Mt Kenya University

P.O. Box 342-01000
Thika
Email: mailto:Info@mku.ac.ke
Web: www.mku.ac.ke

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: BIT 1202

COURSE TITLE: BASIC OF DIGITAL ELECTRONICS AND LOGIC

Instructional Material for BBIT - distance learning

COURSE OUTLINE

BIT 1202	**BASIC OF DIGITAL ELECTRONICS AND LOGIC**
Purpose	To introduce students to digital electronics concepts and provide a strong foundation of basic principles through the classical approach before engaging in practical design approach and the use of the digital electronic technology and concepts which is basis for all current computer technology
Objectives	By the end of the course unit, a student should : ▪ Describe and implement digital electronic concepts ▪ Build simple digital devices ▪ Develop a hands-on experience and an understanding of the design of digital circuits and the basic components of a complete computer hardware technology
Course Content	**Week 1 and 2** ▪ Introduction To Digital Systems 1. Decimal Number systems 2. Binary Number systems 3. Octal Number systems 4. Hexadecimal Number systems 5. Arithmetic processes 6. Complement **Week 3 and 4** ▪ Binary codes 1. -8-4-2-1 BCD CODE 2. -Excess 3code 3. -Error Detecting Codes 4. -Gray Code **Week 5and 6** ▪ **Logic gates.** 1. AND Gate 2. NOT Gate. 3. NAND Gate 4. OR GATE. 5. NOR GATE 6. Exclusive – OR.(XOR) 7. Exclusive NOR. 8. Universal Gates 9. Two-level Implementation of Logic Networks **Week 7,8 and 9** ▪ BOOLEAN ALGEBRA. 1. Postulates and theorems 2. Boolean relations. 3. De Morgans theorem 4. Simplifying Expressions Using Boolean Algebra

	5. Standard Forms 6. Karnaugh map reduction technique 7. Don't care conditions 8. Two-level implementations 9. Realization of complement forms 10. Examples of logic networks 11. Introduction to tabulation methods **Week 10,11 and 12** ▪ DIGITAL CIRCUITS - Sequential and combination circuits - combination circuits –Adders, Magnitude Comparator, Subtractor, Multiplexers, Encoders - Combinational Logic With MSI And LSI - Sequential Logic Circuits; Asynchronous Sequential logic circuits, Asynchronos sequential logic design and analysis - flip-flops, triggering techniques, - Registers; ripple and synchronous counters, shift and ring registers - Introductory state machine theory;mealy machines and Moore machine, algorithmic state machine chart, - Applications of digital electronic and logic

Rerences	**Author (and year of publication)**	**Title**
	A shaba & N Maana	Digital principles and logic designs
	Roth C.H. (1992)	Fundamentals of Logic Design
	Yorbrough, J. M (1997)	Digital Logic Applications and Design
	Alan C. (1997)	Microprocessor Systems Design.
	Bartlet Terry (2002)	Digital Electronics
	Dixon, Allan (2000)	A practical Approach to Digital Electronics

MODE OF ASSESSMENT MARKS (%)	*C. A.Ts and Assignments 30* *Final Examination 70* *Total 100* *Pass mark 40*

CHAPTER ONE

INTRODUCTION TO DIGITAL SYSTEMS

Chapter objectives

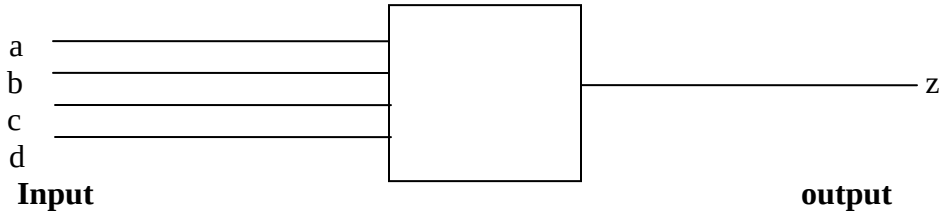
1. explain the digital systems concepts
2. describe the various number systems
3. convert between various number system
4. perform the various number system arithmetic

1.1 Introduction to Digital Systems

Electronic circuits/systems are classified as being analogue or digital, the distinction between the two circuits is not the semicircular material used to construct them but rather the way they are operating, that is Current and voltage variations during performance. Analog electronics deals with things that are continuous in nature and digital electronics deals with things that are discrete in nature. But they are very much interlinked.

By contrast, the digital circuit is the one which voltage levels alternate among a finite number of distance value that is, they are 2 voltage levels high a low. The transmission between high and low and vice versa is too brief and therefore in between voltage value are disregarded

The term linear is used to mean analogue or non-digital, discrete means distinct or having individual identifiable value or elements. Digital circuits produce discrete outputs, distinct circuits have individual components e.g. resistors, and transistors. Hybrids circuits contain both integrated and discrete components. Digital circuits are often called logic circulatory because the level of each output voltage depends on several input voltages and the inputs voltage may appear in many different combinations.



1.2 Number Systems.

Numbers are difficult to define, they are simply a symbolic representation of ideas; number systems are positional in nature and therefore a symbolic of numbers has weights.

1.2 1 Decimal Number System

A digit is a symbol given to an element of a number system. The radix, or base of a counting system is defined as the number of unique digits in a given number system.
Decimal number is represented by arranging 10 symbols by 0-9; there are known as decimal digits. Radix of decimal number system is 10. The position of each digital has a numerical weight and each digital is multiplier of the weight of its position e.g. 10^2 , 10^1 ,10^0 –weights. We represent the radix of decimal number system by putting the radix in subscript to the right of the digits. For example,

3_{10} Represents 3 in decimal (base 10).

When our count exceeds the highest digit available, the next digit to the left is incremented and the original digit is reset to zero. For example:

$$9_{10} + 1_{10} = 10_{10}$$

Because we are dealing with a base-10 system, each digit to the left of another digit is weighted ten times higher. Using exponential notation, we can imagine the number 10 as representing:

$$10_{10} = 1 \times 10^1 + 0 \times 10^0$$

$$314_{10} = 3 \times 10^2 + 1 \times 10^1 + 4 \times 10^0$$

1.22 Binary Number Systems

Since a digital circuit has two values that is. ON & OFF or TRUE& FALSE then the binary number systems is the best used in circulatory; it has two values 0 and 1. The base (radix) is 2. The base of the number system is equal to the number of system in those systems**.**

Counting

Similar to decimals, binary digits have a positional weight. Each bit is weighted twice as much as the bit to the right of it

$$0110_2 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

1.23 Basic Binary Arithmetic

Binary addition, subtraction, multiplication and division operations work essentially the same as they do for decimals.
For addition, you add equally-weighted bits, much like decimal addition (where you add equally-weighted digits) and carry as required to the left.

$$\begin{array}{rrr} & & 0100 \\ 0100_2 + 0111_2 \Rightarrow & + & \underline{0111} \\ & = & 1011 \end{array}$$

As you can see, a carry is generated in the 2^2column which increments the 2^3column.

