each of these has a specific symbol and a clearly-defined behaviour.

In binary number system there are only 2 digits 0 and 1, and any number can be represented by these two digits. The arithmetic of binary numbers means the operation of addition, subtraction, multiplication and division.

In this unit you will study about adder, half adder, full adder,binary adder, half subtractor and full subtrator.

Adder

An adder is a digital circuit that performs addition of numbers. In many computers and other kinds of processors adders are used in the arithmetic logic units. They are also utilized in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators, and similar operations.

Half Adder

Before designing a binary adder, let us know some basic rules of binary addition. The most basic binary addition is addition of two single bit binary numbers i.e. addition of two binary digits.

i. 0 + 0 = 0
ii. 0 + 1 = 1
iii. 1 + 0 = 1
iv. 1 + 1 = 10

The binary digits are 0 and 1. Hence, there must be four possible combinations of binary addition of two binary bits. In the above list, first three binary operations result in one bit but fourth one results in two bits. In one bit binary addition, if augend and addend are 1, the sum will have two digits. (Augend is the number to which another is added. Addends is any of the numbers that are added together. Example: In 8 + 3 = 11, the 8 and the 3 are addends.) The higher significant bit (HSB) or Left side bit is called carry and the list significant bit (LSB) or right side bit of the result is called sum bit. The logical circuit that performs this one bit binary addition is called half adder.

Design of Half Adder

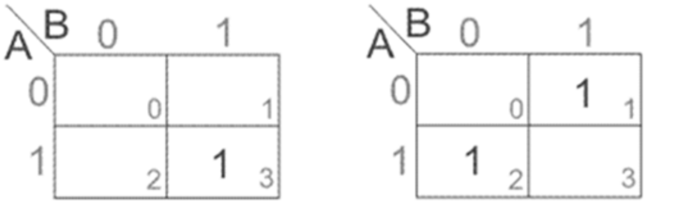
For designing a half adder logic circuit, we first have to draw the truth table for two input variables i.e. the augend and addend bits, two outputs variables carry and sum bits. In first three binary additions, there is no carry hence the carry in these cases are considered as 0.

Inputs		*Outputs*	
Augend(A)	*Addend(B)*	*Carry(C)*	*Sum(S)*
0	*0*	*0*	*0*
0	*1*	*0*	*1*
1	*1*	*0*	*1*
1	1	1	0

Truth Table for Half Adder

K-map for Half Adder

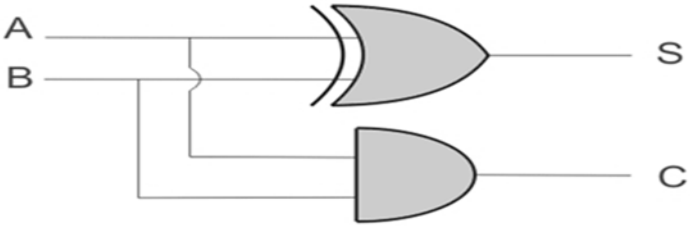
Now from this truth table we can draw K-map for carries and sums separately.



K-map for Carry K-map for Sum

For above K-maps we get, $Carry\ \ C = AB\ and\ Sum\ \ C = A\overline{B} + \overline{A}B.$

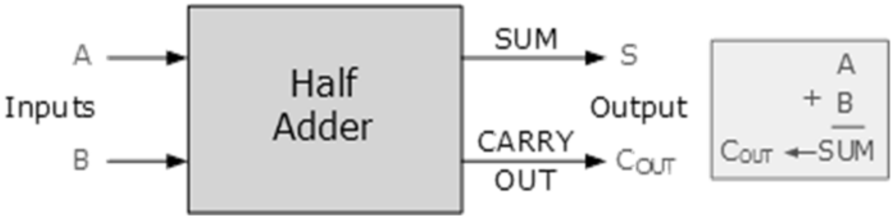
Hence, the logical design of Half Adder would be



Although from truth table it is clearly seen that carry (C) column signifies AND operation and sum (S) column signifies XOR operation between input variables but till we went through K-map as it is general practice to do so for more complex binary logic operations.

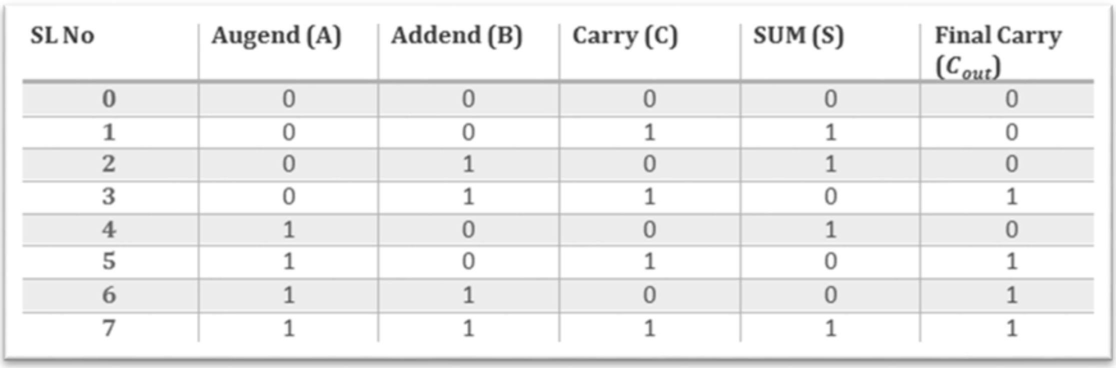
Binary Adder:-The binary adder is one of the basic combinational logic circuits. The outputs of a combinational logic circuit depend on the present input only. In other words, outputs of combinational logic circuit do not depend upon any previously applied inputs. It does not require any memory like component. Binary adder is one of the basic combinational logical circuits as present state of input variables.

Binary Adder Block Diagram



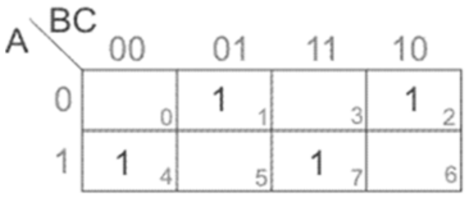
Full Adder

The full adder is a conditional circuit which performs full binary addition. It means it adds two bits and a carry and outputs a sum bit and a carry bit. Any bit of augend can either be 1 or 0 and we can represent it with variable A, similarly any bit of addend we represent with variable B. The carry after addition of same significant bit of augend and addend can be represented by C. Hence truth table for all combinations of A, B and C is as follows,

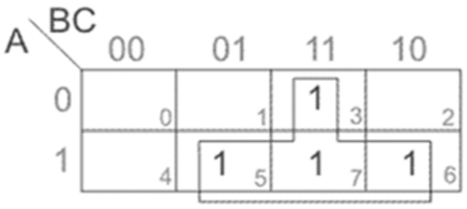


SL No	Augend (A)	Addend (B)	Carry (C)	SUM (S)	Final Carry (C_{out})
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

From the above table, we can draw K-map for sum (s) and final carry (Cout).



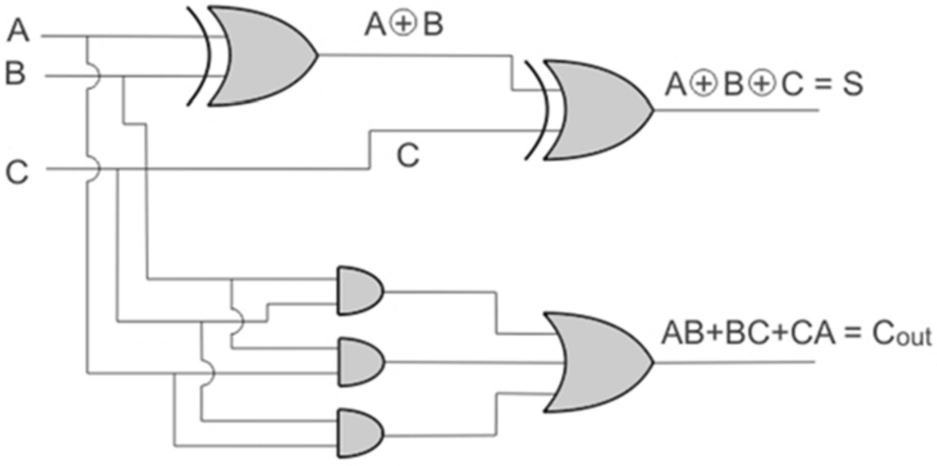
K-map for Sum (S)



K-map for Carry (C_{out})

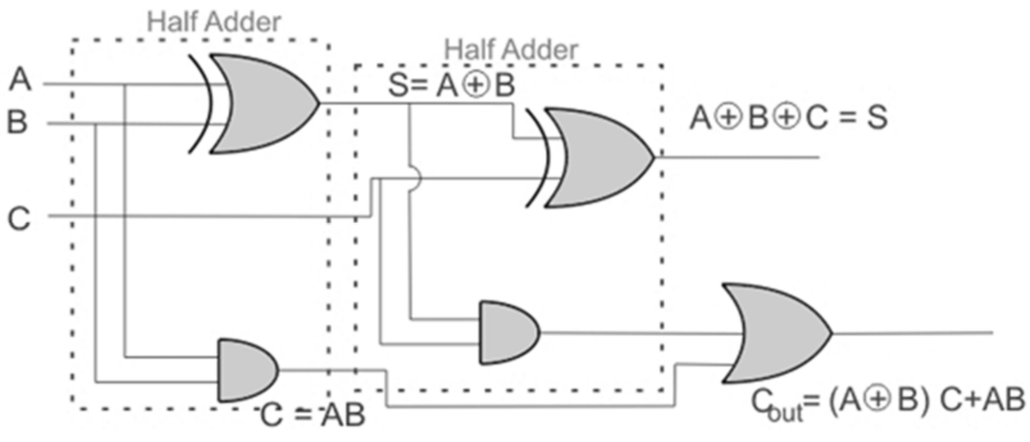
$$
\begin{aligned}
S &= A\overline{B}\overline{C} + \overline{A}\,\overline{B}C + ABC + \overline{A}B\overline{C} \\
&= C\,(AB + \overline{A}\,\overline{B}) + \overline{C}\,(\overline{A}B + A\,\overline{B}) \\
&= C\,(\overline{\overline{A}B + A\,\overline{B}}) + \overline{C}\,(\overline{A}B + A\,\overline{B}) \\
&= C\,(\overline{A \oplus B}) + \overline{C}\,(A \oplus B)\ = A \oplus B \oplus C.
\end{aligned}
$$

Hence, from K-maps,



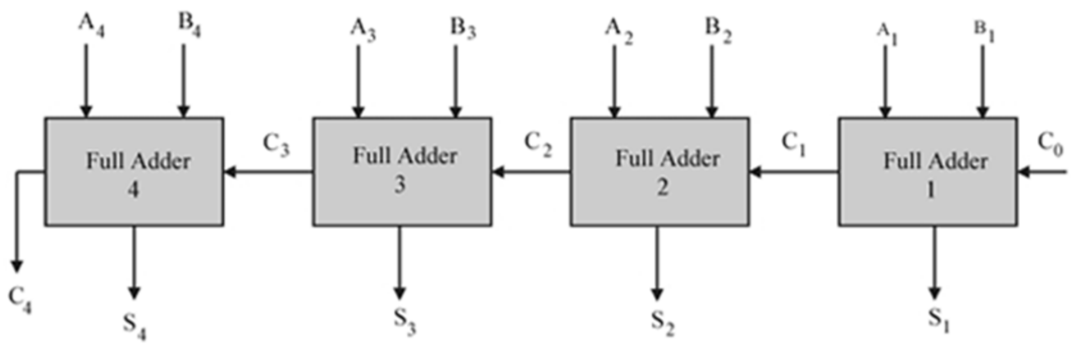
$$
\begin{aligned}
C_{out} &= \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \\
&= (\overline{A}B + A\overline{B})\,C + AB\,(\overline{C} + C) \\
&= (A \oplus B).\,C + AB.
\end{aligned}
$$

Therefore, the equation for sum, carry for this circuit can be given as



Binary Parallel Adder

A full binary adder performs addition of any single bit of one binary number, same significant or same position bit of another binary numbers, the carry comes from result of addition of previous right side bits of both binary numbers. But a single full adder cannot add more than one bits binary number instantly. This can be done only by connecting as many full adders as the number of bits of the binary numbers whose addition is to be performed. This parallel combination of full adders which performs addition of specific bits binary numbers is called binary parallel adder. For adding two 4 bit binary numbers we have to connect 4 full adders to make 4 bit parallel adder. The inter connection of 4 full adder in 4bit parallel adder is shown below:



Let us examine the justification of the above circuit by taking an example of addition of two 4 bit binary numbers. Let us add 1011 with 1101.

$$\begin{array}{r} 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ \hline 1\,1\,0\,0\,0 \\ \hline \end{array}$$

$$\begin{aligned} Here,\ & A_1 = 1,\ A_2 = 1,\ A_3 = 0\ A_4 = 1. \\ & B_1 = 1,\ B_2 = 0,\ B_3 = 1\ B_4 = 1. \end{aligned}$$

As there is no previous carry C_0=0.

$$\begin{aligned} Now,\ & C_0 + A_1 + B_1 = 0 + 1 + 1 = 10 \rightarrow S_1 = 0, C_1 = 1 \\ & C_1 + A_2 + B_2 = 1 + 1 + 0 = 10 \rightarrow S_2 = 0, C_2 = 1 \\ & C_2 + A_3 + B_3 = 1 + 0 + 1 = 10 \rightarrow S_3 = 0, C_3 = 1 \\ & C_3 + A_4 + B_4 = 1 + 1 + 1 = 10 \rightarrow S_4 = 1, C_4 = 1 \end{aligned}$$

Therefore, final result of the addition would be $C_4\ S_4\ S_3\ S_2\ S_1 = 11000$ .There are 1 bit, 2 bits and 4 bits parallel Adders ICs commercially available in market. For n bit parallel adder required number of such ICs are connected together. 4 bit parallel adder IC is 4008. In n bit parallel adder, output carry terminal of one IC would be connected with input carry terminal of next IC.

Binary Substractor

$$\begin{array}{r} -\ 1\,1\,0\,0\,1\,1 \\ 1\,0\,0\,1\,0\,1 \\ \hline 0\,0\,1\,1\,1\,0 \end{array}$$

1) 0 – 0 = 0
2) 0 – 1 = 1
3) 1 – 0 = 1
4) 1 – 1 = 0

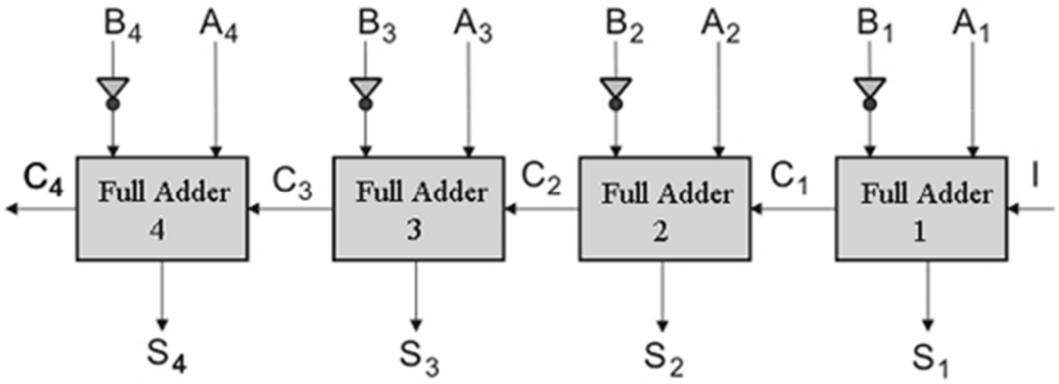
Before discussing about binary substractor, let us discuss about the method of substracting two multi bit binary numbers. For above substraction we used general rules which are, and borrow 1 which to be added to next higher significant bit of first binary number. Then same positioned bit of second binary number would be substracted from that. But there are other methods by which two binary numbers can be substracted confidently. One of these is 2’s complement method of substraction. Here, first binary number (from which another binary number to be substracted) is kept as it is. Then each bit of second binary numbers (which to be substracted) is complemented. Then 1 is added to LSB of complemented second binary number. This results 2’s complement of second binary number. Now, Finally we add first binary number with 2’s complement of the second binary number and we get the final result of substraction.

Here in the previous example, first binary number was 110011 and second binary number was 100101. Complement or 1's complement of 100101 is 011010. Now by adding 1 with LSB of this 1's complement

$$\begin{array}{r} 0\,1\,1\,0\,1\,0 \\ +\;1 \\ \hline 1\,1\,0\,1\,1 \end{array}$$

number we get, Now by adding first number, 110011 and 2's complement of second number i.e. 11011. We get, Hence, 4 bit substractor can be drawn like,

$$\begin{array}{r} 1\,1\,0\,0\,1\,1 \\ 0\,1\,1\,0\,1\,1 \\ \hline 1\,0\,0\,1\,1\,1\,0 \end{array}$$



Here, A4, A3, A2, A1 is minuend and B4, B3, B2, B1 is subtrahend. S4, S3, S2, S1 is result of substraction where C4 is final carry which is ignored.

Half Substractor

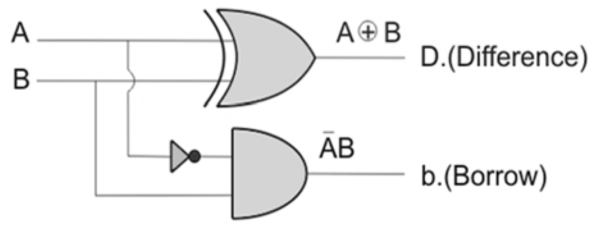
Half substractor is a combinational circuit which performs substraction of single bit binary numbers. The substraction combinations of two single bit binary numbers can be,

i. 0 - 0 = 0
ii. 0 - 1 = 1 with borrow 1
iii. 1 - 0 = 1
iv. 1 - 1 = 0

Now if we draw a truth table for that, with all differences (D) and borrow (b), we get,

Minuend (A)	Subtrahend (B)	Difference (D)	Borrow (b)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

hence, from truth table it is found that, $D = A \oplus B \ and \ b = \overline{A}B$ The above equations can be represented using logic gates.