Subtraction works just like decimal arithmetic, using borrowing as required.

$$
\begin{array}{rcr}
 & & 1011 \\
1011_2 - 0111_2 \Rightarrow & - & \underline{0111} \\
 & = & 0100
\end{array}
$$

Here, a borrow is required, reducing the 2^3column to $0 - 0 = 0$and changing the 2^2column to $2 - 1 = 1$.

Multiplication is straightforward also:

$$
\begin{array}{rcr}
 & & 1011 \\
 & \times & \underline{\quad 0111} \\
 & & 1011 \\
1011_2 \times 0111_2 \Rightarrow & & 10110 \\
 & & \underline{101100} \\
 & = & 1001101
\end{array}
$$

Binary Division

Binary division follows the same procedure as decimal division. The rules regarding binary division are listed in table below

Table 1.4

Dividend	*Divisor*	*Result*
0	0	Not allowed
0	1	0
1	0	Not allowed
1	1	1

Example 1.31. *Divide the following binary numbers:*

(a) 11001 and 101 and (b) 11110 and 1001.

Solution.

(a) 11001 ÷ 101

```
                    1   0   1
            ---------------------
1   0   1 | 1   1   0   0   1
            1   0   1
            ---------
            0   0   1   0   1
                    1   0   1
                    ---------
                    0   0   0
```

Answer: 101

(b) 11110 ÷ 1001

```
                    1   1.  0   1   0
            ---------------------------
1   0   0   1 | 1   1   1   1   0
                1   0   0   1
                -------------
            0   1   1   0   0
                1   0   0   1
                -------------
```

1.2 4 Conversion between binary and other number system

When you consider a binary number in exponential form, you can easily perform a decimal conversion:

Binary to decimal

Example 1

Convert 10110_2 into a decimal number.

Solution.

The binary number given is 1 0 1 1 0

Positional weights *4 3 2 1 0*

The positional weights for each of the digits are written in italics below each digit.

Hence the decimal equivalent number is given as:

$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$= 16 + 0 + 4 + 2 + 0$

$= 22_{10}$.

Example 2

Convert 0110_2 into a decimal number.

Solution.

The binary number given is 0 1 1 0

Positional weights *3 2 1 0*

$$0110_2 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

...simply add up the factors.

$$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 0 + 4 + 2 + 0 = 6$$

Decimal to binary

To convert from decimal to binary, you can repeatedly divide the decimal by two until the result of the division is zero. Starting from the rightmost bit, write 1 if the division has a remainder, zero if it does not. For example, to convert the decimal 74 into binary:

* 74 / 2 = 37 remainder 0; -> 0
* 37 / 2 = 18 remainder 1; -> 10
* 18 / 2 = 9 remainder 0; -> 010
* 9 / 2 = 4 remainder 1; -> 1010
* 4 / 2 = 1 remainder 0; -> 01010
* 2 / 2 = 1 remainder 0; -> 001010
* 1 / 2 = 0 remainder 1; -> 1001010

$$1001010_2 = 1 \times 2^6 + 1 \times 2^3 + 1 \times 2^1 = 64 + 8 + 2 = 74$$

1.2 5 Octal Number System

The name octal implies eight; octal number system has a radix of eight, and uses the following octal digits:

1, 2, 3, 4, 5, 6, 7. An octal number has the subscript 8.

1.26 Conversion between octal and other number system

Each octal digit is representable by exactly three bits. This becomes obvious when you consider that the highest octal digit is seven, which can be represented in binary by 111_2

To convert a binary number to octal, group the bits in groups of three starting from the rightmost bit and convert each triplet to its octal equivalent.

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

$$1000100111101_2 = (100\ 010\ 011\ 101)_2 = (4\ 2\ 3\ 5)_8 = 4235_8$$

To convert an octal number to binary, simply write the equivalent bits for each octal number.

276_8 .

2 7 6

010 111 110

$=010111110_2$

14_8

1 4

001 100

$=001100_2$

$$752_8 = (111\ 101\ 010)_2 = 111101010_2$$

Conversion from decimal to octal

The repeated-division method described for binary will also work for octal, simply by changing the divisor to eight. To convert 67 (base 10 into octal):

* 67 / 8 = 8 remainder 3; -> 3
* 8 / 8 = 1 remainder 0; -> 03
* 1 / 8 = 0 remainder 1; -> 103

1.27 Hexadecimal number system

The most commonly-used number system in computer systems is the hexadecimal, or more simply hex, system. It has a radix of 16, and uses the numbers zero through nine, as well as A through F as its digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.28 Conversion between hexadecimal and other number system

Each hex digit is represent able by exactly four bits. This becomes obvious when you consider that the highest hex digit represents fifteen, which can be represented in binary by 1111.

To convert a binary number to hex, group the bits in groups of four starting from the rightmost bit and convert each group to its hex equivalent.

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

$$10001111_2 = (1000\ 1111)_2 = (8\ F)_{16} = 8F_{16}$$

To convert an hex number to binary, simply write the equivalent bits for each hex number.

Example 1
AC9.E1

A	C	9	E	1
1010	**1100**	**1001**	**1110**	**0001**

Conversion from an Octal to Hexadecimal Number and Vice Versa
Conversion from octal to hexadecimal and vice versa is sometimes required. To convert an octal number into a hexadecimal number the following steps are to be followed:

i. First convert the octal number to its binary equivalent
ii. Then form groups of 4 bits, starting from the LSB.
iii. Then write the equivalent hexadecimal number for each group of 4 bits.

Similarly, for converting a hexadecimal number into an octal number the following steps are to be followed:

i. First convert the hexadecimal number to its binary equivalent.
ii. Then form groups of 3 bits, starting from the LSB.
iii. Then write the equivalent octal number for each group of 3 bits.

Example
Convert the following hexadecimal numbers into equivalent octal numbers.

a) A72E
b) 4.BF85

Solution.

a) Given hexadecimal number is A 7 2 E

Binary equivalent is 1010 0111 0010 1110
= 1010011100101110
Forming groups of 3 bits from the LSB
001 010 011 100 101 110
Octal equivalent 1 2 3 4 5 6
Hence the octal equivalent of $(A72E)_{16}$ is $(123456)_8$.

b) Given hexadecimal number is 4 B F 8 5
Binary equivalent is 0100 1011 1111 1000 0101
= 0100.1011111110000101
Forming groups of 3 bits
100. 101 111 111 000 010 100
Octal equivalent 4 5 7 7 0 2 4
Hence the octal equivalent of (4.BF85)16 is (4.577024)8.