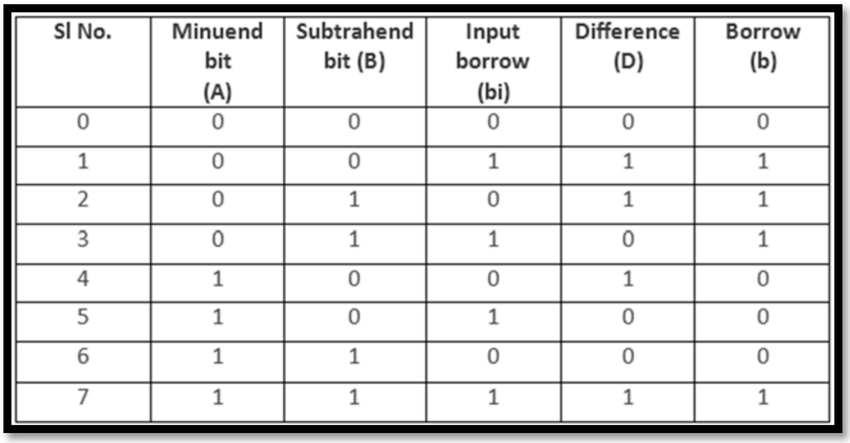


The above circuit is logical half substractor circuit.

Full Substractor

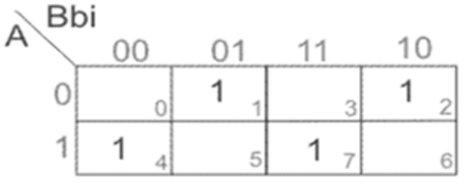
This is not practical to perform substraction only between two single bit binary numbers. Instead binary numbers are always multi bits. The substraction of two binary numbers is performed bit by bit from right (LSB) to left (MSB). During substraction of same significant bit of minuend and subtrahend, there may be one borrow bit along with difference bit. This borrow bit (either 0 or 1) is to be added to the next higher significant bit of minuend and then next corresponding bit of subtrahend to be subtracted from this. It will continue up to MSB. The combinational logic circuit performs this operation is called full substractor. Hence, full substractor is similar to half substractor but inputs in full substractor are three instead of two.

Two inputs are for the minuend and subtrahend bits and third input is for borrowed which comes from previous bits substraction. The outputs of full adder are similar to that of half adder, these are difference (D) and borrow (b). The combination of minuend bit (A), subtrahend bit (B) and input borrow (bi) and their respective differences (D) and output borrows (b) are represented in a truth table, as follow

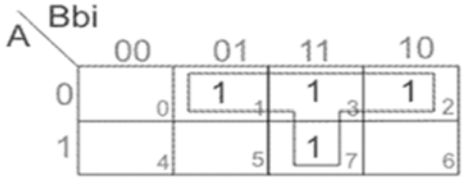


Sl No.	Minuend bit (A)	Subtrahend bit (B)	Input borrow (bi)	Difference (D)	Borrow (b)
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

Let us draw K-map for D and b.



K-map for D



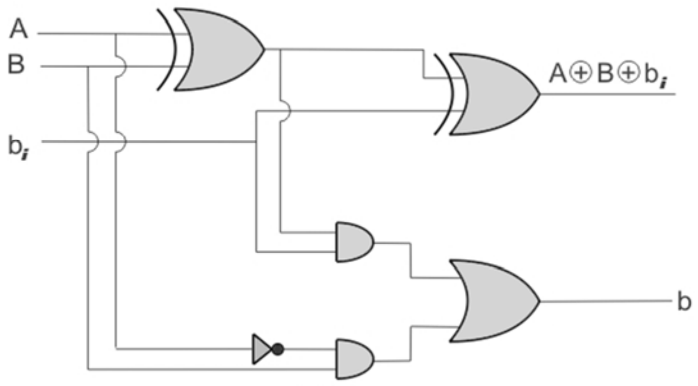
K-map for b

$$D = A\overline{B}\,\overline{bi} + ABbi + \overline{A}B\overline{bi} \;= \overline{bi}(A\overline{B} + \overline{A}B) + bi(\overline{A}\,\overline{B} + AB)$$

$$= \overline{bi}(A\overline{B} + \overline{A}B) + bi(\overline{A\overline{B} + \overline{A}B}) \;= bi \oplus A \oplus B = A \oplus B \oplus bi$$

$$b = \overline{A}\,\overline{B}bi + \overline{A}Bbi + \overline{A}B\overline{bi} + ABbi \;= \overline{A}B\,(bi + \overline{bi}) + (AB + \overline{A}\,\overline{B})\,bi$$

$$= \overline{A}B + (\overline{A \oplus B})\,bi$$



SUMMARY

- Binary Arithmetic: Binary arithmetic is essential part of all the digital computers and many other digital systems.Binary arithmetic is essential part of all the digital computers and many other digital systems.In binary number system there are only 2 digits 0 and 1, and any number can be represented by these two digits.
- In addition, an augend and an addend are added to find a sum. In the following equation, 6 is the augend, 3 is the addend, and 9 is the sum: 6 + 3 = 9. NOTE: Sometimes both the augend and addend are called addends. Sometimes the sum is called the total.
- Augend: the number to which another is added.
- Addends: Any of the numbers that are added together. Example: In 8 + 3 = 11, the 8 and the 3 are addends.
- Half Adder: The logical circuit that performs one bit binary addition is called half adder.
- Binary Adder: The binary adder is one of the basic combinational logic circuits as present state of input variables.
- Half Substractor:The half substractor is a combinational circuit which performs substraction of single bit binary numbers.
- Full Adder: The full adder is a conditional circuit which performs full binary addition. That means it adds two bits and a carry and outputs a sum bit and a carry bit.
- Full Substractor: The full substractor is similar to half substractor but inputs in the full substractor are three instead of two.

Self evaluation:

1. What is binary arithmetic logic?
2. What is half adder? How it design? Explain with truth table for half adder.
3. How many bits does a half adder add?
4. What is full adder? Explain with truth table.
5. What is the difference between half adder and full adder? Give examples of both types of adder.
6. What is binary adder? Draw its block diagram.
7. What is binary parallel adder? Explain in brief.
8. What is binary subtractor? Explain in brief.
9. What is half subtractor? Describe with logic gates.
10. What is full subtractor? Explain with truth table.

https://www.youtube.com/watch?v=7NqUpsQru8o[Full Adder]
http://isweb.redwoods.edu/INSTRUCT/CalderwoodD/diglogic/srflip.htm
http://info.iet.unipi.it/~luigi/biomedica/sito/cosc205.pdf

UNIT-6
Combinational Logic Circuit

Objectives

After competitions of this unit students will be able to

1. Define and design the different types of combinational logic circuit.
2. Design BCD to Seven Segment Decoder circuit.

Introduction

The digital Logic circuit may be combinational or sequential. A combinational circuit consists of logic gates whose output at any time is determined directly from the present combinations of inputs without regard for previous inputs. A combinational circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the input and generate signals to the outputs. This process transforms binary information from the given input data to the required output data. Clearly, both input and output data are represented by binary signals; i.e. they exist in two possible values one representing logic-1 and the other logic-0.

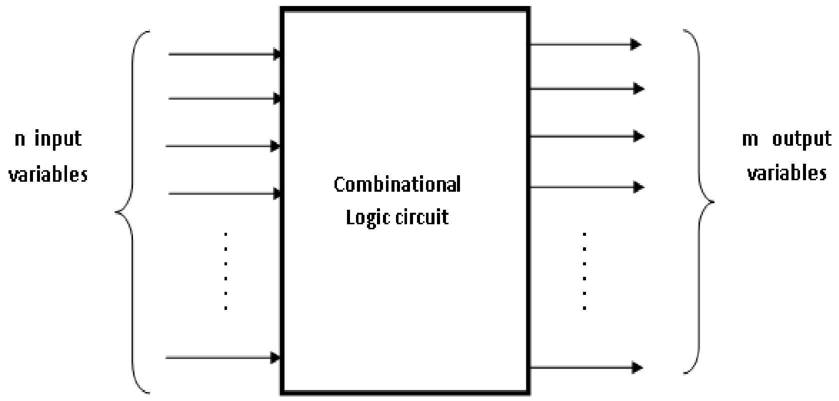


Figure 6-1

Block diagram of combinational circuit

Design Procedure of Combinational Circuit:

The design procedure involves the following steps.

1. The problem is stated.
2. The numbers of available input variables and required output variables are determined.
3. Each input and output variable is assigned a letter symbol.
4. The truth table that defines the required relations between input and output is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

Characteristics of Combinational Circuits:

1. The output of combinational circuit at any instant of time, depending only on the level present at input terminal.
2. There is no memory in combinational circuit so the previous state of input does not have any effect on the present state of the circuit.
3. A combinational circuit can have n numbers of inputs and m numbers of outputs.

Under the combinational logic, we will study about different types of circuit such as Multiplexer, Demultiplexer, encoder, decoder, and seven –segment- decoder

Multiplexers:

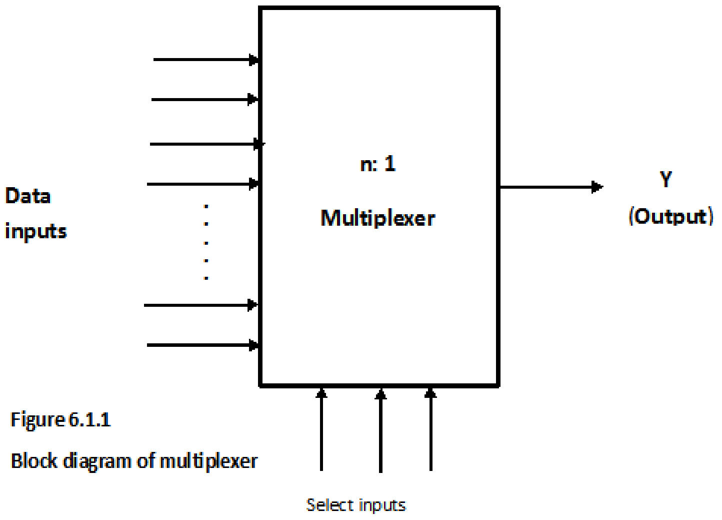


Figure 6.1.1
Block diagram of multiplexer

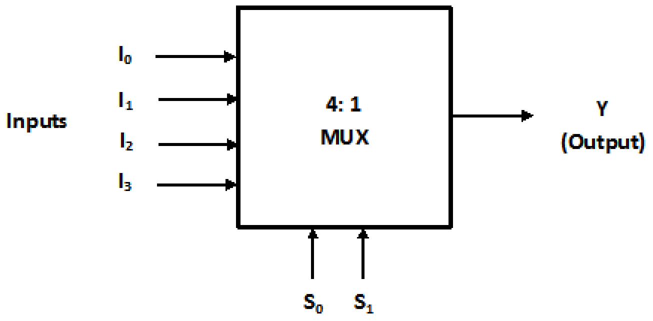
Multiplexing means transmitting a large numbers of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit which has n – data inputs one output and m select inputs with n=2m. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the enable input which is useful for cascading. It is generally active low terminal that means it will perform the required operation when it is low.

There are different types of multiplexer circuits. Some of them are as follows.

A 4-to- 1-line Multiplexer

S1	S0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 6.1.2
(b) Truth Table of 4:1 Mul



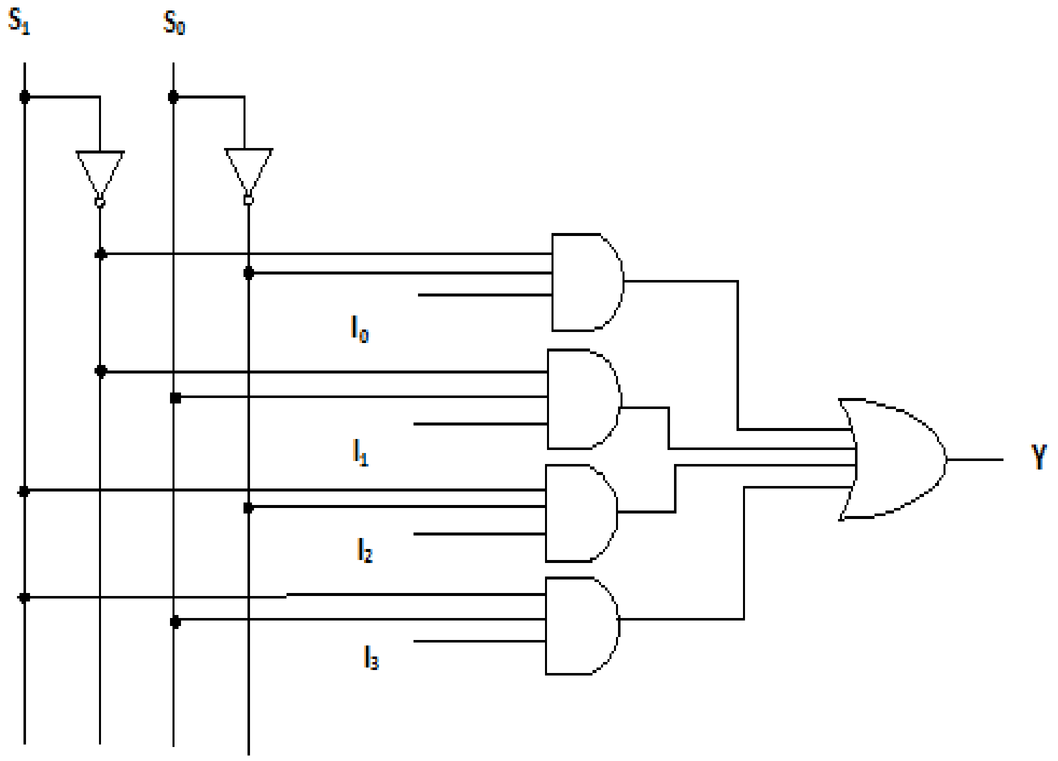
Select lines

FIGURE 6.1.2

(a) Blockdiagram of 4:1 Multiplexer

This multiplexer has four input lines, two select lines and one output line. This particular multiplexer consists of input lines I0 through I3, select lines S0 and S1 and output line Y. the output depends on the value of selection lines S0 and S1 which is control inputs. The control inputs determine which of the input line is transmitted to the output line. For example, in the figure when S0S1=00, I0 appears at the output line Y while all the other inputs are disabled. Similarly when S0S1=11 then bottom AND gate is enabled and input I3 is transmitted to the output line Y and so on. An example of 1-to-4 multiplexer is IC 74153 in which output is same as the input. Another example of 4-to-1 line multiplexer is 45352 in which the output is the complement of input. IC 74150 is a 16-to-1 multiplexer.

Now we can obtain the Boolean expression from the relationship between input and output as shown in the table 6.1.2(b) as follows.



(c) Logic diagram of 4:1 Multiplexer

$Y=\overline{S1}.\overline{S0}.I0+\overline{S1}.S0.I1+S1.\overline{S0}.I2+S1.S0.I3$

An 8-to- 1-line Multiplexer:

In this particular multiplexer, there are eight input lines I0 through I7, three selection lines S2, S1 and S0 and one output line Y. the output depends only on the selection lines S2, S1 and S0 which is control input. The control inputs determine which of the input (I0 through I7) is transmitted to the output Y. For example, as shown in the figure, if S2S1S0=000, I0 appears to the output line while all the other inputs are disabled. Similarly, when S2S1S0=111, I7 appears to the output line Y and all the other inputs are disabled and so on. The figure 6.1.3 (a) below shows the block diagram of an 8-to-1 multiplexer with enable input that enable or disable the multiplexer.

S_2	S_1	S_0	Y
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

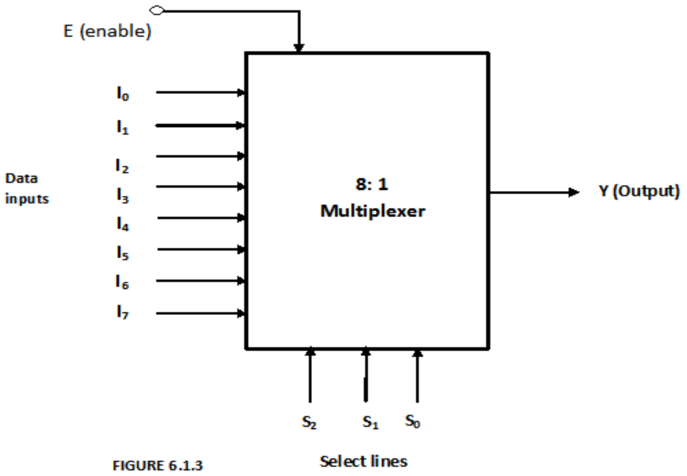


FIGURE 6.1.3

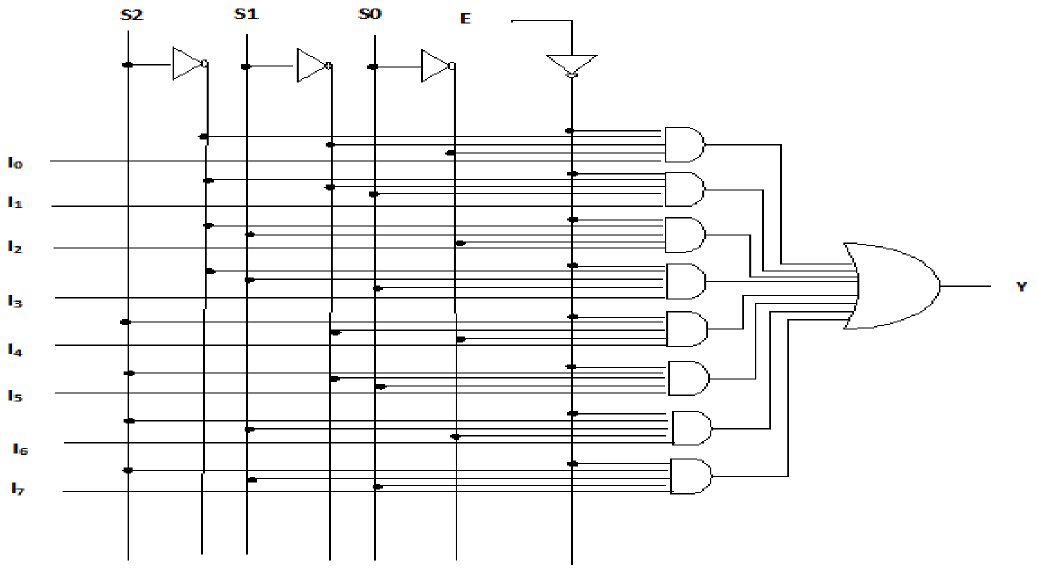
(a) Block diagram of 8:1 Multiplexer

Table 6.1.3
(b) Truth Table of 8:1 Multiplexer

Logical Expression:

$Y=\overline{S2}.\overline{S1}.\overline{S0}.I0+\overline{S2}.\overline{S1}.S0.I1+\overline{S2}.S1.\overline{S0}.I2+\overline{S2}.S1.S0.I3+S2.\overline{S1}.\overline{S0}I4+$ $S2.\overline{S1}.S0.I5+S2.S1.\overline{S0}.I6+S2.S1.S0.I7$

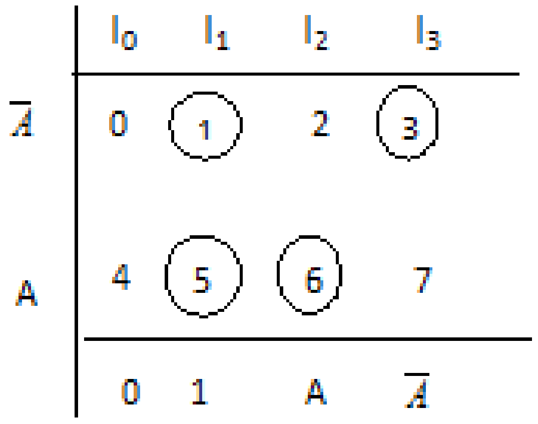
We can draw the logic diagram of 8-to-1 multiplexer using the Boolean expression, eight AND gates and three NOT gates in the figure 6.1.3 (c).