1.29 Fractional Conversion

If the number contains the fractional part we have to deal in a different way when converting the number from a different number system (i.e., binary, octal, or hexadecimal) to a decimal number system or vice versa. We illustrate this with examples.

Example 1 Convert 1010.011_2 into a decimal number.
Solution. The binary number given is 1 0 1 0. 0 1 1
Positional weights 3 2 1 0 -1-2-3
The positional weights for each of the digits are written in italics below each digit.
Hence the decimal equivalent number is given as:
$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$
= 8 + 0 + 2 + 0 + 0 + 0.25 + 0.125
= 10.37510.

Example 2 Convert 362.35_8 into a decimal number.
Solution. The octal number given is 3 6 2. 3 5
Positional weights 2 1 0 -1-2
The positional weights for each of the digits are written in italics below each digit.
Hence the decimal equivalent number is given as:
$3 \times 8^2 + 6 \times 8^1 + 2 \times 8^0 + 3 \times 8^{-1} + 5 \times 8^{-2}$
= 192 + 48 + 2 + 0.375 + 0.078125
= 242.45312510.

Example 3 Convert $42A.12_{16}$ into a decimal number.
Solution. The hexadecimal number given is 4 2 A. 1 2
Positional weights 2 1 0 -1-2
The positional weights for each of the digits are written in italics below each digit.
Hence the decimal equivalent number is given as:
$4 \times 16^2 + 2 \times 16^1 + 10 \times 16^0 + 1 \times 16^{-1} + 1 \times 16^{-2}$
= 1024 + 32 + 10 + 0.0625 + 0.00390625
= 1066.0664062510

1.3 Complement

In computers subtraction is accomplished by performing addition operation .when the difference x-y is required, the computer forms the complements of the subtracted y and adds that to x. the complement of y is the number that is added to another number and produces the difference x-y. There are many kinds of complements used and depending of the kind used, there many may be other operations which are necessary to obtain the two difference of x-y.

1.31 Decimal Compliment

Although arithmetical operations are performed in computers using binary numbers, the kind of hybrid decimal arithmetical involving binary coded Decimal code is encountered.

9's compliment of 150 999
150
849

9's compliment of 150 = 849

9's compliment of 150.45
999.99
150.45
849.54

9's compliment of 150.45 =849.54

Example 1
762
-143

9's compliment of 143 is 999
-143
856

762
+856
1618
+1
619

Example 1
505
-70

9's compliment of 70 is 999
70
929

```
  505
+929
(1)434
   +1
  435
```

Example 3

```
 809.15
-750.10
```

9's compliment of 809.15 is 999.99

```
 -750.10
  249.89
```

```
  809.15
+ 249.89
(1)059.04
    +1
   59.05
```

10's Compliment

10's compliment of decimal number is found by adding 1 to the least significant digital of the 9's compliment

10's compliment of 150 = 999

```
 -150
  849
```

```
  849
  +1
 850
```

10's compliment of 150 = 850

When subtracting x-y using 10's compliment add 10's compliment of y to x. if any carry is generated in the most significant position its carried /ignored.

Example 1

```
 73
-34
```

9's compliment of 34 is 99

```
 -34
```

65

10's compliment of 34 is 65+1=66

73
+ 66
(1)39 =39

Ignore

Example 2

105.20
-28.96

9's compliment of 28.96= 999.99
28.96
971.03

10's compliment of 28.96=971.03
+ 1
971.04

105.20
+971.04
(1)076.24

= 76.24

1.32 Binary compliment. (one's & two's compliment)

We have 1's complement and 2's complement. 1's complement of a binary number is got by subtracting each bit from one or changing every 1 to 0 and every 0 to a 1.

Example 1

101101
-100011

1's complement of 100011 = 111111
-100011
011100

101101
+011100
(1)001001
+1
001010 =001010

2's Compliment.

To obtain two's compliment, add one to the 1's compliment.

Example 1

101101
-100011

1's complement of 100011 = 011100
2's complement of 100011= 011100
+ 1
011101

101101
+ 011101
1 001010= 001010

Discard

Example 2

1011011.11
-1011100.01

1's complement of 1011100.01=0100011.10
2's complement of 1011100.01=0100011.11

1011011.11
+ 0100011.11
1111111.10

=1111111.10

Changing the bit to its opposite is known as complementing a bit

OCTAL SUBRACTION (7' s and 8's)

Example 1

62.5_8
-13.4_8

7's complement of 13.4_8 = 77.7
13.4
64.3_8

62.5_8	5+3=8	8-8=0 carry 1
$+ 64.3_8$	2+4+1(add the carry)	
147.0	6+6=12	12-8=4 carry 1
+1		
47.1	=47.1	

Example 2 using 8's complement

62.5_8

-13.4_8

8's complement of 13.4_8 =77.7

13.4

64.3_8

+1

64.4

62.5_8

$+ 64.4_8$

1 47.1 =47.1

Discard

1.4 Signed Numbers

The most significant bit or position can be interpreted as signed bit indicating when the number representing by the remaining bit is negative or positive .sign bit '0' represents positive and sign bit 1 negative or positive.

Example

$01001011 = +75_{10}$

$11001011 = -75_{10}$

1 1 0 0 1 0 11

Sign magnitude

Computers are designed to perform arithmetical operation in either in is 1's or 2's compliments. Most modern computers are designed for the 2's complent operation. In both kinds of operation 0's or 1's followed by the magnitude bits represents positive and negative numbered respectively.

1.5 Chapter Review Questions

1. Convert the following binary numbers to decimal:

   $i)1001_2, ii)11001010_2, iii)101011111$

2. Convert the following decimal numbers to binary:

   $i)102_{10}, ii)432_{10}, iii)7654$

3. Add the following numbers:
   $i)1010_2 + 1100_2$
   $ii)165_8 + 544_8$
4. Convert the following decimal numbers to binary: 12.345, 103, 45.778, and 9981.
5. Convert the following binary numbers to decimal: 11110001, 00101101, 1010001, and 1001110.
6. Perform the subtractions with the following binary numbers using (a) 1's complement and (b) 2's complement. Check the answer by straight binary subtractions.
   i. 10011 – 10001,
   ii. 10110 – 11000, and
   iii. 100111011 – 10001.
7. Perform the subtractions with the following decimal numbers using (a) 9's complement and (b) 10's complement. Check the answer by straight subtractions.
   i. 1045 – 567,
   ii. 4587 – 5668, and
   iii. 763 – 10001.
8. Explain how division and multiplication can be performed in digital systems

CHAPTER TWO

BINARY CODES

Chapter objectives

1. describe the different binary codes
2. convert between different code
3. perform the various binary codes arithmetic

2.1 Binary Codes

Digital data is encoded whereby groups of bit called bit patterns are used represent both numbers and letters the alphabet as well as special charatetics and control functions (functional keys) the equivalent decimal value of the sequence of but many or may not bear there emulation of the data the data it represents .binary codes are classified into two categories.

a) Numeric
b) Alphanumeric

Numeric codes used to represent numbers. They have special characteristic that facilitate mathematical operations e.g. the sign bit to indicate algebraic sign. Alphanumeric codes are used to represent characteristics, which are alphanumeric letters and numerals. In these codes, a numeral is treated as any other symbol rather than a number or numeral value. Alphanumeric codes are used to transmit data e.g. from the keyboard to the CPU. Numerals transmitted in alphanumeric codes are then converted to numeric codes so mathematical operations can be performed .the numeric results can then be converted to alphanumeric form and then retransmitted into an output display unit e.g. VDU. Data in the computer memory in stored both in numeric and alphanumeric form.

2.1 2.8-4-2-1 BCD CODE

This is the numeric code in which each digit of the decimal number is represented by a separate group of bit. The 8-4-2-1 BCD code represents decimal number 0-9 using 4 bit binary number.

	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

The bit patterns 1010,1011,1100,1 1101, 1110, 1111, are not allowed in BCD codes (10-15) and therefore any BCD operation producing this pattern must be corrected

8-84-2-1 BCD ARITHEMETIC

In 8-4-2-1 BCD arithmetic, add each binary number corresponding to each digit.

For example 24 » 0010 0100
+34 +0011 0100
0101 1000

0101 1000
5 8
=58

If the sum of digits is greater than 9, then there is a problem because one or more of the disallowed bit patterns will result.

For example 25 » 0010 0101
+38 +0011 1000
0101 1101

0101 1101
invalid

It's also possible that the sum of the 2 bit digital number will produce a 5 bit digit number.

For example 8 » 1000
+9 +1001
10001

10001
Not allowed

RULES TO SOLVE THE ABOVE INVALID OPERATIONS

If the sum of the 2 BCD digital is greater than 9, that is 1000 then add 6 , that is 0110 and then propagate the carry to the next significant sum bit. The carry is added to the next most significant digit of that sum. If the bits are 5 digit number or it results from collection the add 6 (0110)

25 =01011101
+38

=01011101
+0110
01100011

0110 0011
6 3 **=63**

8 = 10001
+9

10001
+0110
10111

0001 0111 =17
1 7

Questions

172	579
+429	+299

2.2 Excess 3code

	8	**4**	**2**	**1**	**excess-3**			
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

The excess 3 code is obtained by adding a 3 to the natural BCD code. This code is self complementing meaning that; the 1's complement of the coded number ythat islds the 9's complement of the number itself.
For example

	BCD	excess-3	1's complement
2	0010	0101	1010

9's complement of 2 → 7

The two complement of the coded number also ythat islds the 10's complement of the number itself.

For example

	BCD	Excess-3	1's complement	2's complement
2	0010	0101	1010	1011

10's complement of 2 → 8

Questions

Write the excess-3 code of 492_{10} and its 9's complement.

EXCESS (3) ARITHMETIC

Each excess-3 code group has a value that is greater than the decimal digital it represent and therefore the sum of 2 excess-3 code group will have a value that is 6 greater than the sum of the two decimal group it represents. For example;

3		6
+4	»	+7
7		13

In each excess-3 sum must be corrected by subtracting 3(0011) if a carry in generated when adding 2 excess-3 code group, this is an indication that an invalid code group has been generated and in this case add 3 to correct that. The carry is propagated to the next most significant code group of the sum which in corrected in the same way. That is. by adding a 3 if a carry in generated and by subtracting a 3 if no carry

Example1 . Convert $(367)_{10}$ into its Excess-3 code.

Solution. The decimal number is	3	6	7
Add 3 to each bit	+3	+3	+3
Sum	6	9	10

Converting the above sum into 4-bit binary equivalent, we have a

4-bit binary equivalent of	0110	1001	1010

Hence, the Excess-3 code for $(367)_{10}$ = 0110 1001 1010

Example2 Convert $(58.43)_{10}$ into its Excess-3 code.

Soluton. The decimal number is	5	8	4	3
Add 3 to each bit	+3	+3	+3	+3
Sum	8	11	7	6

Converting the above sum into 4-bit binary equivalent, we have a

4-bit binary equivalent of	1000	1011	0111	0110

Hence, the Excess-3 code for $(58.43)_{10}$ = 10001011.01110110

2.3 Error Detecting Codes

Binary data is transmitted and processed inform of electrical signals which are susceptible to noise that can alter /distort its content. A one can be changes to a zero or a zero to a one.

Parity

To detect the above errors caused by novice, one or more additional bits are often added. The most common of this is a parity which signifies whether the total number of 1's in a code group is odd or even.

In an add parity system ,the parity bit is made 0 or 1 as necessary to make the total number of 1's odd that is including the parity bit.

When the digital data is received, a parity checking circuit generates an error signal. If the total number of 1's is odd in an even parity system or if it's even in an odd parity systems, then there is an error.

This parity check always detects a single error that is 1 bit changed from zero to 1 or from1 to zero. But may not detect 2 or more errors. The odd parity is used more often than the even because even parity does not detect a situation where all zeros are created, due to a short circuit or a fault condition.

Block parity is used to take care of multiple errors. In this case several binary words are transmitted and this correction of bits is regarded as block of data

Example

Odd parity system

Data	parity column
110010	0
001001	1
101011	1
000111	0
101001	0
110110	1
011110	1
011010	0
110011	1

2.4 Gray Code

Gray code belongs to a class of code known as minimum change code, in which a number changes by only one bit as it proceeds from one number to the next. Hence this code is not useful for arithmetic operations. This code finds extensive use for shaft encoders, in some types of analog-to-digital converters, etc. Gray code is reflected code and is shown in Table below. The Gray code may contain any number of bits. Here we take the example of 4-bit Gray code. The code shown in the Table is only one of many such possible codes. To obtain a different reflected code, one can start with any bit combination and proceed to obtain the next bit combination by changing only one bit from 0 to 1 or 1 to 0 in any desired random fashion, as long as two numbers do not have identical code assignments. The Gray code is not a weighted code.

Table 2.2 Four-bit refl ected code

Refl ected Code	*Decimal Equivalent*
m4 0000	0
m3 0001	1
0011	2
m2 0010	3
0110	4
0111	5
0101	6
m1 0100	7
1100	8
1101	9
1111	10
m5 1110	11
1010	12
m6 1011	13
m7 1001	14
1000	15

Conversion of a Binary Number into Gray Code

Any binary number can be converted into equivalent Gray code by the following steps:

i. The MSB of the Gray code is the same as the MSB of the binary number;

ii. The second bit next to the MSB of the Gray code equals the Ex-OR of the MSB and second bit of the binary number; it will be 0 if there are same binary bits or it will be 1 for different binary bits;

iii. The third bit for Gray code equals the exclusive-OR of the second and third bits of the binary number, and similarly all the next lower order bits follow the same mechanism.