(c) Logic diagram of 8:1 Multiplexer

Example 1: Implementing $F(A, B, C) = \sum(1,3,5,6)$ with a multiplexer

Minterms	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0



	I_0	I_1	I_2	I_3
$\bar{A}$	0	1	2	3
A	4	5	6	7
	0	1	A	$\bar{A}$

(c) Implementation table

Table 6.1.4 (b) Truth Table of implementations of 4:1 Mux

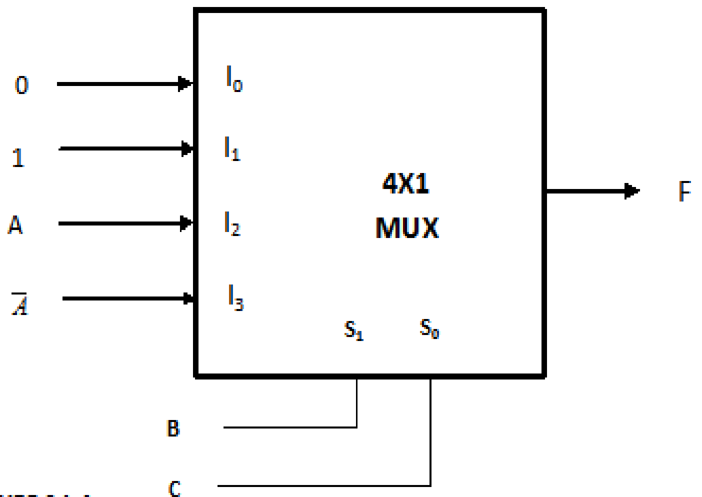


FIGURE 6.1.4

(a) Implementation of 4:1 Multiplexer

The function F (A, B, C) $= \sum(1,3,5,6)$ can be implemented with a 4-to 1-line multiplexer as shown in the figure 6.1.4 (a) above. The variables B and C are applied to selection lines S1 and S0 respectively. The variable B is connected to the high-order selection lines and C is connected to the next lower selection lines. Now consider a variable A, this variable will be complemented in the first half of list of the eight minterms. The second half of variable A will be uncomplemented. It means variable A is complemented for minterms 0 to 3 and uncomplemented for the minterms 4 to 7.

List the inputs of the multiplexer and under them list all the minterms in two rows. The first row list all the minterms where A is complemented and the second row list all the minterms where A is uncomplemented as shown in the implementation table 6.1.4(c). Circle all the minterms of the function.

Rules for Constructing Implementation Table

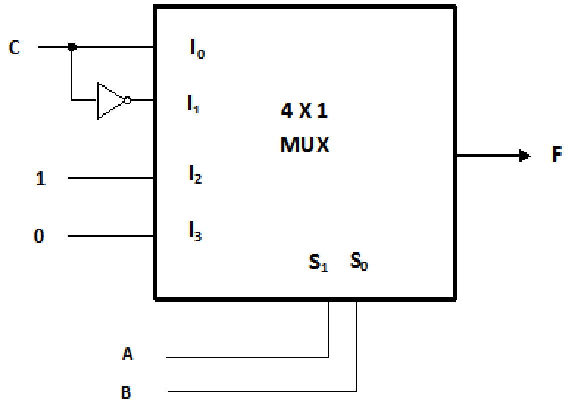
1. If the two minterms in the column are not circled, apply 0 to the corresponding multiplexer input.
2. If the two minterms are circled, apply 1 to the corresponding multiplexer input.
3. If the bottom minterm is circled and the top is not circled, apply A to the corresponding multiplexer input.
4. If the top minterm is circled and the bottom is not circled, apply $\overline{A}$ to the corresponding multiplexer input.

Example 2: Implementing F (A, B, C) $= \sum(1,2,4,5)$ with a multiplexer

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

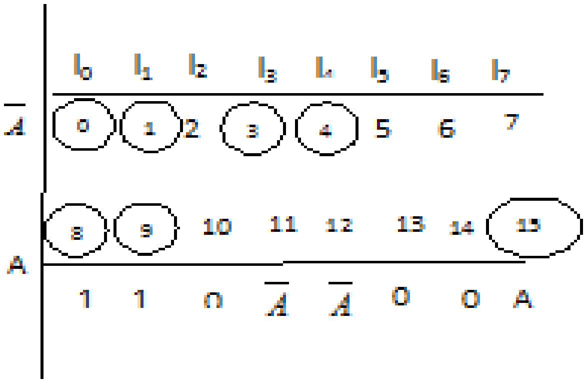
Table 6.1.5

(b) Truth Table of implementations of 4:1 Mux



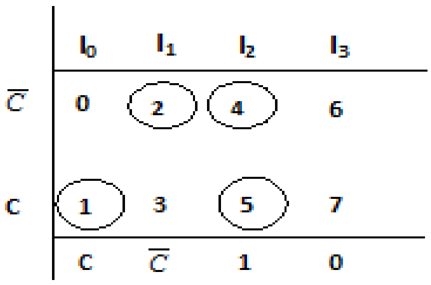
E 6.1.5

plementation of 4:1 Multiplexer


(a) Implementation table of 8:1 Multiplexer

In this example, variables A and B are connected to selection lines S_1 and S_0 respectively, and the variable C in the rightmost position is for the data inputs of the multiplexer. The variable C is complemented for the minterms 0, 2, 4, and 6, and it is uncomplemented for the minterms 1, 3, 5 and 7 in the implementation table 6.1.5 (c). Circle the minterms of the given function by using the rules stated in the example1.We will get the multiplexer input data. It is not always necessary to choose the leftmost variable for the data input of the multiplexer as in example1. We can choose any one of the variable for the inputs of the multiplexer.

Example 3: Implementing F (A, B, C) = $\sum(0,1,3,4,8,9,15)$ with a multiplexer.


(c) Implementation table

This is a four variable function so we need three selection lines and eight inputs to the multiplexer. The variables B, C and D are connected to the selection lines S2, S1, S0 respectively. The implementation table is shown below in figure 6.1.6 (b). The first half of the minterms are associated with complements of A and the second half of the minterms are associated with uncomplemented A. Now we can apply the rules to find the value of multiplexer inputs by circling the minterms of the function from the implementation table of 6.1.6 (a)

Example 4: Implement the following function using 4-to-1 multiplexer.

Y (A, B, C) =(2, 3, 5, 6)

Let us take B, C as the select bits and A as input. To decide the input we write.
Y = A'BC'+A'BC+AB'C+ABC'
= 0 if B=0, C=0
= A if B=0, C=1
= 1 if B=1, C=0
= A' if B=1, C=1

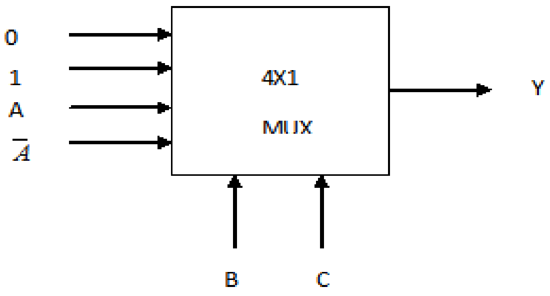


Fig: 6.1.7 (a) implementation of 4X1 MUX

Applications of Multiplexer

Multiplexer is used where multiple data can be transmitted using a single transmission line. Following are some of the areas where multiplexer can be implemented.

1. Communication system.
2. Telephone network.
3. Computer memory
4. Transmission from computer system to the satellite

Demultiplexer

It is a combinational circuit which performs the reverse operations of the multiplexer i.e. it receives one input and transmits it to over several output lines. It has only one input, n outputs and m select input lines. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. In short it is also called as DMUX. The enable input decides whether the circuits is operational or not, if this enable is zero the circuit is not operational and we will have zero at all the output

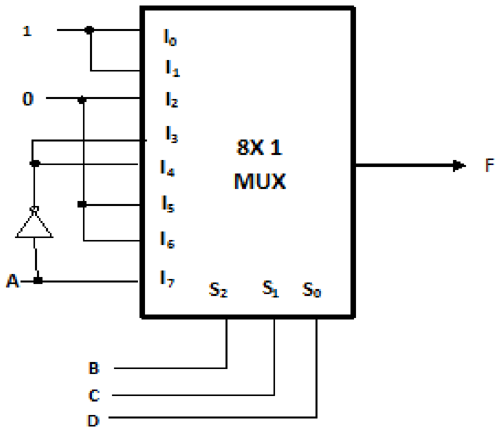


FIGURE 6.1.6

(b) Implementation of 8:1 Multiplexer

terminals. If the enable input is high, the demultiplexer is operational and the data input is transferred to any of the output line depending upon the select inputs. There is a relationship between select line and output. If n is the number of output lines and m is the number of select lines, then $n=2^m$. Taking the log on both side then $m=\log_2 n$. so we can find the select lines depending upon the output lines. Let us suppose that there is n=4 so, putting the value of n in $m=\log_2 n$, we will get m=2 select lines. So we can find the no of select lines depending upon the number of output lines. The figure 7.1 shows the common structure of demultiplexer.

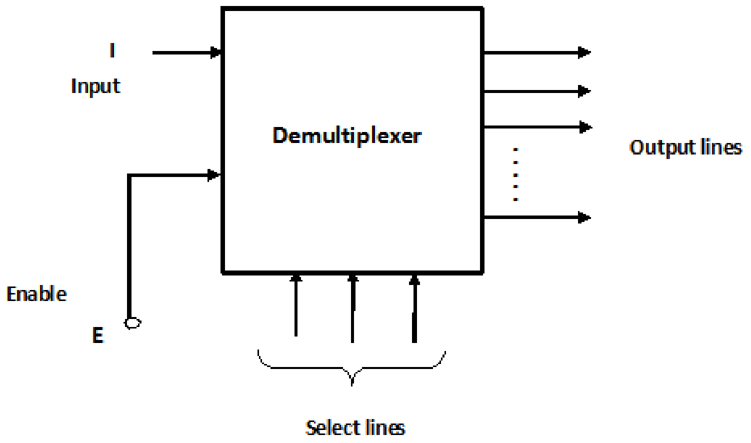


FIGURE 7.1

(a) Block diagram of Demultiplexer

There are different types of demultiplexer based on the output configuration such as 1:4, 1:8, and 1:16. These multiplexers are available in different ic packages and some of the most commonly used demultiplexers IC includes 74139 (Dual 1:4 DEMUX), 73136(1:8 DEMUX), 74154(1:16 DEMUX), 74159(1:16 open controller type) etc.

1-to-2 Demultiplexer

This demultiplexerconsists of one input line, two output lines and one select line. The outputs of the demultiplexer depend on the signals provided by the select line to one of the two output lines. The figure 7.1.1(a) shows the block diagram of 1-to-2 demultiplexer with additional enable input.

Select	Input	Output	
E	S_0	Y_0	Y_1
0	0	0	0
0	1	0	0
1	0	I	0
1	1	0	I

Table 7.1.1 (b) Truth Table of 1:2 DMUX

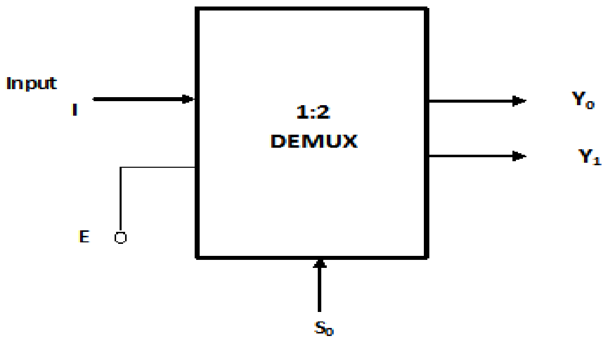


FIGURE 7.1.1

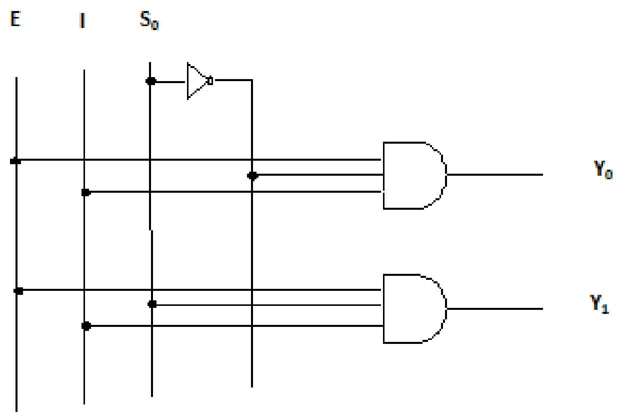
(a) Block diagram of 1:2 Demultiplexer

The truth table of 1:2 Demultiplexer is shown in the table 7.1.1 (b), when E =0 and S=0, demultiplexer is not operational so both the output Y_0 and Y_1 is 0. In the same way, when E=0 and S=1, both outputs are zero. When E=1 and S=0, demultiplexer is operational and data input is transferred to Y_0. When E=1 and S=1, data input is transferred to Y_1.

Now we can find the Boolean expression from the above truth table 7.1.1 (b) as follows.

$Y_0=E.\overline{S}.I$ and $Y_1=E.S.I$

Based on these two outputs Y_0 and Y_1 Boolean expression we can draw the logic diagram of 1:2 demultiplexers using two AND gates and one NOT gate in the figure 7.1.1 (c) below.



(c) Logic diagram of 1:2 Demultiplexer

1:4 Demultiplexer

A1-to-4 has single input I, two selection lines S1 and S0 and four outputs Y0 to Y3. The input data goes to any of the four output lines at a time for particular combinations of selection lines. This is also called 2-to -4 line demultiplexer which means that two select lines and four output lines.The block diagarm of 1-to-4 line demultiplexer is shown in the figure 7.1.2 (a).

The truth table of 1:4 demultiplexer is shown in the table 7.1.2 (b) below. From the truth table, it is clear that when S0=0 and S1=0 the data input is connected to output Y0 and when S0=0 and S1=1 the data input is connected to Y1.

Inputs			Outputs			
E	S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	I	0	0	0
1	0	1	0	I	0	0
1	1	0	0	0	I	0
1	1	1	0	0	0	I

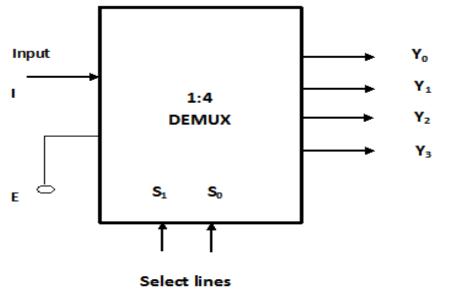


FIGURE 7.1.2

(a) Block diagram of 1:4 Demultiplexer

Table 7.1.2 (b) Truth Table of 1:4 Dmultiplexer

From the truth table of 7.1.2 (b), the relation between input- outputs of the demultiplexer can be expressed as minterms and are given below,

$Y0=I\overline{S1}.\overline{S0}$

$Y1=\overline{S1}.S0.I$

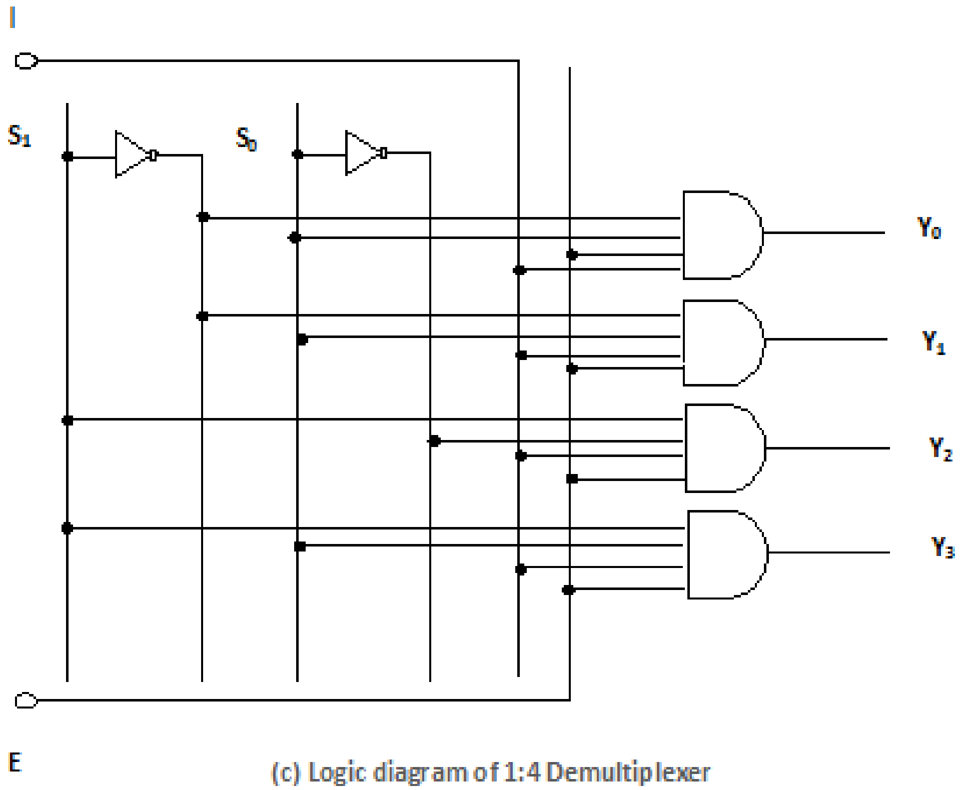
$Y2=S1.\overline{S0}.I$

$Y3=S1.S0.I$

Where" I" is the input data line S0 and S1 are the select lines and Y0 to Y3 are output lines.