Example 1 Convert $(101011)_2$ into Gray code.

Solution.

Step 1.The MSB of the Gray code is the same as the MSB of the binary number.

1 0 1 0 1 1 Binary

↓

1 Gray

Step 2.Perform the ex-OR between the MSB and the second bit of the binary. The result is 1, which is the second bit of the Gray code.

1 ⊕ 0 1 0 1 1 Binary

↓

1 1 Gray

Step 3.Perform the ex-OR between the second and the third bits of the binary. The result is 1, which is the third bit of the Gray code.

1 0 ⊕ 1 0 1 1 Binary

↓

1 1 1 Gray

Step 4.Perform the ex-OR between the third and the fourth bits of the binary. The result is 1, which is the fourth bit of the Gray code.

1 0 1 ⊕ 0 1 1 Binary

↓

1 1 1 1 Gray

Step 5.Perform the ex-OR between the fourth and the fifth bits of the binary. The result is 1, which is the fifth bit of the Gray code.

1 0 1 0 ⊕ 1 1 Binary

↓

1 1 1 1 1 Gray

Step 6.Perform the ex-OR between the fifth and the sixth bits of the binary. The result is 0, which is the last bit of the Gray code.

1 0 1 0 1 ⊕ 1 Binary

↓

1 1 1 1 1 0 Gray

After completing the conversion the Gray code of binary 101011 is **111110**

Conversion of Gray Code into a Binary Number

Any Gray code can be converted into an equivalent binary number by the following steps:

i. The MSB of the binary number is the same as the MSB of the Gray code;
ii. The second bit next to the MSB of the binary number equals the Ex-OR of the MSB of the binary number and second bit of the Gray code; it will be 0 if there are same binary bits or it will be 1 for different binary bits;
iii. The third bit for the binary number equals the exclusive-OR of the second bit of the binary number and third bit of the Gray code, and similarly all the next lower order bits follow the same mechanism.

Example 1 Convert the Gray code 101101 into a binary number.

Solution.

Step 1. The MSB of the binary number is the same as the MSB of the Gray code.

1 0 1 1 0 1 Gray
↓
1 Binary

Step 2.Perform the ex-OR between the MSB of the binary number and the second bit of the Gray code. The result is 1, which is the second bit of the binary number.

1 0 1 1 0 1 Gray
↓
1 → 1 Binary

Step 3.Perform the ex-OR between the second bit of the binary number and the third bit of the Gray code. The result is 0, which is the third bit of the binary number.

1 0 1 1 0 1 Gray
↓
1 1 → 0 Binary

Step 4.Perform the ex-OR between the third bit of the binary number and the fourth bit of the Gray code. The result is 1, which is the fourth bit of the binary number.

1 0 1 1 0 1 Gray
↓
1 1 0 → 1 Binary

Step 5.Perform the ex-OR between the fourth bit of the binary number and the fifth bit of the Gray code. The result is 1, which is the fifth bit of the binary number.

1 0 1 1 0 1 Gray
↓
1 1 0 1 → 1 Binary

Step 6.Perform the ex-OR between the fifth bit of the binary number and the sixth bit of the Gray code. The result is 0, which is the last bit of the binary number.

1 0 1 1 0 1 Gray
↓
1 1 0 1 1 → 0 Binary

After completing the conversion, the binary number of the Gray code 101101 is 110110

2.5 Chapter Review Questions

1. Express the following decimal numbers in Excess-3 code form:
   i. 245,
   ii. 739,
   iii. 4567, and
   iv. 532.
2. Express the following Excess-3 codes as decimals:
   i. 100000110110,
   ii. 0111110010010110, and
   iii. 110010100011.
3. Convert the following binary numbers to Gray codes:
   i. 10110,
   ii. 1110111,
   iii. 101010001,
   iv. 1001110001110.
4. Express the following decimals in Gray code form:
   i. 5,
   ii. 27,
   iii. 567, and
   iv. 89345.
5. Express the following decimals in (1) 2,4,2,1 code and (2) 8, 4, –2, –1 code form:
   i. 35,
   ii. 7,
   iii. 566,
   iv. 8945.
6. Why is Gray code called the reflected code? Explain.
7. Explain with an example how BCD addition is carried out?

CHAPTER THREE

LOGIC GATES

Chapter objectives

1. Describe the various types of logic gates and their symbols
2. Derive logic expressions from logic diagrams
3. Draw logic diagrams from logic expressions
4. Construct universal gates
5. Explain the two level implementation of logic networks

3.1 Logic Gates.

They are the fundamental building blocks of digital systems. Logic gate is a device which has the ability to make decisions based on input values levels or rather a device that produces one input level when some combinations of input levels are present. And a different output level when other input combinations are present. They are electronic devices constructed in wide variety of forms. Many are embedded in IC with a large number of devices. They are not accessible or identifiable. They are also constructed in small scale IC's. The inputs and outputs are accessible and therefore external connections can be made to them. Gates can be interconnected to perform a variety of logic operations. Accessible logic gates are called **discreet logic gates** while else inaccessible very large IC embedded logic gates are **dedicated** to specific logic operation. The discreet logic gate interfaces a large IC and very large Scale IC.

3.1 1 AND GATE.

The digital levels produced by digital circuits are referred variously as: True & false, ON and OFF, High and Low and I & O. AND Gate are a device whose output is one if and only if all its inputs are one. In Logic gates, the inputs or outputs logic gates is a one or a zero. There are no in between. Thus the output of the AND Gate is 0 if any or one more of its inputs is zero. AND Gates have only one output but can have two or more inputs. The inputs are regarded as digital variables which can be assigned symbols of the alphabet. For example a, b, c, d,…..

A list of all possible input combinations and their outputs is called truth table. A two input AND gate will have the following **truth table**.

A	B	Output(Q)
0	0	0
0	1	0
1	0	0
1	1	1

The logical AND Gate function of two or more variables is expressed symbolically as the same manner as Multiplication in Algebra either by using parenthesis or a dot or simply writing the variables side by side. Thus the logical AND functions of two variables A and B can be expressed as

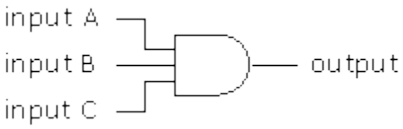
Y = (A)(B)
=AB
=A.B

The logic symbol of the AND Gate (ANSI) is

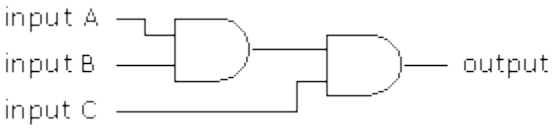
Question

1. Draw the logic symbol of a 3 input gate and tabulate the truth table.

Solution
Logic symbol



Or



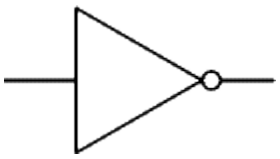
Truth table

input A	*input B*	*input C*	*Output Q*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

A 3-input AND gate can be made by joining, or cascading, 2-input gates:

3.1 2 NOT GATE.

It has only one input and one output. The input is logic 1 and output is logic 0 and vice versa. The logic symbol is

A  Q

Truth table for NOT gate

A	**Q=$\overline{A}$**
0	1
1	0

3.1 3 NAND GATE.

A NAND gate is an AND gate followed by a NOT gate. Therefore the output of a NAND Gate is a 1 if at least one of its inputs is 0. The output is a Zero only when its inputs are ones (1s)

The logic symbol is

Truth table for NAND gate

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

3.1 4 OR GATE.

It's an electronic device which has two or more inputs and only one output. Output of the device is a 1 if at least one of the inputs is 1 and It's Zero if both of the inputs are 0.

The logic symbol is

Truth table for OR gate

Input A	**Input B**	**Output Q**
0	0	0
0	1	1
1	0	1
1	1	1

3.1 5 NOR GATE.

A NOR gate is made up from a OR gate followed by a NOT Gate.

The logic symbol is

Truth table for NOR gate

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

3.1 6 Exclusive – OR.(XOR)

It's a gate that implements the combination of the three fundamental gates (NOT, OR, AND).It has always two inputs. The output is 1 if exactly one of the inputs is 1. The name of the gate is derived from the fact that the output is 1 when exclusively one of the input is a 1 that is, It excludes the combination where both the inputs are 1's.
Note that the NOR Gate is also known as Exclusive OR gate because it excludes this combination. An XOR gate is constructed similarly to an OR gate, except with an additional NAND gate inserted such that if both inputs are high, the inputs to the final NAND gate will also be high, and the output will be low. This effectively represents the formula: "(A NAND (A NAND B)) NAND (B NAND (A NAND B))".

Truth Table		
Input A	**Input B**	**Output Q**
0	0	0
0	1	1
1	0	1
1	1	0

3.1 7 EXCLUSIVE NOR.

Also known as the coincidence gate coz it has 2 inputs. The output of this gate is a one IF and only IF both the inputs are one (1) or if both of the inputs are Zero (0). This means the output is 1 if input coincide (same) thus this logic complimentary to that of exclusive OR gate.
The exclusive OR gate followed by an inverter gives a coincidence.

Truth Table		
Input A	**Input B**	**Output Q**
0	0	1
0	1	0
1	0	0
1	1	1

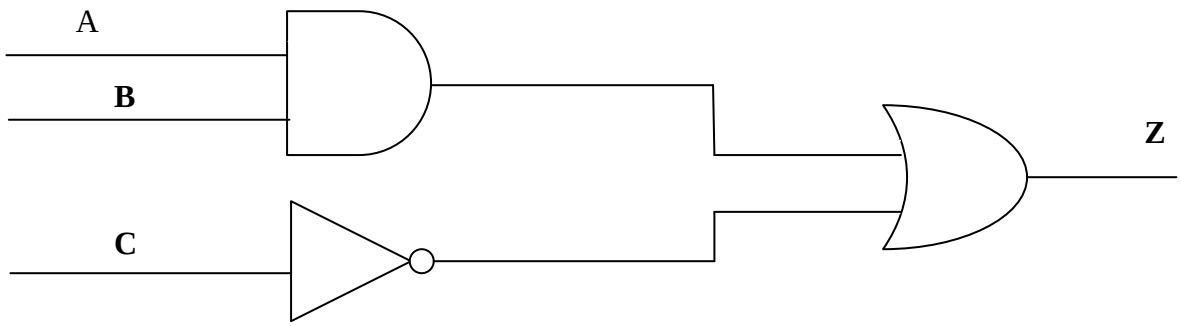
Constructing truth tables and logic expressions from logic diagrams.
You construct a separate column in the truth table for the output of each logic gate.

Input A	**Input B**	**AB**	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

You move from one logic gate to the next to develop the logic expression by gradually expanding that is compounding the logical operation as they are performed from the input to the output.

Questions

1. Construct the truth table and develop the logic expression for the output Z in the following diagram.



Solution

Truth table

A	B	C	$\bar{C}$	A.B	AB+$\bar{C}$
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	1	1

Logic expression= AB+$\bar{C}$

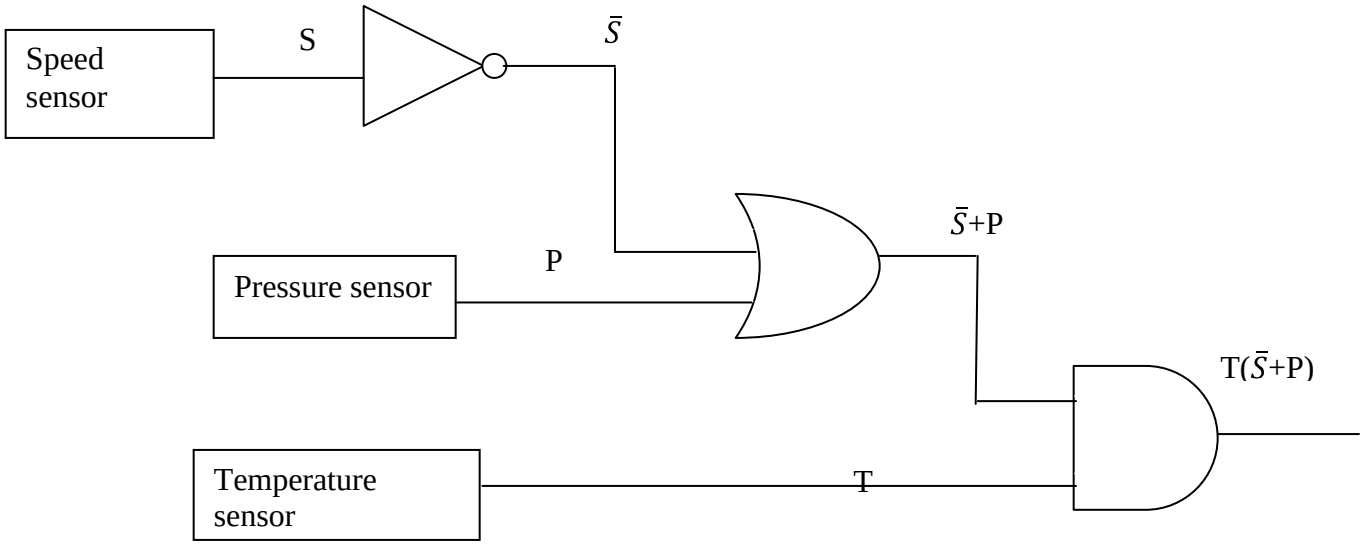
2. An aircraft engine is equipped with a safety system that turns on warning light when certain combination of engine speed, pressure and temperature occurs. The device that senses these quantities produces a 1 or 0 according to the table below.