Now, we can draw the logic circuit of 1:4 demultiplexers using four AND gates and two NOT gates in the figure 7.1.2 (c).

This type of demultiplexer is available in IC form and a typical IC 74139 is the most commonly used dual 1:4 demultiplexer.



(c) Logic diagram of 1:4 Demultiplexer

1:8 Demultiplexer

This type of demultiplexer consist of single input lineI and three selection lines S0, S1 and S2, and eight output lines from Y0 to Y7.

It is also called 3-to-8 line demultiplexer due to three select input lines. It sends one input line to one of 8 output lines depending on the combinations of three selection lines. The figure 7.1.3 (a) shows the common block diagram of 1:8 demultiplexer.

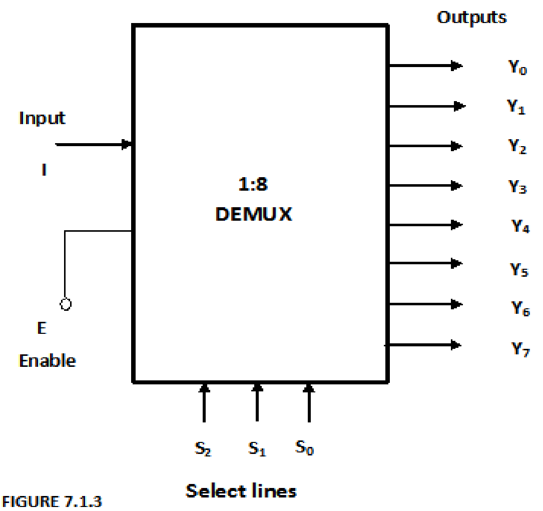


FIGURE 7.1.3

(a) Block diagram of 1:8 Demultiplexer

The truth table of the 1:8 demultiplexer is shown in the table 7.1.3 (b).

Input	Select inputs			Outputs							
I	S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
I	0	0	0	0	0	0	0	0	0	0	I
I	0	0	1	0	0	0	0	0	0	I	0
I	0	1	0	0	0	0	0	0	I	0	0
I	0	1	1	0	0	0	0	I	0	0	0
I	1	0	0	0	0	0	I	0	0	0	0
I	1	0	1	0	0	I	0	0	0	0	0
I	1	1	0	0	I	0	0	0	0	0	0
I	1	1	1	I	0	0	0	0	0	0	0

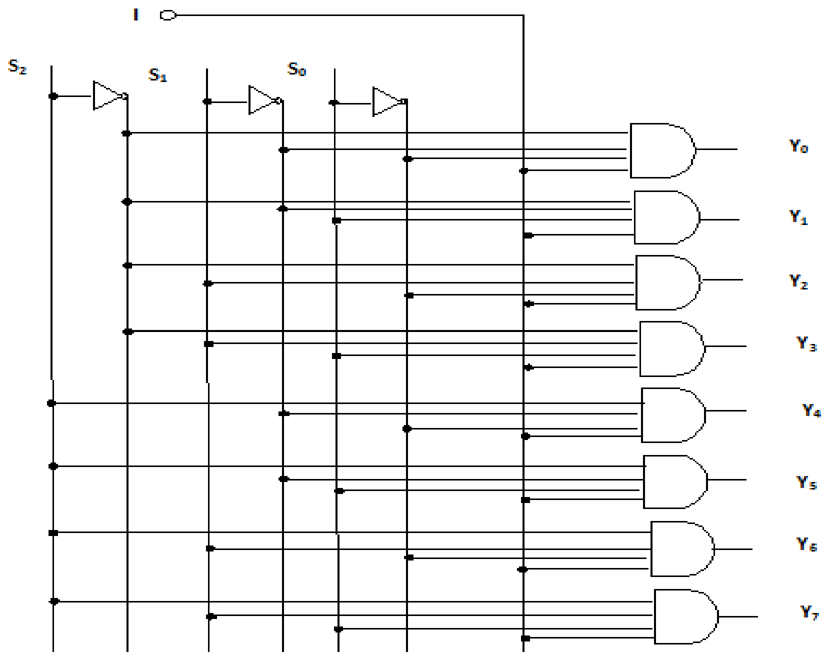
Table 7.1.3(b) Truth Table of 1:8 Dmultiplexer

The input line I is connected to one of the eight output lines from Y0 to Y7depending upon the selection lines S0, S1 and S2. If S2S1S0=000, then input I is connected to Y0. Similarly, if S2S1S0=001, then the input I is connected to Y1 and so on.

From the truth table 7.1.3 (b) the Boolean expression can be written as follows.

1. Y0=I.$\overline{S2}.\overline{S1}.\overline{S0}$
2. Y1=I. $\overline{S2}.\overline{S1}$.S0
3. Y2=I. $\overline{S2}$.S1. $\overline{S0}$
4. Y3=I. $\overline{S2}$.S1.S0
5. Y4=I.S2. $\overline{S1}.\overline{S0}$
6. Y5=I.S2. $\overline{S1}$.S0
7. Y6=I.S2.S1. $\overline{S0}$
8. Y7=I.S2.S1.S0

On the basis of these Boolean expressions, we can clearly draw the logic circuit of 1:8 demultiplexer using eight AND gates and three NOT gates in the figure 7.1.3 (c) below.



(c) Logic diagram of 1:8 Dmultiplexer

Applications of Demultiplexer

1. Demultiplexer is used to connect a single source to a multiple destinations. The main applications of Demultiplexer are in the communication system where multiplexers are used.
2. Communication system: The multiplexer is used to carry the audio, video, image and other forms of data in a single transmission line so the demultiplexer receives the output signal of the multiplexer and converts them back to the original signal.
3. Arithmetic and Logic Unit (ALU): The Demultiplexer is used to store the outputs of the ALU unit to register or storage units.
4. It is used in serial to parallel data converter circuit.

Code Converters

A code converter is a logic circuit that converts one form of digital code to another form i.e. it converts BCD code to Binary code, HEX code to Binary etc. Combinational circuit performs this transformation by means of logic gates.

Decoders

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2n unique output lines. If the n-bit decoded information has unused or don't care combinations, the decoder output will have fewer than 2noutputs.

The decoder presented here is called n-to–m line decoders, where m≤2n .Their purpose is to generate the 2n or fewer minterms of n input variables. Consider the 3-to-8line circuit of fig.(…). The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variable. The three inverters provide the complements of the inputs, and each one of the AND gates one of the minterms. The application of this Decoder is to convert binary- to-octal. The input is in the form of binary numbers, and the outputs will be represent eight digit octal number system.

A decoderwith an enable input can function as a demultiplexer. A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2n output lines. The selection of a specific output line is controlled by the bit value of the n selections lines. The decoder in fig.() can function as a demultiplexer if E is taken as data input line and A and B lines are takes as the selections lines. This is shown in the fig.(8.1). The input E has occurred to all outputs, but the input information is directed to only one of the output lines with the binary value of the two selections lines A and B. The input and output relation can be verified from the truth table shown in the fig.(8.1). For example, if the selection lines AB=10, then output D2 will be 0 and all other outputs are maintained at 1.

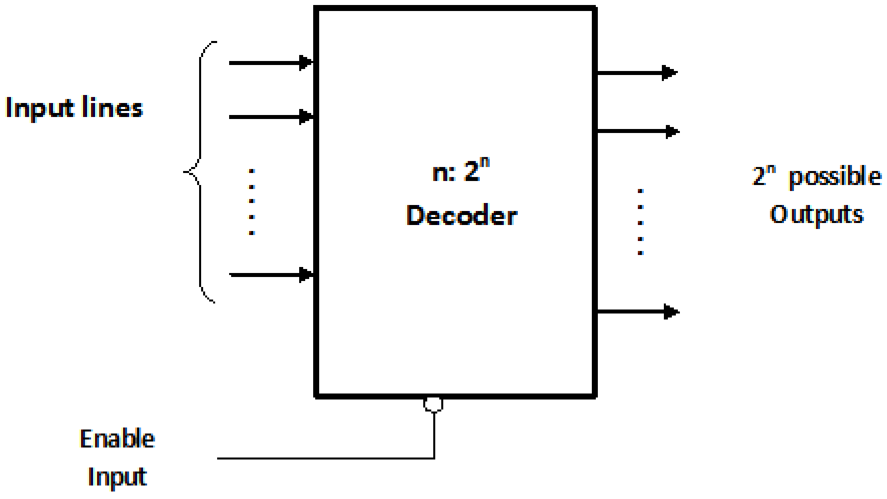


FIGURE 8.1
Block diagram of Decoder

Binary Decoder

Abinary decoder is a combinational circuit that converts binary code of n input lines to the one of the 2n output lines depending on the combinations of input lines. It is used when there is needed to activate exactly one of 2n output lines based on the n input values.

The Figure(8.2) shows the general structure of Binary Decoder in which encoded information is accepted at n input lines and the output is produced at 2n possible output lines.

Depending on the numbers of input lines, the binary input lines can be 2-bit, 3-bit and 4-bit code. Usually the number of bits in output code is more than the number of input lines. The most common Decoders are 2-to-4 line, 3-to-8 line and 4-to-16 line ones.

2- to-4 binary Decoder

In a 2-to-4 binary decoder two inputs are decoded into four outputs so it is called 2-to-4 line decoder. Only one output is activated at a time while the other outputs are maintained at logic level 0 and the output which is high depends on the combinations of the bit values of two inputs. The block diagram of 2-4 Binary Decoder is shown in the figure 8.2 (a) and its truth table is in the Table 8.2 (b).

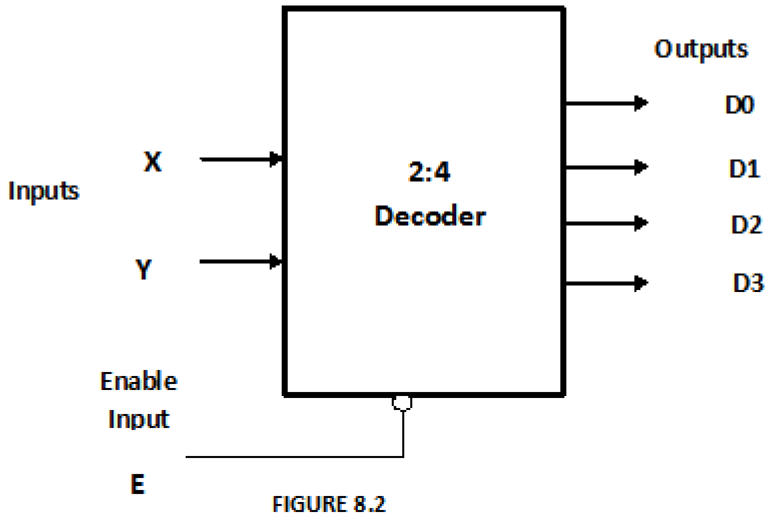


FIGURE 8.2

(a) Block diagram of 2:4 Decoder

For the given input, the outputs D0 through D3 are active high if the enable input E=1. When E=0 and X=Y=0, the decoder is on the off state. So we simply use don't care condition.

When E=1 and XY=00, the decoder output D0 is high while all the other outputs are low. When X=0 and Y=1 then D1 is active. The relationship between inputs and outputs are clearly shown in the truth table 8.2 (b).

Inputs			Outputs			
E	X	Y	D0	D1	D2	D3
0	X	X	0	0	0	0
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

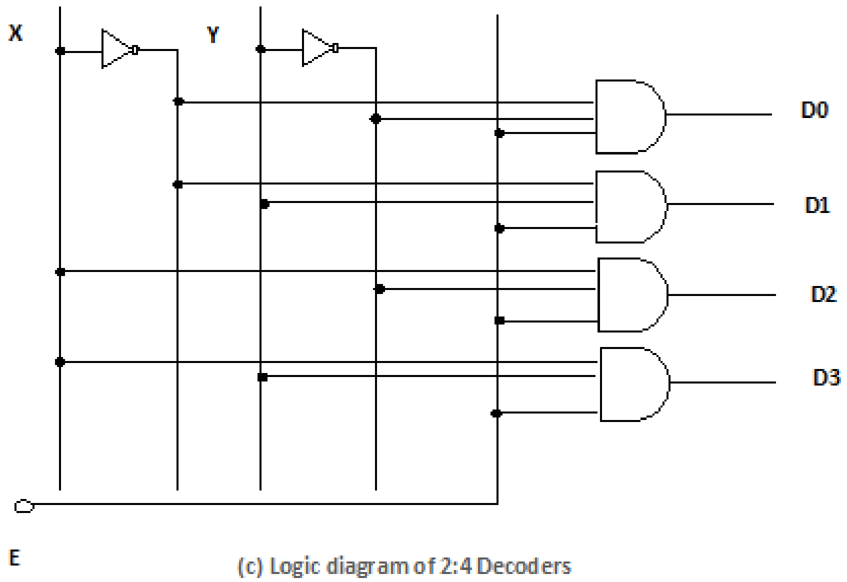
Table 8.2 (b) Truth Table of 2:4 Decoders

From the above truth table we can obtain the Boolean expression as follows.

$D0=\overline{X}.\overline{Y}$ $D2=X.\overline{Y}$

$D1=\overline{X}.Y$ $D3=X.Y$

Now we can clearly draw the logic diagram of 2:4 decoders using the Boolean expressions, four AND gates and two NOT gates that provides the complement inputs in the figure 8.2 (c). A common enable input is connected to all the AND gates such that when E=0 all outputs are 0 and E=1 depends on the input X and Y, outputs are generated. Each output represents one of the minterms of the two input variables.



(c) Logic diagram of 2:4 Decoders

3-to-8 line Decoder:

In3-to-8 line decoder, three inputs are decoded into eight output lines. It has three inputs as X, Y and Z and eight outputs through (D0 to D7). Based on the combinations of the three inputs only one output is selected. The table 8.3 (a) shows the truth table of 3-to-8 line Decoder. The enable input E is provided to activate the Decoded output depends on the input combinations X, Y and Z. If X=Y=Z=1, then output D7=1 and all the outputs remains at the logic low. From the truth table of 3-to-8 line Decoder, we can express the Boolean equations as follows.

Inputs			Outputs							
X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Table 8.3 (a) Truth Table of 3 to 8 Decoders

$D0=\overline{X}.\overline{Y}\,\overline{Z}$

$D1=\overline{X}.\overline{Y}.Z$

$D2=\overline{X}.Y.\overline{Z}$

$D3=\overline{X}.Y.Z$

$D4=X.\overline{Y}.\overline{Z}$

$D5=X.\overline{Y}.Z$

$D6=X.Y.\overline{Z}$

$D7=X.Y.Z$

Using the above minterms we can draw the logic diagram of 3-to-8 line decoder using eight AND gates and there NOT gates in the figure 8.3 (b). Only one output is high at a given time for particular combinations of inputs. A particular application of this

Decoder is binary to octal conversion. The input variable may represents a binary number, and the output will then represents the eight digits in the octal number system.