Speed (S)	S<5000	0
	S≥5000	1
Pressure(P)	P≤200	1
	P>200	0
Temperature(T)	T>180	0
	T≤180	1

Solution.

Below is the logic diagram that controls the warning light in response to the input variables S,P and T. Assuming that a 1 turns on the light, develop the overall logic expression and construct the truth table if the warning light is on when:

a) The speed is 6250, the pressure is 280 and the temp is 150.
b) The speed is 7400, the pressure is 180 and the temp is 200.



Truth Table

S	P	T	$\bar{S}$	$\bar{S}$+P	T($\bar{S}$+P)
0	0	0	1	1	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	1	0
1	1	1	0	1	1

a) S=1, P=0, T=1

LIGHT OFF

b) S=1, P=1, T=0

LIGHT OFF

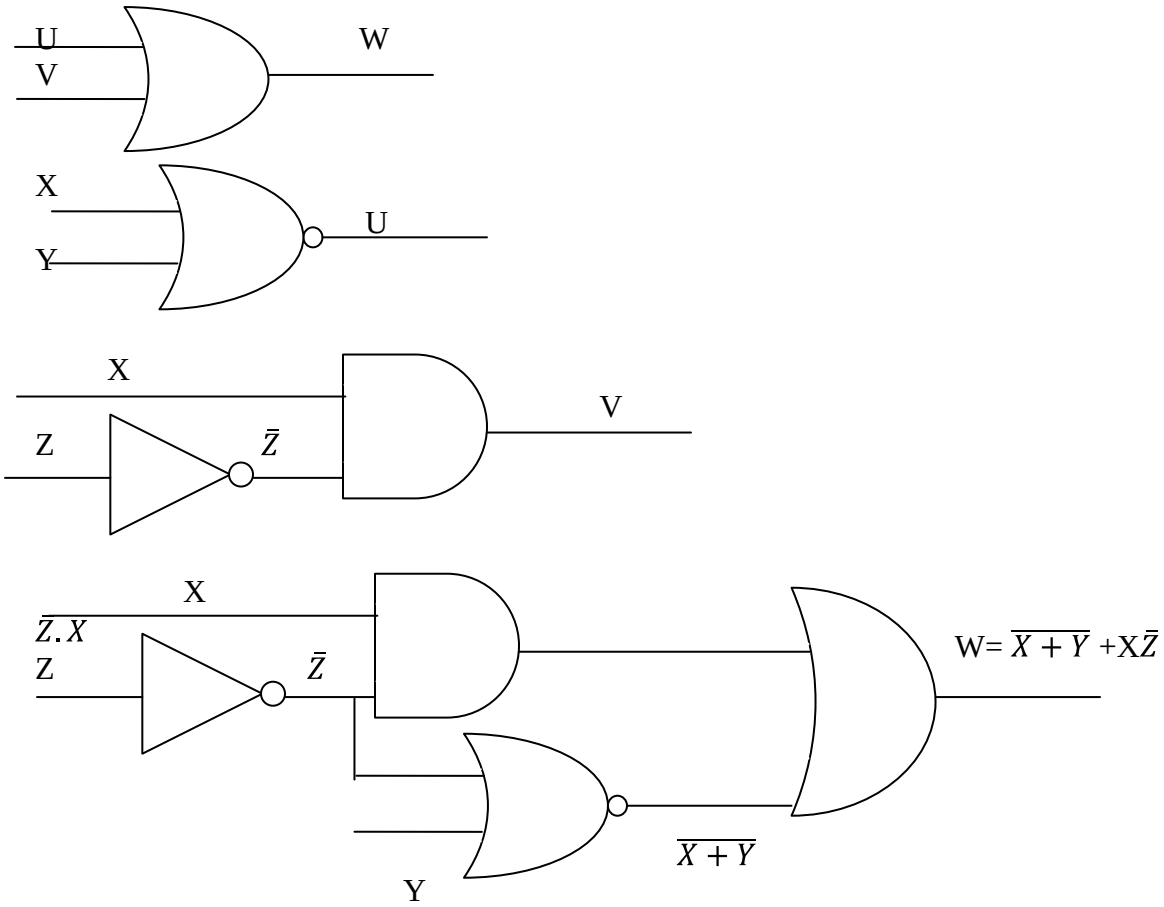
Constructing Logic Diagrams and Truth Tables from Logic Expressions.

In a typical design problem, a logic expression is derived first and a logic diagram that implements or performs the specified logical operation must be developed. The process simply involves drawing a gate for which logical operation contained in the expression. Make sure the correct variables or combination variables operated by this gates. The truth tables can be constructed directed on the logic expression or logic diagram.

Example.