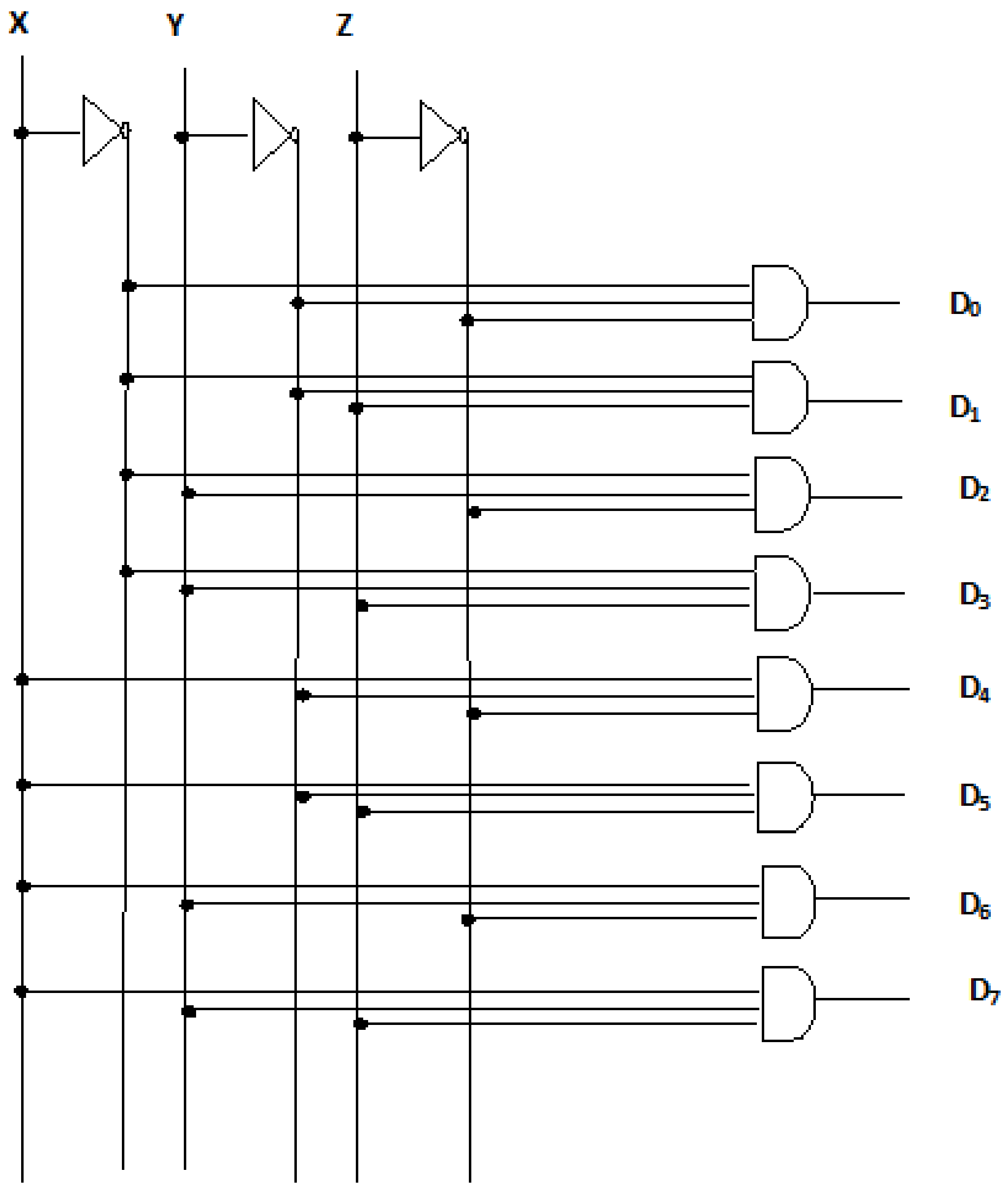


FIGURE: 8.3

(b) Logic diagram of Binary to Octal Decoder

BCD to Decimal Decoder

In BCD to Decimal decoder, the elements of information are 10 decimal digits represented by the BCD code. The functions of the decoder are to supply one output for each decimal digit. Each output is equal to only one when the input variables constitute bit combinations corresponding to the decimal digit as represented in BCD. The table 8.4 (a) is a truth table which shows the input-output relations of the decoder. Only the first 10 input combinations are valid code assignments, the last six are not used due to don't care conditions. The table lists the output for six 'not used' ones.

Combinations of the BCD code, but these combinations clearly have no meaning in this circuit. Decoders and encoders have many applications in digital systems. Decoders are useful for displaying discrete elements of information stored in registers. For example, a decimal digit represented in BCD and stored in four-cell register may be displayed with the help of a BCD to decimal decoder. Another application of decoder circuit is in the generations of Timing and sequencing signals for control purpose.

Inputs				Outputs									
W	X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	1	0	0	0	0	0	1	0
1	0	1	1	0	0	0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1	0	0	0	1	0
1	1	0	1	0	0	0	0	0	1	0	0	0	1
1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1	0	1

Table 8.4 (a) Truth Table BCD to Decimal Decoders

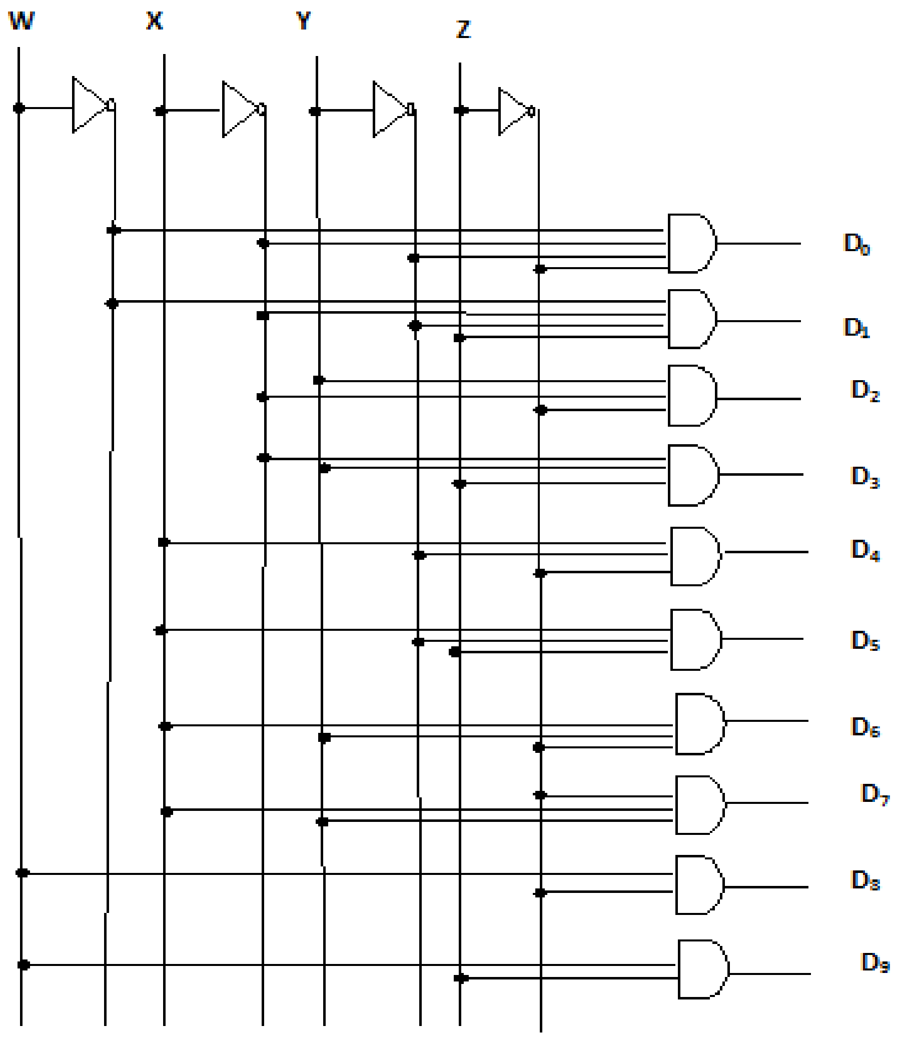


FIGURE 8.4

(b) Logic diagram of BCD to Decimal Decoder

Applications of Decoders

1. Decoders are used to route input data to a specified output line in addressing core memory where input data is to be stored in a specific memory location.
2. Decoders are used in the code conversions.
3. Decoders may also be used for data distribution i.e. demultiplexing.
4. It is very useful for generations of timing and sequencing signals for control purpose.

Encoders

An encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. It has 2n input lines and n output lines. The output lines generate the binary code corresponding to the input value. Encoder circuits are very useful for binary code formation when the discrete elements of information are each available from a single line. The encoders are very useful in digital electronics that translate the decimal values into the binary a value in order to perform the binary functions such as addition, subtraction, multiplication, etc. The figure below shows the block diagram of encoder which consists of 2n input lines and n output lines.

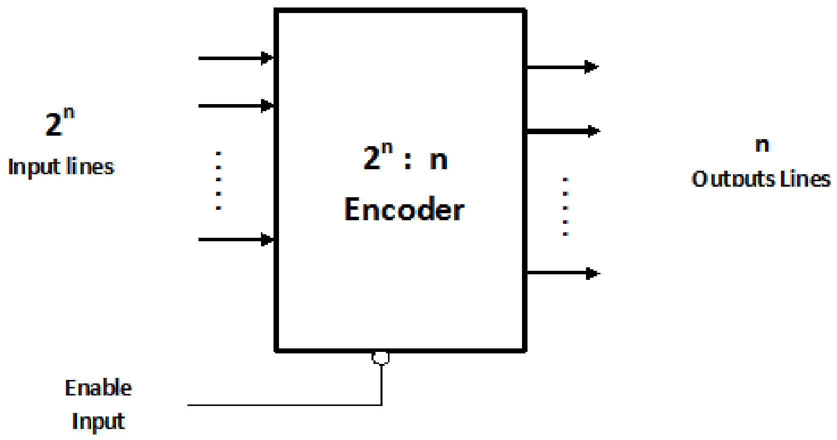


FIGURE 9.1
Block diagram of Encoder

There are varieties of encoder circuit. They are as follows.

- Octal to Binary Encoder
- Priority Encoder
- Decimal to BCD Encoder
- Hexadecimal to Binary Encoder

Octal to binary encoder:

In case of octal number system, the base or radix is 8 i.e. r=8 which means it has eight distinct digits and they are from (0 to 7). It consists of eight input lines and three output lines. Each input line corresponds to each octal digit and three outputs generate corresponding binary code. It is constructed with OR gates. It is assumed that only one input line can be equal to 1 at any time, otherwise the circuit has no meaning. Its truth table is shown in fig.(9.2)..

Inputs								**Outputs**		
D0	D1	D2	D3	D4	D5	D6	D7	B2	B1	B0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Table 9.2 (a) Truth Table of Octal to Binary Encoder

From the truth table, we can see that output B2 is high for the last four cases and it is low for the first four cases. So

B2=D4+D5+D6+D7.

Now for the B1: D2, D3, D6 and D7 inputs are in high and all the remaining inputs are at logic 0. So we can write

B2=D2+D3+D6+D7

Similarly, for B0: all the even numbers of inputs are high and all other remaining inputs are in low state. So we can write the Boolean expression as follows.

B0=D1+D3+D5+D7

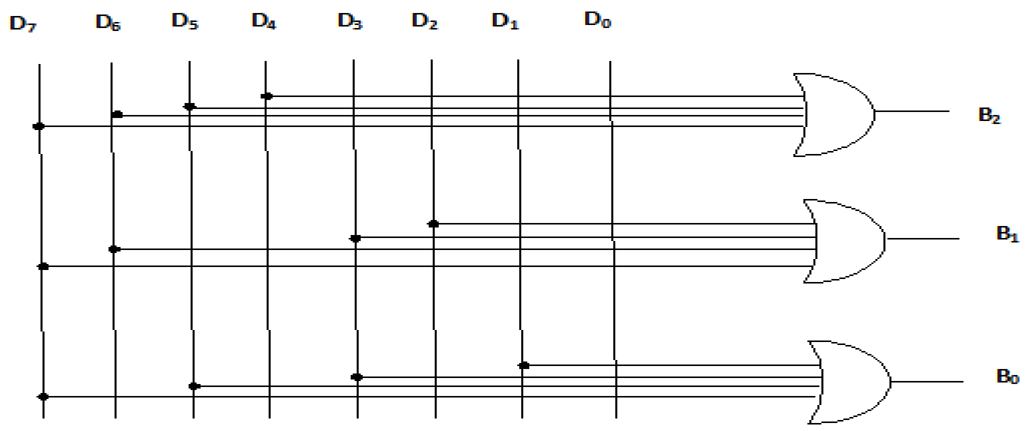


FIGURE 9.2

(b) Logic diagram of Octal to Binary Encoder

Decimal to BCD Encoder

The decimal is a number system in which the base or radix is 10, i.e. r =10. So the total number of digit is from (0 to 9). This encoder consists of 10 input lines and four output lines. Each input line corresponds to each decimal digit and four output lines correspond to the BCD code. It takes the decoded decimal data as an input and encodes it to BCD output to the output lines. The figure below shows the truth table of Decimal to BCD encoder. The truth table shows the BCD code for each decimal digit. The table shows the truth table of Hexadecimal to Binary Encoder.

Inputs										Outputs			
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D	C	B	A
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Table 9.3 (a) Truth Table of Decimal to BCD Encoder

From the truth table, we can see that A is high for the even numbers of inputs. So output A=D1+D3+D5+D7+D9 and all the other inputs is low.

Similarly, for B: the inputs D2, D3, D6, and D7 are high and remaining inputs are low state. Therefore we can write output B as

10. B=D2+D3+D6+D7

Now in case of output C, the inputs D4, D5, D6, and D7 are in high state and all other inputs are at low state. So we can write a Boolean function as

C=D4+D5+D6+D7

In the last case that is for the D output only last two inputs are high and remaining inputs are in logic state low. So we can express Boolean function as D=D8+D9

So the required Boolean expressions for the logic diagram of Decimal to BCD are as follows:

11. A=D1+D3+D5+D7+D9
12. B=D2+D3+D6+D7
13. C=D4+D5+D6+D7
14. D=D8+D9

Now we can easily draw the logic diagram of Decimal to BCD encoder using the above Boolean expression.

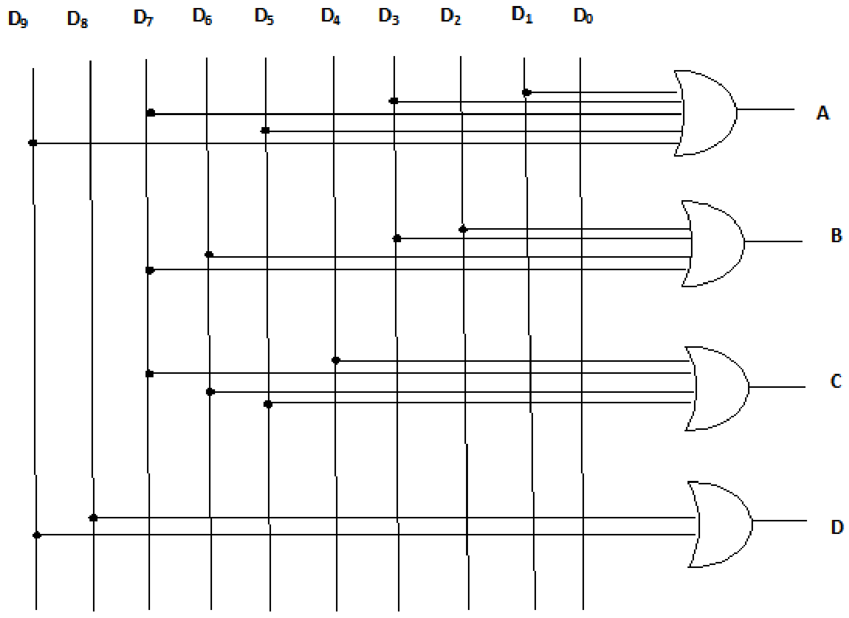


Fig. 9.3 (b): Logic circuit of decimal to BCD Encoder

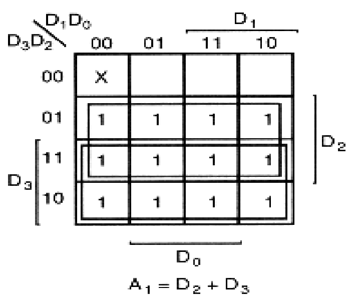
Priority Encoder

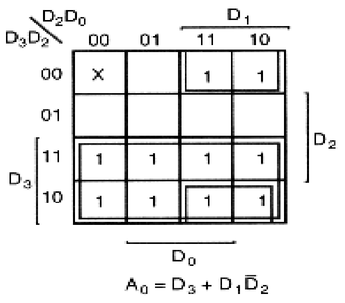
A priority encoder is an encoder circuit that includes the priority function. It operates in a manner that if two or more inputs are equal to 1 at the same time, then the input line with highest priority will be considered. The truth table for the priority encoder is given in table. The X's are the don't-care conditions that designate the fact that the binary value may be equal to either 0 or 1. Input D3 has the highest priority; so regardless of the value of the other inputs, when this input is 1, the output for x y is 11. D2 has the next priority level. The output is 1 0 if D2=1, provided that D3=0, regardless of the value of the other two lower priority inputs. The output D1 is generated only if the highest-priority inputs are 0, and so on down the priority level. A valid indicator v is set to 1 only when one or more of the inputs are equal to 1.If all inputs are 0, v is equal to 0, and the other two outputs of the circuit are not used. The priority encoder is implemented in figure. 9.4(c) using the following boolean functions.

$X=D2+D3$ $Y=D3+D1.\overline{D2}$ $V=D0+D1+D2+D3$

Inputs				Outputs		
D0	D1	D2	D3	X	Y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table 9.4 (a) Truth Table of Priority Encoder





(b) K Maps of Priority Encoder

Hexadecimal to Binary Encoder

In Case of hexadecimal number system the base or radix is 16, i.e. r=16, which means there are sixteen distinct digits in total. They are ranging from 0 to F. For example 15 is written F in hexadecimal number system. Similarly, 10 are written as A. It is very clear that 16 is the number of inputs for our decoder. So n=16. To find the number of output lines we have n=2m. Taking logs on both side we will get the values of output lines m=4. So we require a 4 bit to represent a single Hexadecimal number.

In-put	Outputs			
	B 3	B 2	B 1	B 0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

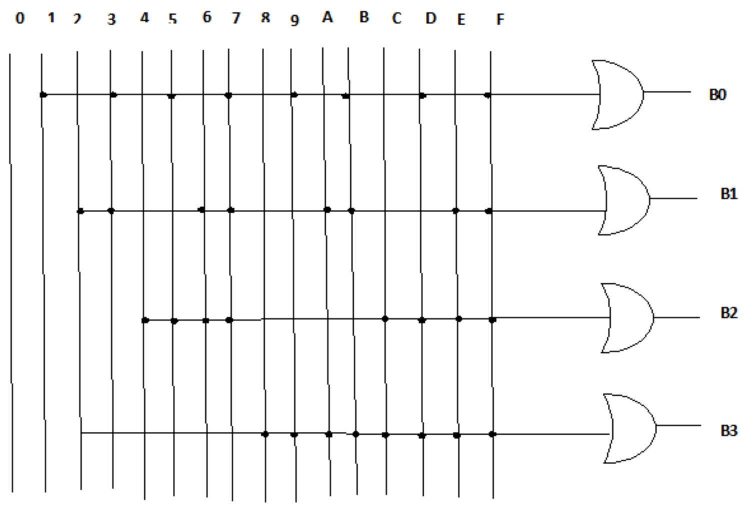


FIGURE 9.5
(b) Logic circuit of HEX to Binary Encoder

Table: 9.5: (a) Logic circuit of HEX to Binary Encoder

From the truth table of 9.5(a), we can clearly mention the outputs B0, B1, B2 and B3. B0 is high for all the odd numbers of inputs. That is it is high for inputs 1, 3, 5, 7, 9, B, D and F.

B1 is high for 2, 3, 6, and 7, A, B, E, and F.

similarly, B2 is high for 4,5,6,7,C,D,E,F. and the last output B3 is high for second half of the inputs and low for the first half of the inputs.

We can design a logic circuit of HEX to Binary encoder using the following Boolean expressions shows in the figure 9.5 (b).

B0=1+3+5+7+9+B+D+F
B1=2+3+6+7+A+B+E+F
B2=4+5+6+7+c+D+E+F
B3=8+9+A+B+C+D+E+F

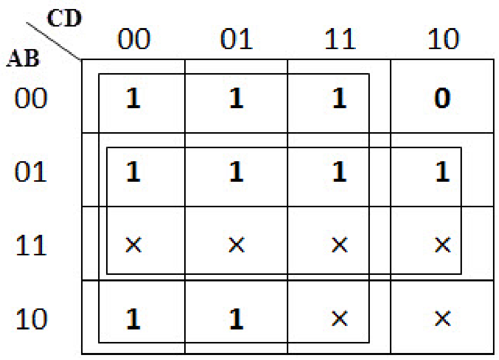
Seven -Segment Display Decoder

Introduction

The seven-segment decoder consists of seven LED's arranged in the rectangular fashion. Each of the seven LED is called segment because when illuminated the segment forms part of the numerical digits both decimal and Hex to be displayed. A LED or Light Emitting Diode is a solid state optical PN junction diode which emits light energy in the forms of Photons when the diode is in the forward biased mode. The basic idea to drive a seven-segment LEDs display is combinational logic circuit. The circuit is designed with four inputs and seven outputs, each representing an input to the seven-segment IC. Using K-Map, we can design a logic circuit for each of the input of the IC. The **figure 10.1** below shows the seven-segment display device images taken from the website http://www.electronicshub.org/seven-segment-displays/

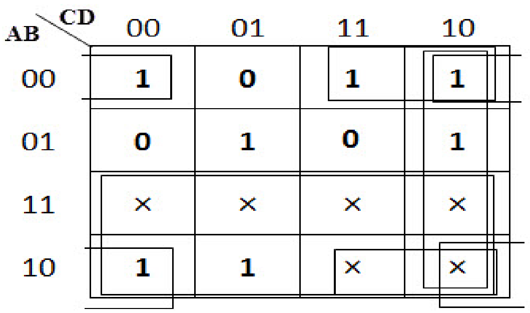
Applications of Seven –Segment Display

- The applications of seven segments are mostly in digital calculators, electronic meters, digital clocks, odometers, digital clocks, clock radios, etc.
- Today most of the 7 segment applications are using LCDs, because of low current consumption.



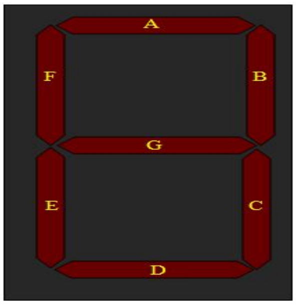
AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	×	×	×	×
10	1	1	×	×

$c = B + \overline{C} + D$

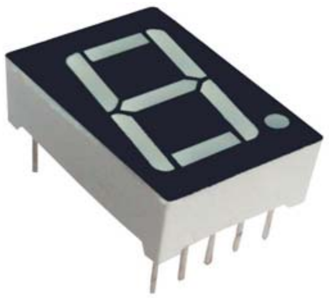


AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	×	×	×	×
10	1	1	×	×

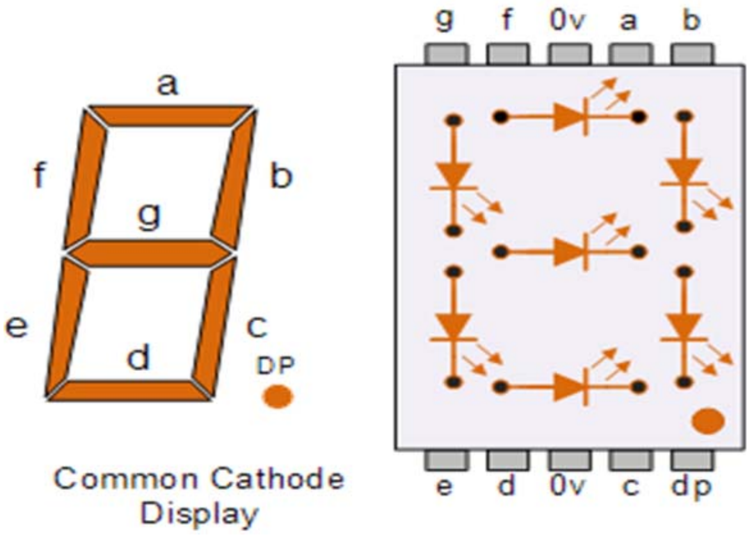
$d = \overline{B}\,\overline{D} + C\overline{D} + B\overline{C}D + \overline{B}C + A$



(b) Seven segment display device structure

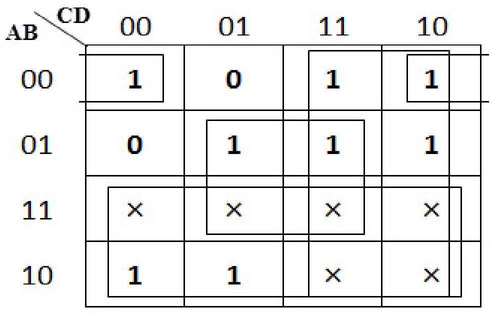


(a) Seven segment display Device

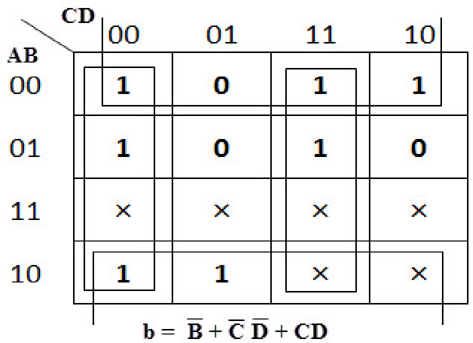


(c) Seven segment display Device diode placement

Figure 10.1:



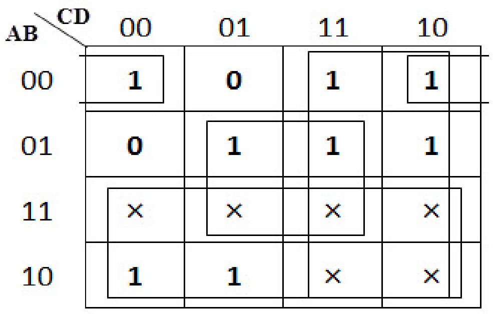
$a = A + C + BD + \overline{B}\,\overline{D}$



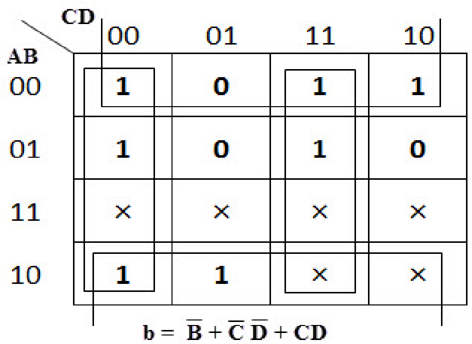
$b = \overline{B} + \overline{C}\,\overline{D} + CD$

Inputs				Outputs						
A	B	C	D	a	B	c	d	E	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
1	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

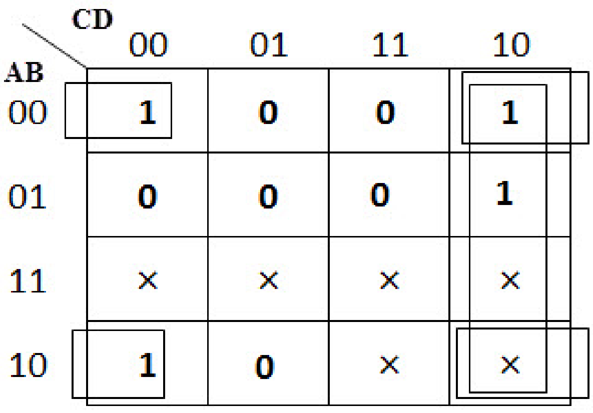
Table 10.1 : Truth table of seven segment Display



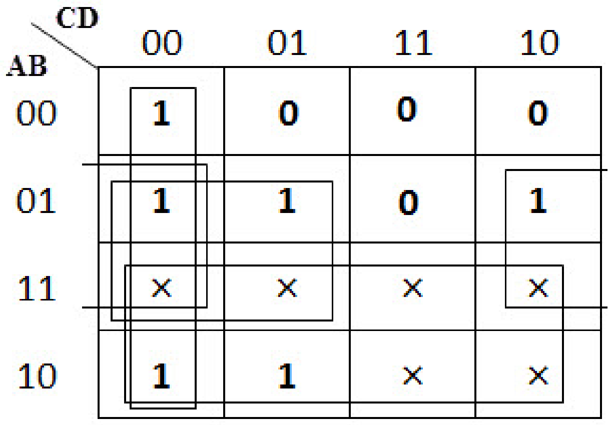
$a = A + C + BD + \overline{B}\,\overline{D}$



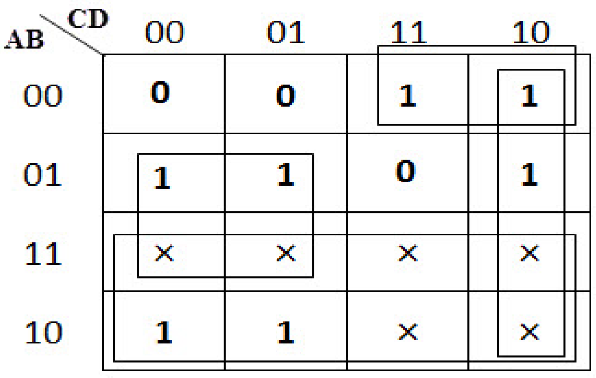
$b = \overline{B} + \overline{C}\,\overline{D} + CD$



$e = \overline{B}\,\overline{D} + C\,\overline{D}$

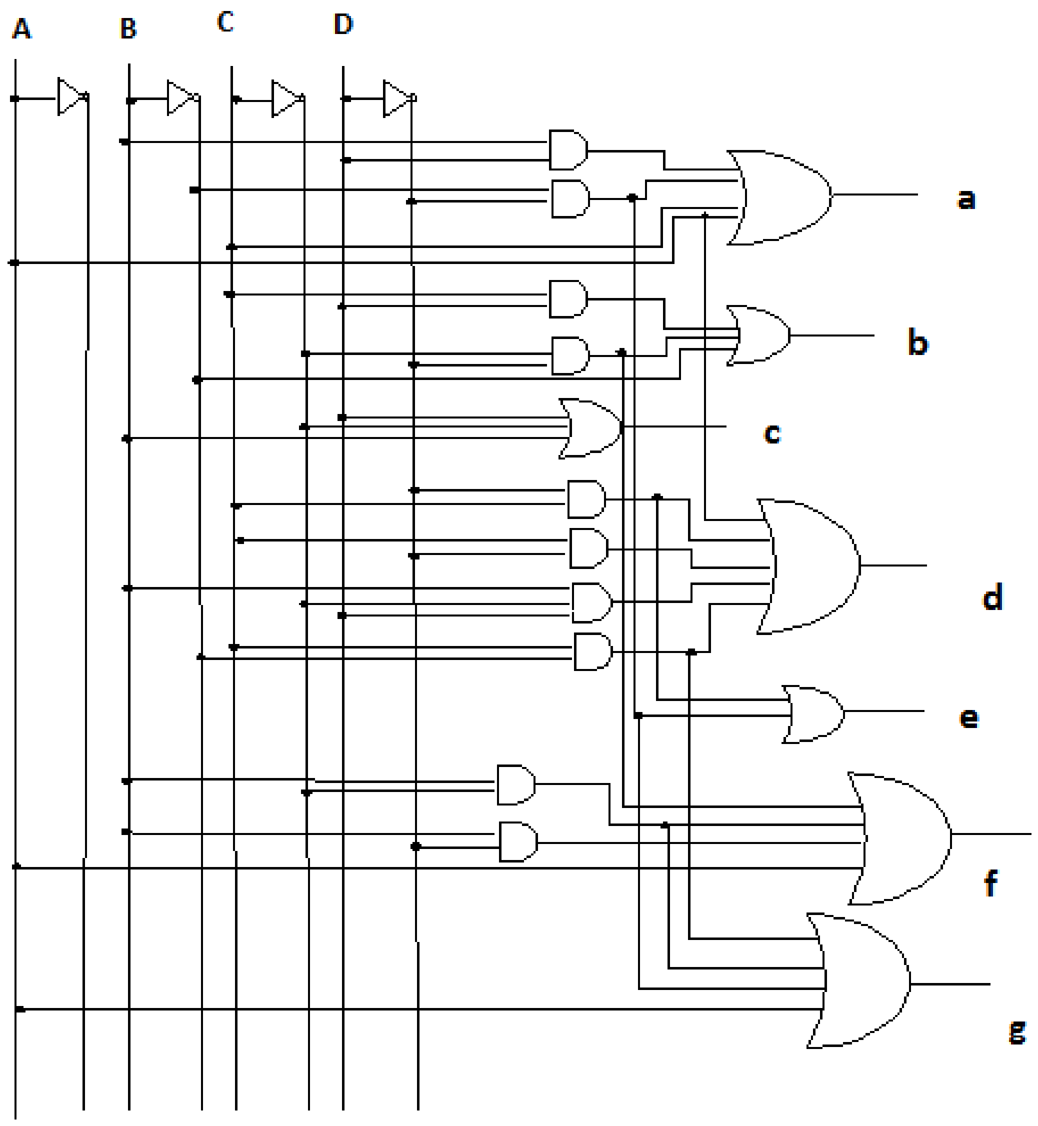


$f = A + \overline{C}\,\overline{D} + B\,\overline{C} + B\,\overline{D}$



$g = \overline{B}\,C + C\,\overline{D} + B\,\overline{C} + B\,\overline{C} + A$

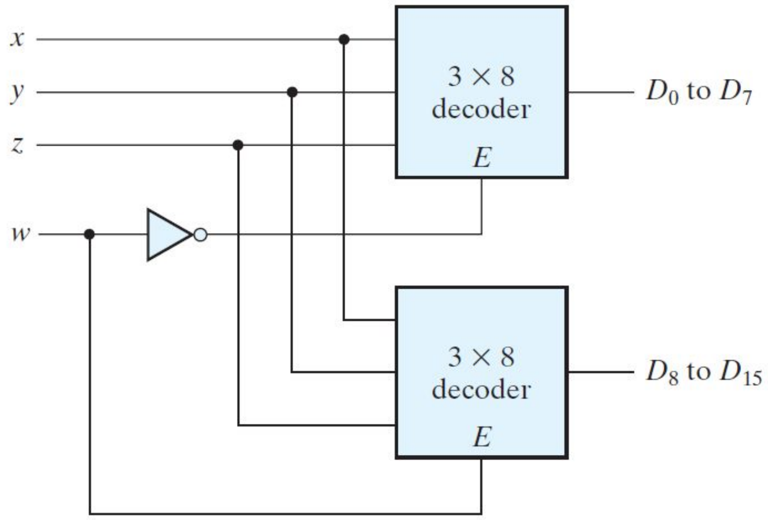
(c) K-Map for Seven segment Display



(f) Logic Circuit of seven segment Display Device

Construction of 4x16 with Two 3x8 Decoders

We can construct a large decoder/demultiplexer circuit by combining two 3×8 decoders to form a 4×16 decoder with an enable input as shown in the fig. When w =0, the top decoder is enabled and the bottom decoder is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0100. When w=1, the top Decoder outputs are all 0's and the bottom decoder outputs generate minterms 1000 to 1111.



Summary

- A combinational circuit consists of logic gates whose output at any time is determined directly from the present combinations of inputs irrespective of previous inputs.
- Multiplexing means transmitting a large numbers of information units over a smaller number of channels or lines.

There are varieties of multiplexers. Some of them are as follows:

- 4:1 multiplexer
- 8:1 multiplexer
- 16:1 multiplexer

- Demultiplexer combinational circuit performs the reverse operations of the multiplexer. In short it is also called as DMUX. There are different types of Demultiplexer based on the output configuration such as 1:4, 1:8, and 1:16.
- 1-to-2 Demultiplexer consists of one input line, two output lines and one select line. The outputs of the demultiplexer depend on the signals provided by the select line to one of the two output lines.

- A1-to-4 has single input I, two selection lines S1 and S0 and four outputs Y0 to Y3. The input data goes to any of the four output lines at a time for particular combinations of selection lines.
- 1:8 demultiplexer consist of single input lineI and three selection lines S0, S1 and S2, and eight output lines from Y0 to Y7. It sends one input line to one of 8 output lines depending on the combinations of three selection lines.
- A code converter is a logic circuit that converts the one form of digital code to another form i.e. it converts BCD code to Binary code, HEX code to binary, etc. Combinational circuit performs this transformation by means of logic gates.
- Decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2n unique output lines. If the n-bit decoded information has unused or don't care combinations, the decoder output will have fewer than 2noutputs.
- A binary decoder is a combinational circuit that converts binary code of n input lines to the one of the 2n output lines depending on the combinations of input lines. It is used when it is necessary to activate exactly one of 2n output lines based on the n input values. The most common decoders are 2-to-4 line, 3-to-8 line and 4-to -16 lines.
- In 2-to-4 binary Decoder two inputs are decoded into four outputs so it is called 2-to-4 line decoder. Only one output is activated at a time while the other outputs are maintained at logic level 0 and the output which is high depends on the combinations of the bit values of two inputs.
- In3-to-8 line Decoder, three inputs are decoded into eight output lines. It has three inputs as X, Y and Z, and eight outputs through (D0 to D7). Based on the combinations of the three inputs only one output is selected.
- In BCD to Decimal decoder, the elements of information are 10 decimal digits represented by the BCD code. The functions of the decoder are to supply one output for each decimal digit.
- Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has 2n input lines and n output lines. The output lines generate the binary code corresponding to the input value.

There are varieties of encoder circuit. They are as follows.

    - Octal to Binary Encoder
    - Decimal to BCD encoder
    - Priority Encoder

- Hexadecimal to Binary encoder
- In octal to binary encoder number system, the base or radix is 8 i.e. r=8 which means it having eight distinct digits and they are from (0 to 7). It consists of eight input lines and three output lines. Each input line corresponds to each octal digit and three outputs generate corresponding binary code.
- The decimal is a number system in which the base or radix is 10, i.e. r=10. So the total number of digit is from (0 to 9). This encoder consists of 10 input lines and four output lines. Each input line corresponds to the each decimal digit and four output lines correspond to the BCD code. It takes the decoded decimal data as an input and encodes it to BCD output to the output lines.
- A priority encoder is an encoder circuit that includes the priority function. It operates in a manner that if two or more inputs are equal to 1 at the same time, then the input line with highest priority will be considered.
- In case of hexadecimal number system the base or radix is 16, i.e. r=16, which means there are in total sixteen distinct digits. They are ranging from 0 to F.
- The seven-segment decoder consists of seven LED's arranged in the rectangular fashion. Each of the seven LED is called segment because when illuminated the segment forms part of the numerical digits both decimal and Hex to be displayed.

Self- Evaluation

(Group A)

Short Answer Questions

1. Consider the following statements
2. A multiplexer
3. Selects one of the several inputs and transmit it to a single output.
4. Routes the data from a single input to one of the many outputs.
5. Converts parallel data into serial data.
6. Is a combinational circuit.

Which of these statements are correct?

(a) 1, 2 and 4 b) 2, 3, and 4 c) 1,3 and 4 d) 1, 2 and 3

1. Obtain the logic diagram of a decimal to BCD encoder.
2. Explain the working of a demultiplexer with the help of an example.
3. Explain the operation of 8:1 multiplexer.
4. Explain the operation of octal to binary encoder.
5. What is a demultiplexer? Discuss the differences between a demultiplexer and a decoder.
6. Implement the following Boolean function with an eight-input digital multiplexer.
7. Y(I,S2,S1,S0)= $\Sigma(2,4,7,10,12,14)$
8. What is a decoder? Draw the logic circuit of a 3 line to 8 line decoder and explain its working.

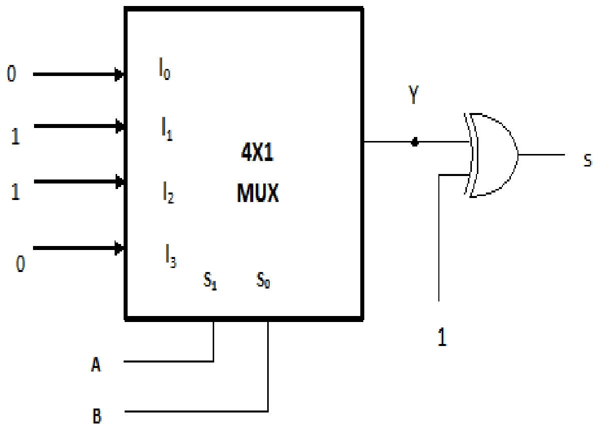
(Group B)

Long Answer Questions

1. Design a combinational circuit whose input is a four bit number and whose output is the 2's complement of the input number.
2. Design a combinational circuit with four input lines that represent a decimal digit in BCD and four output lines that generate the 9's complement of the input digit.
3. Design a combinational circuit that converts a decimal digit from the 8, 4,-2,-1 code to BCD.
4. Design a combinational circuit that converts a decimal digit from the 2,4,2,1, code to the 8, 4,-2,-1.
5. Design a combinational circuit that accepts a three bit number and generate an output equal to the square of the input number.
6. The circuit shown doesn't represent

   a. $S(A,B)=\Sigma(1,2)$ b. Ex-OR gate with A and B as input

   c. $S(A, B) =\pi (0, 3)$ d. $\overline{A}.\overline{B}$ +AB



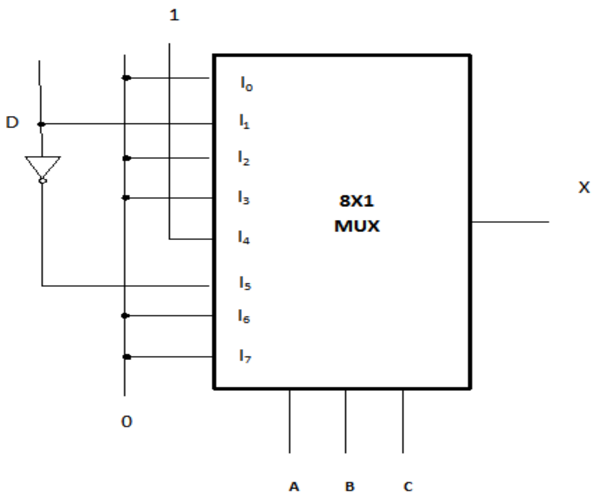
1. Design a combinational circuit with a decoder and external gates by the following Boolean functions.

   F1= $\overline{X}.Y.\overline{Z} + XZ$

   F2=$X.\overline{Y}.\overline{Z} + \overline{X}.Y$

   F3=$\overline{X}.\overline{Y}.\overline{Z} + X.Y$

1. The circuit below represents function X(A,B,C,D) as :
   a. Σ(3,8,9,10)
   b. Σ(3,8,10,14)
   c. п(0,1,2,4,5,6,7,11,12,13,15)
   d. п(0,1,2,4,5,6,7,10,12,13,15)



1. A combinational circuit has four inputs and one output. The output is equal to 1 when:

    1. 1) All the inputs are equal to 1
    2. 2) None of the inputs are equal to 1
    3. 3) An odd number of inputs are 1

Questions:

a) obtain the truth table

b) find the simplified output function in sum of product

c) find the simplified output function in Product of sum.

d) Draw the logic diagram.

1. What is an encoder? Draw the logic circuit of Decimal to BCD encoder and explain its working.
2. What is a Decoder? Compare a decoder and a demultiplexer with suitable block diagrams.
3. What is a digital multiplexer? Illustrate its functional diagram. Write the scheme of a 4- input multiplexer using basic gates (AND/OR/NOT) and explain its operation.
4. Implement the following function using 8 to 1 multiplexer Y(,A ,B ,C D) = $\sum$ (0,1,2,5,9,11,13,15).
5. Implement the following function using 4-to-1 multiplexer. Y (, A, B C) = $\sum$ (2, 3, 5, 6).
6. A combinational circuit has 3 inputs A, B, C and output F. F is true for following input combinations

    A is False, B is true

    A is False, C is true

    A, B, C are False

    A, B, C are True

    (i) Write the Truth table for F. Use the convention True=1 and False = 0.

    (ii) Write the simplified expression for F in SOP form.

    (iii) Write the simplified expression for F in POS form.

    (iv) Draw logic circuit using minimum number of 2-input NAND gates.

UNIT:- 7
Sequential Logic

Learning Outcomes

After completion of this unit you will be able to

- to impart to you a formalism of sequential logic.
- to enable you to understand different type of sequential logic.
- to enable you to implement RS or SR, T,D,JK, JK Master Slaves Flip Flop.
- to enable you to understand the latch,level clocking,trigger and its different types likes low, high level triggering etc.

Introduction:-

Sequential logic circuit consists of combinational circuit to which memory elements are connected to form a feedback path.

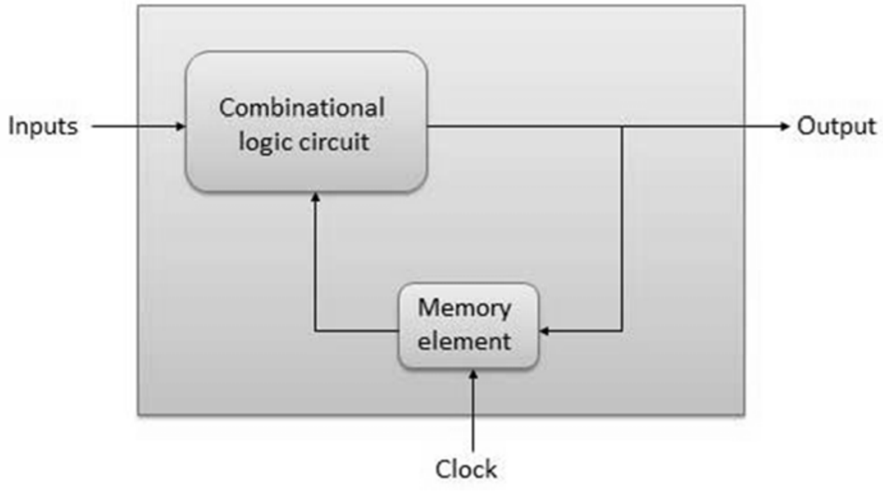


Fig.7.1.1 Block Diagram

The outputs of sequential circuit depends on present inputs and past or previous state. Examples:- Flip flop, register, counters.

Sequential logic is a form of binary circuit design that employs one or more inputs and one or more outputs, whose states are related by defined rules that depend, in part, on previous states. Each of the inputs and output(s) can attain either of two states: logic 0 (low) or logic 1 (high).

A common example of a circuit employing sequential logic is the flip-flop, also called a bistable gate. A simple flip-flop has two stable states. The flip-flop maintains its states indefinitely until an input pulse called a trigger is received. If a trigger is received, the flip-flop outputs change their states according to defined rules, and remain in those states until another trigger is received.

There are different kinds of flip-flop circuits, with designators such as D, T, J-K, and R-S. Flip-flop circuits are interconnected to form the logic gates that comprise digital integrated circuits (IC s) such as memory chips and microprocessors.

Sequential logic differs from combinatorial logic (also called combinational logic). In the latter scheme, the output states depend only on the input states at a specific moment in time, and not on previous states.

Types of sequential logic

1. **Synchronous Sequential Logic**
   In this system signals that affect the memory elements only at discrete instants of time. Synchronous is achieved by a time device called a master clock generation which generates a periodic train of clock pulses. Synchronous sequential circuits that use clock pulses in the input of memory elements are called clocked sequential circuit.

2. **Asynchronous Sequential Logic**
   The behavior of an asynchronous sequential logic system depends upon the order in which its input signals change and can be affected at any instant of time.

3. **Latching Effect or Latch-up:**
   A latch-up is a type of short circuit which can occur in an integrated circuit (IC). More specifically it is the inadvertent creation of a low-impedance path between the power supply rails of a MOSFET circuit, triggering a parasitic structure which disrupts proper functioning of the part, possibly even leading

to its destruction due to over current. A power cycle is required to correct this situation.

A single event latch-up is a latch-up caused by a single event upset, typically heavy ions or protons from cosmic rays or solar flares.

The parasitic structure is usually equivalent to a thyristor (or SCR), a PNPN structure which acts as a PNP and an NPN transistor stacked next to each other. During a latch-up when one of the transistors is conducting, the other one begins conducting too. They both keep each other in saturation for as long as the structure is forward-biased and some current flows through it - which usually means until a power-down. The SCR parasitic structure is formed as a part of the totem-pole PMOS and NMOS transistor pair on the output drivers of the gates.

The latch-up does not have to happen between the power rails - it can happen at any place where the required parasitic structure exists. A common cause of latch-up is a positive or negative voltage spike on an input or output pin of a digital chip that exceeds the rail voltage by more than a diode drop. Another cause is the supply voltage exceeding the absolute maximum rating, often from a transient spike in the power supply. It leads to a breakdown of an internal junction. This frequently happens in circuits which use multiple supply voltages that do not come up in the required sequence on power-up, leading to voltages on data lines exceeding the input rating of parts that have not yet reached a nominal supply voltage. Latch-ups can also be caused by an electrostatic discharge event.

Another common cause of latch-ups is ionizing radiation which makes this a significant issue in electronic products designed for space (or very high-altitude) applications. High-power microwave interference can also trigger latch ups.

Both CMOS integrated circuits and TTL integrated circuits are more susceptible to latch-up at higher temperatures.

A flip-flop is called latch, if the instance at which output should change has no well defined instances clock input. A latch is FF without no edge triggered clocking mechanism for its inputs. A latch may have a gating input (clock input interval) during which input changes affect changes called latching effect.

4. **Level Clocking**

A clock is a control signal that periodically makes a transition from a 0 to 1 and then back to 0 again we usually denote the clock by the symbol clk or cp.

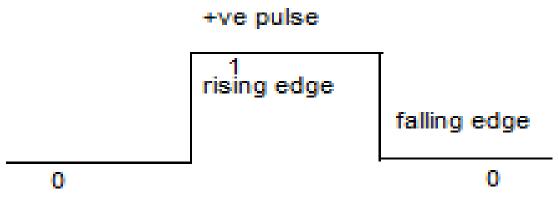


Fig. 7.1.2 Clock

Clock Pulse Transition

The movement of a trigger pulse is always from a 0 to 1 and then 1 to 0 of a signal. Thus it takes two transitions in a single signal. When it moves from 0 to 1 it is called a positive transition and when it moves from 1 to 0 it is called a negative transition. To understand more take a look at the images below.

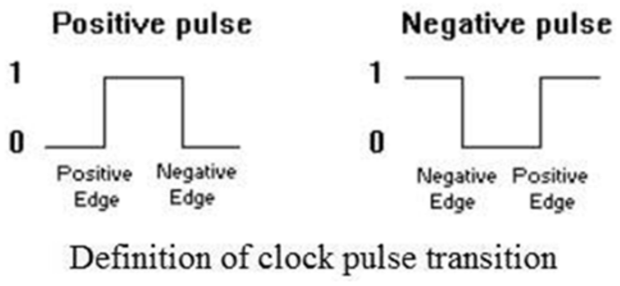


Fig.7.1.2 Clock Pulse Transition

The clocked flip-flops already introduced are triggered during the 0 to 1 transition of the pulse, and the state transition starts as soon as the pulse reaches the HIGH level.

If the other inputs change while the clock is still 1, a new output state may occur. If the flip-flop is made to then the multiple-transition problem can be eliminated.

The multi-transition problem can be stopped is the flip flop is made to respond to the positive or negative edge transition only, other than responding to the entire pulse duration.

Trigger

The number of trigger pulses that is applied to the input of the circuit determines the number in a counter. A single pulse makes the bit move one position, when it is applied onto a register that stores multi-bit data.

In the case of SR Flip Flops, the change in signal level decides the type of trigger that is to be given to the input. But the original level must be regained before giving a second pulse to the circuit.

If a clock pulse is given to the input of the flip flop at the same time when the output of the flip flop is changing, it may cause instability to the circuit. The reason for this instability is the feedback that is given from the output combinational circuit to the memory elements. This problem can be solved to a certain level by making the flip flop more sensitive to the pulse transition rather than the pulse duration.

Types of Trigger

There are mainly four types of pulse-triggering methods. They differ in the manner in which the electronic circuits respond to the pulse. They are

1. High Level Triggering

When a flip-flop is required to respond at its HIGH state, a HIGH level triggering method is used. It is mainly identified from the straight lead from the clock input. Take a look at the symbolic representation shown below.

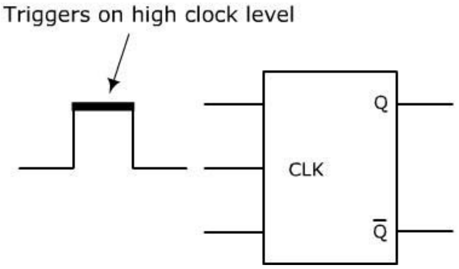


Fig.7.1.3.i. High Level Triggering

2. Low Level Triggering

When a flip-flop is required to respond at its LOW state, a LOW level triggering method is used. It is mainly identified from the clock input lead along with a low state indicator bubble. Take a look at the symbolic representation shown below.

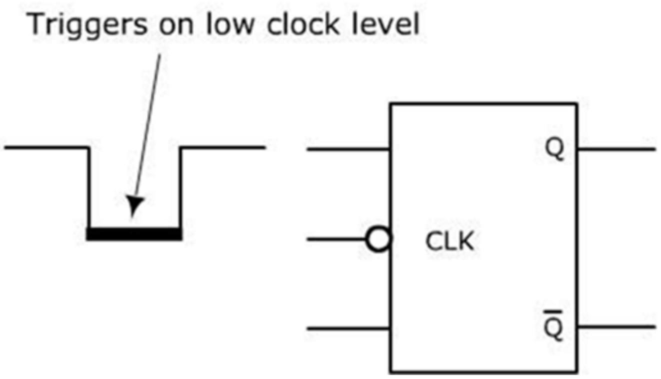
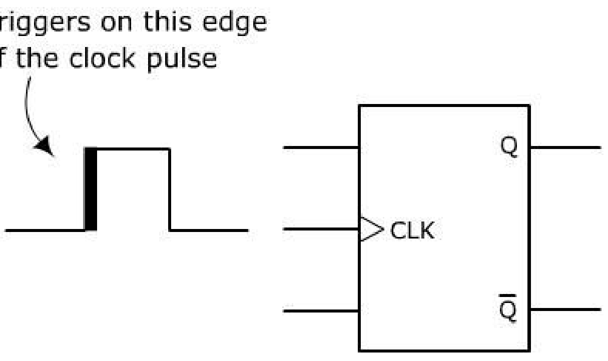


Fig.7.1.3.ii. Low Level Triggering

3. Positive Edge Triggering

When a flip-flop is required to respond at a LOW to HIGH transition state, POSITIVE edge triggering method is used. It is mainly identified from the clock input lead along with a triangle. Take a look at the symbolic representation shown below.



Positive Edge Triggering

Fig.7.1.3.iii. Positive Edge Triggering

4. Negative Edge Triggering

When a flip flop is required to respond during the HIGH to LOW transition state, a NEGATIVE edge triggering method is used. It is mainly identified from the clock input lead along with a low-state indicator and a triangle. Take a look at the symbolic representation shown below.

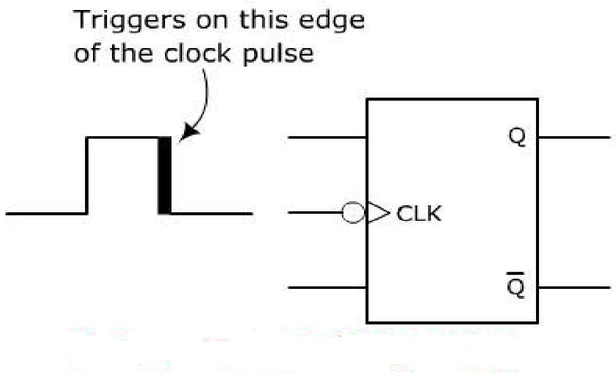


Fig.7.1.3.iv. Negative Edge Triggering

Flip Flop

Flip-flops are binary cells capable of storing one bit information. A flip-flop circuit has two outputs one for the normal value and one for the complement value of the bit stored in it. Binary information can enter in a flip-flop in a variety of ways, a fact which give rise to different types of flip flops.

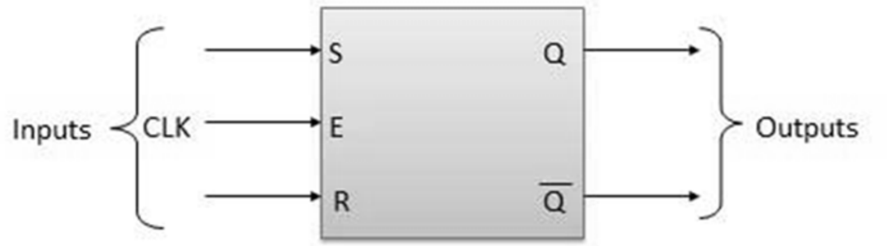
Flip-flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip-flop is said to be edge sensitive or edge triggered rather than being level triggered like latches.

Basic Flip-Flop Circuit

S-R Flip Flop

It is basically S-R latch using NAND gates with an additional enable input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

Block Diagram



Circuit Diagram

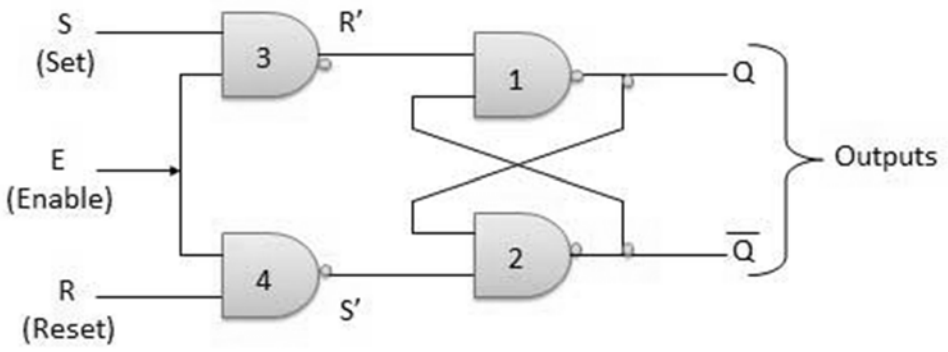
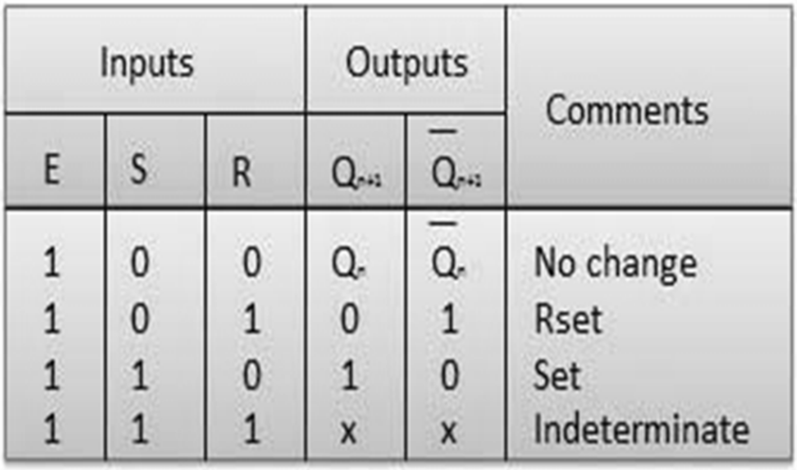


Fig.7.2. SR Flip Flop

Truth Table



Inputs			Outputs		Comments
E	S	R	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	x	x	Indeterminate

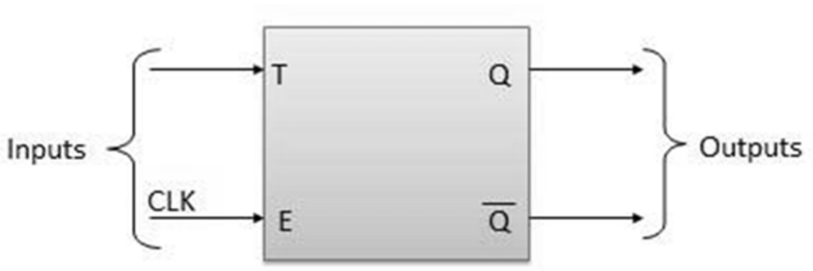
Operation

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

Toggle Flip-Flop / T Flip-Flop

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

Symbol Diagram



Block Diagram

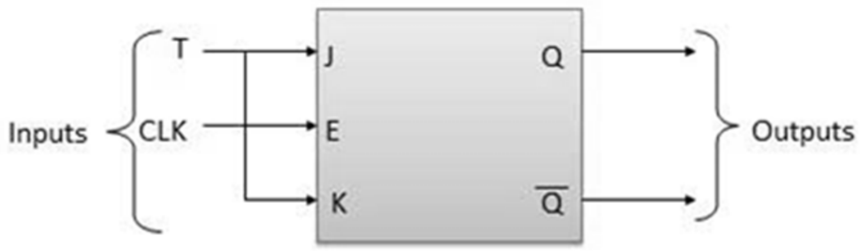
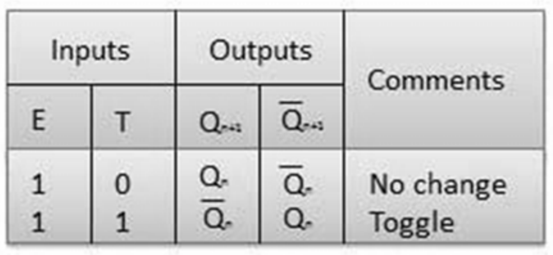


Fig.7.3. T Flip-Flop

Truth Table



Inputs		Outputs		Comments
E	T	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	Q_n	$\overline{Q}_n$	No change
1	1	$\overline{Q}_n$	Q_n	Toggle

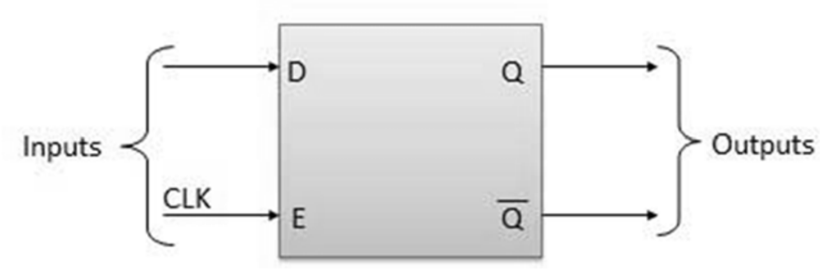
Operation

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

Delay Flip Flop / D Flip-Flop

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

Block Diagram



Circuit Diagram

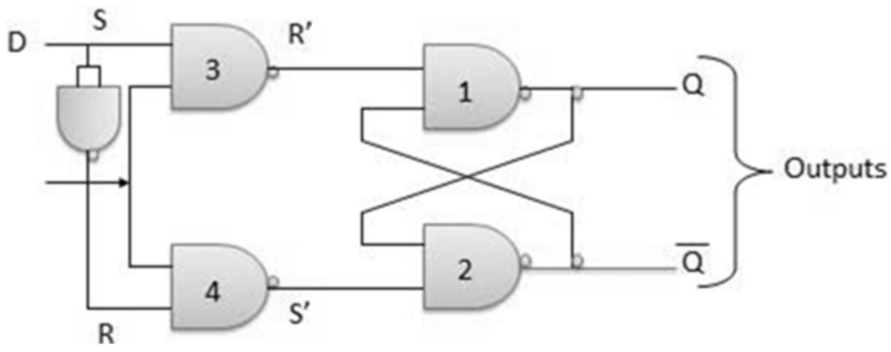
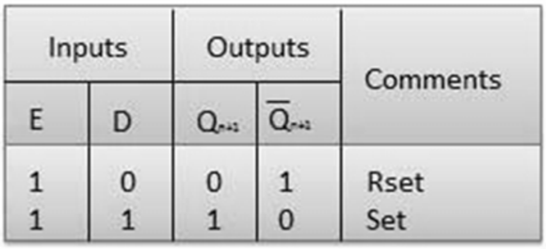


Fig.7.4. D Flip Flop

Truth Table



Inputs		Outputs		Comments
E	D	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	1	Rset
1	1	1	0	Set

Operation

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

JK Flip-flop

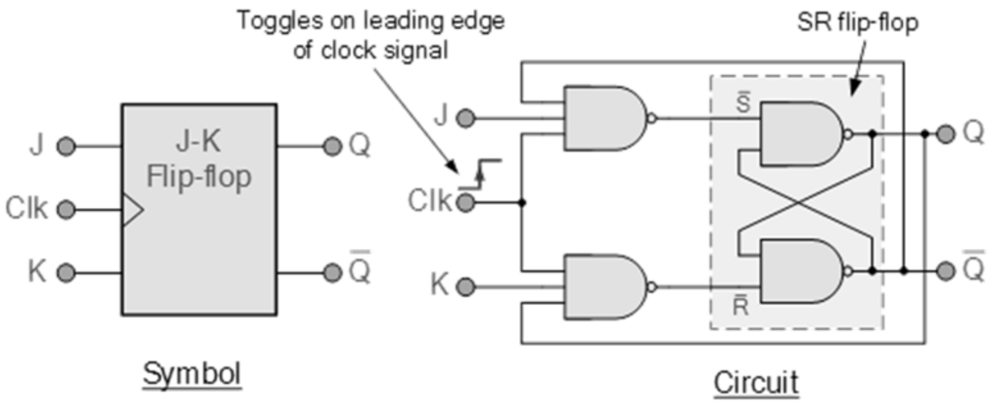


Fig. 7.5 JK Flip-flop

Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack Kilby. Then this equates to: J = S and K = R.

The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked.

If the circuit is now "SET" the J input is inhibited by the "0" status of Q through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q are always different we can use them to control the input. When both inputs J and K are equal to logic "1", the JK flip flop toggles as shown in the following truth table.

The Truth Table for the JK Function

	Input		Output		Description
	J	K	Q	Q	
same as for the SR Latch	0	0	0	0	Memory no change
	0	0	0	1	
	0	1	1	0	Reset Q » 0
	0	1	0	1	
	1	0	0	1	Set Q » 1
	1	0	1	0	
toggle action	1	1	0	1	Toggle
	1	1	1	0	

Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit. Also when both the J and the K inputs are at logic level "1" at the same time, and the clock input is pulsed either "HIGH", the circuit will "toggle" from its SET state to a RESET state, or vice-versa. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are "HIGH".

Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called "race" if the output Q changes state before the timing pulse of the clock input has time to go "OFF". To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC's the much improved Master-Slave JK Flip-flop was developed.

The master-slave flip-flop eliminates all the timing problems by using two SR flip-flops connected together in a series configuration. One flip-flop acts as the "Master" circuit, which triggers on the leading edge of the clock pulse while the other acts as the "Slave" circuit, which triggers on the falling edge of the clock pulse. This results in the two sections, the master section and the slave section being enabled during opposite half-cycles of the clock signal.

Master Slave JK Flip-Flop

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

Circuit Diagram

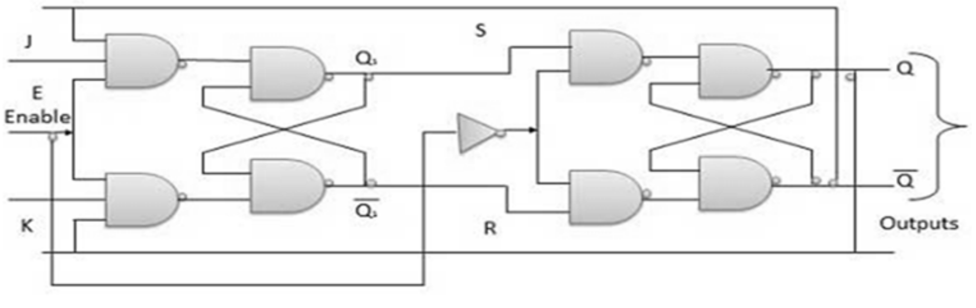
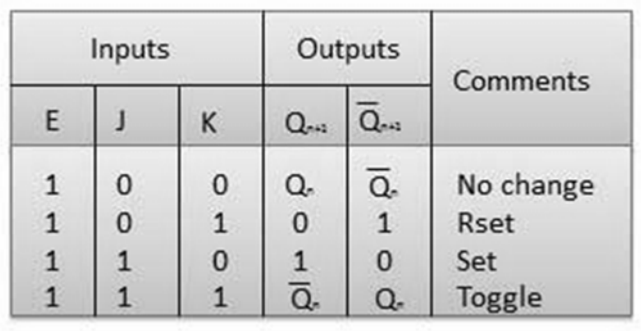


Fig.7.6 Master Slave JK Flip Flop

Truth Table



Inputs			Outputs		Comments
E	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	$\overline{Q}_n$	Q_n	Toggle

Operation

S.N.	Condition	Operation
1	J = K = 0 (No change)	When clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged. Therefore outputs will not change if J = K =0.
2	J = 0 and K = 1 (Reset)	Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q1 = 0 and Q1 bar = 1. That means S = 0 and R =1. Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 0 and Q bar = 1. Again clock = 1 − Master active, slave inactive. Therefore even with the changed outputs Q = 0 and Q bar = 1 fed back to master, its output will be Q1 = 0 and Q1 bar = 1. That means S = 0 and R = 1. Hence with clock = 0 and slave becoming active the outputs of slave will remain Q = 0 and Q bar = 1. Thus we get a stable output from the Master slave.
3	J = 1 and K = 0 (Set)	Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q1 = 1 and Q1 bar = 0. That means S = 1 and R =0.

		Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 1 and Q bar = 0. Again clock = 1 − then it can be shown that the outputs of the slave are stabilized to Q = 1 and Q bar = 0.
4	J = K = 1 (Toggle)	Clock = 1 − Master active, slave inactive. Outputs of master will toggle. So S and R will be inverted. Clock = 0 − Slave active, master inactive. Outputs of slave will toggle. These changed outputs are returned back to the master inputs. But since clock = 0, the master is still inactive. So it does not respond to these changed outputs. This avoids the multiple toggling which leads to the race around condition. The master slave flip flop will avoid the race around condition.

SUMMARY

- Sequential Logic: Sequential logic is a form of binary circuit design that employs one or more inputs and one or more outputs, whose states are related by defined rules that depend, in part, on previous states.
- Synchronous: In this system, signals that affect the memory elements only at discrete instants of time.
- Asynchronous: In this sequential logic system depends upon the order in which its input signals change and can be affected at any instant of time.
- Latch: A latch is a class of flip-flop in place of clock-edge instance at which output changes, a clock interval exists during which that output changes.
- Clock: It is a control signal that periodically makes a transition from a 0 to 1 and then back to 0 again we usually denote the clock by the symbol clk or cp.
- High Level Triggering: When a flip flop is required to respond at its HIGH state, a high level triggering method is used.
- Low Level Triggering: When a flip flop is required to respond at its LOW state, a Low Level Triggering method is used.
- Positive Edge Triggering: When a flip flop is required to respond at a LOW to HIGH transition state, positive edge triggering method is used.

- Negative Edge Triggering: When a flip flop is required to respond during the HIGH to LOW transition state, a NEGATIVE edge triggering method is used.
- Flip-Flop: A flip flop is a device that can store 1 bit of information. There are three kinds of flip flops: D, T, and JK.
- SR Flip-Flop: SR Flip-Flop is an arrangement of logic gates that maintains a stable output even after the inputs are turned off. This simple flip flop circuit has a set input (S) and a reset input (R). The set input causes the output of 0 (top output) and 1 (bottom output). The reset input causes the opposite to happen (top = 1, bottom =0). Once the outputs are established, the wiring of the circuit is maintained until S or R go high, or power is turned off to the circuit.
- D flip-flop: tracks the input, making transitions with match those of the input D. The D stands for "data"; this flip-flop stores the value that is on the data line. It can be thought of as a basic memory cell.
- T Flip-Flop: The T or "toggle" flip-flop changes its output on each clock edge, giving an output which is half the frequency of the signal to the T input.
- J-K Flip-Flop: J-K flip-flop is the most versatile of the basic flip-flops. It has the input- following character of the clocked D flip-flop but has two inputs, traditionally labeled J and K. If J and K are different then the output Q takes the value of J at the next clock edge.

Self-Evaluation

1. What is meant by sequential logic circuits? Draw a block diagram of sequential logic circuits.
2. What is clock? Write its different clock levels.
3. What is trigger? Explain its types.
4. What is flip flop? List basic flip flop circuits.
5. What is SR or RS flip flop? Draw a block diagram, circuit diagram and truth table.
6. What is Toggle (T) flip flop? Draw a block diagram and truth table with operation.
7. What is the use of T flip flop?
8. What is Delay (D) flip flop? Draw a block diagram, circuit diagram and truth table with operation.
9. What is a JK flip flop? Write its symbol diagram and circuit diagram with its operation.
10. What is master slave flip flop? Draw its circuit diagram and truth table with operation.