1. Construct a truth table and a logic diagram that implements the following expression.

$$W=\overbrace{\overline{X+Y}}^{U}\ \overbrace{+X\bar{Z}}^{V}$$

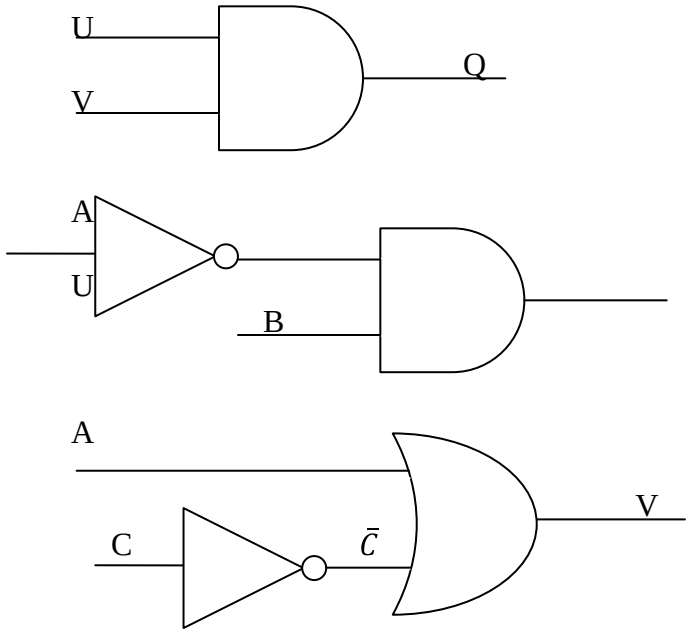


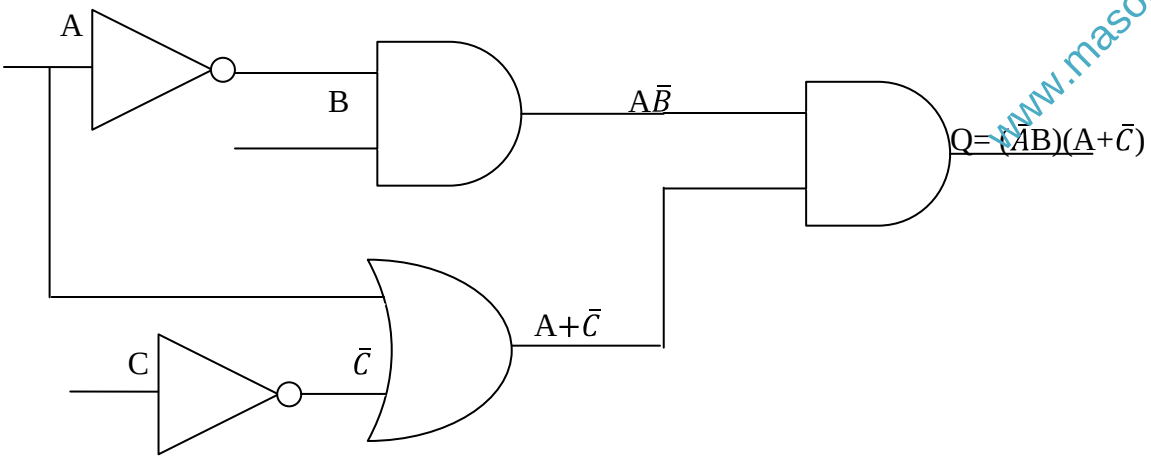
Truth Table

X	Y	Z	$\bar{Z}$	$\bar{Z}$.X	Y+X	$\overline{X+Y}$	$\overline{X+Y}$+X$\bar{Z}$
0	0	0	1	0	0	1	1
0	0	1	0	0	0	1	1
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	0	1	0	0
1	1	0	1	1	1	0	1
1	1	1	0	0	1	0	0

2. $\overbrace{\overline{AB}}^{U}\ \overbrace{(A+\bar{C})}^{V}$

Q= $(\overline{AB})$ (A+$\bar{C}$)





TRUTH TABLE

A	B	C	$\bar{A}$	$\bar{A}$.B	$\bar{C}$	A+$\bar{C}$	Q=($\bar{A}$B)(A+$\bar{C}$)
0	0	0	1	0	1	1	0
0	0	1	1	0	0	0	0
0	1	0	1	1	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	0
1	0	1	0	0	0	1	0
1	1	0	0	0	1	1	0
1	1	1	0	0	0	1	0

3. The computer circuit performs its oprations using three binary number A_2, $_{A1,}A_0$ where A_2 is the most and A_0 the least significant bit. if the numbers has certain decimal values, the circuit produces a GO signal that launches a rocket. A circuit produces a 1 in accordance to expression.

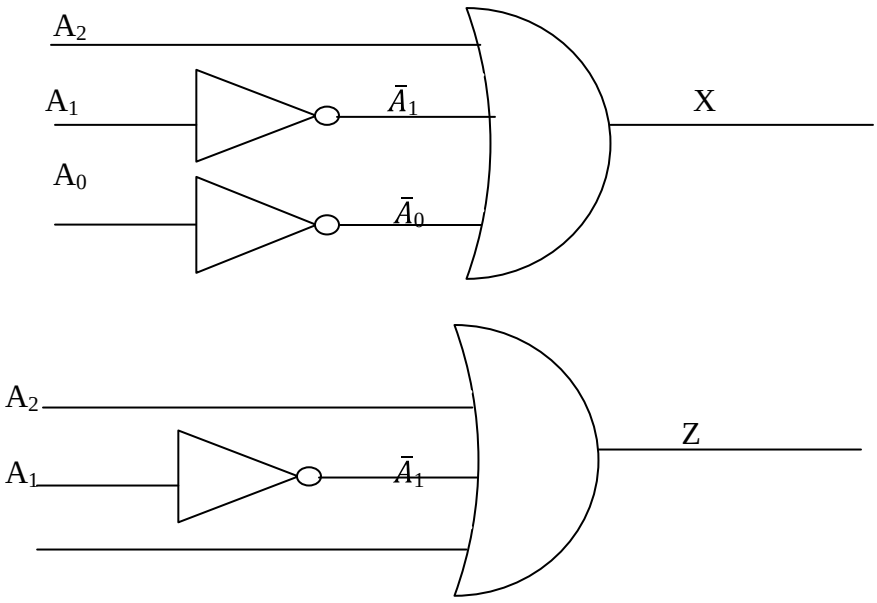
   $GO= \overline{(A_2+\bar{A}_1+\bar{A}_0)\ (\bar{A}_2+A_1+A_0)\ (A_2+\bar{A}_1+A_0)}$

   Draw a logic diagram and construct a truth table and further determine which decimal values which launches the socket when the Go signal = 1.

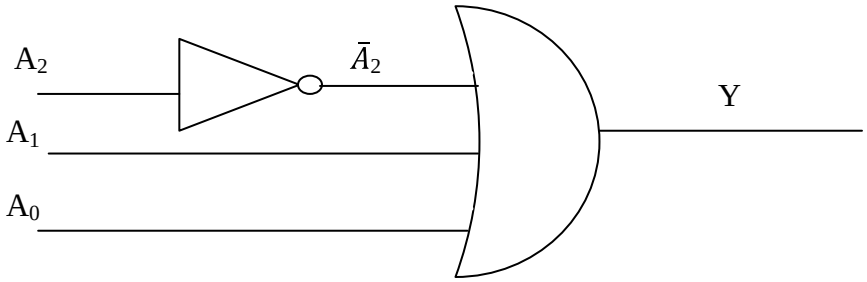
	A	B	C	$\bar{A}_2$	$\bar{A}_1$	A_0	$A_2+\bar{A}_1+\bar{A}_0$	$\bar{A}_2+A_1+A_0$	$A_2+\bar{A}_1+A_0$		GO
0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	0	1	1	1	1	0
2	0	1	0	1	0	1	1	1	0	0	1
3	0	1	1	1	0	0	0	1	1	0	1
4	1	0	0	0	1	0	1	0	1	0	1
5	1	0	1	0	1	0	1	1	1	1	0
6	1	1	0	0	0	1	1	1	1	1	0
7	1	1	1	0	0	0	1	1	1	1	0

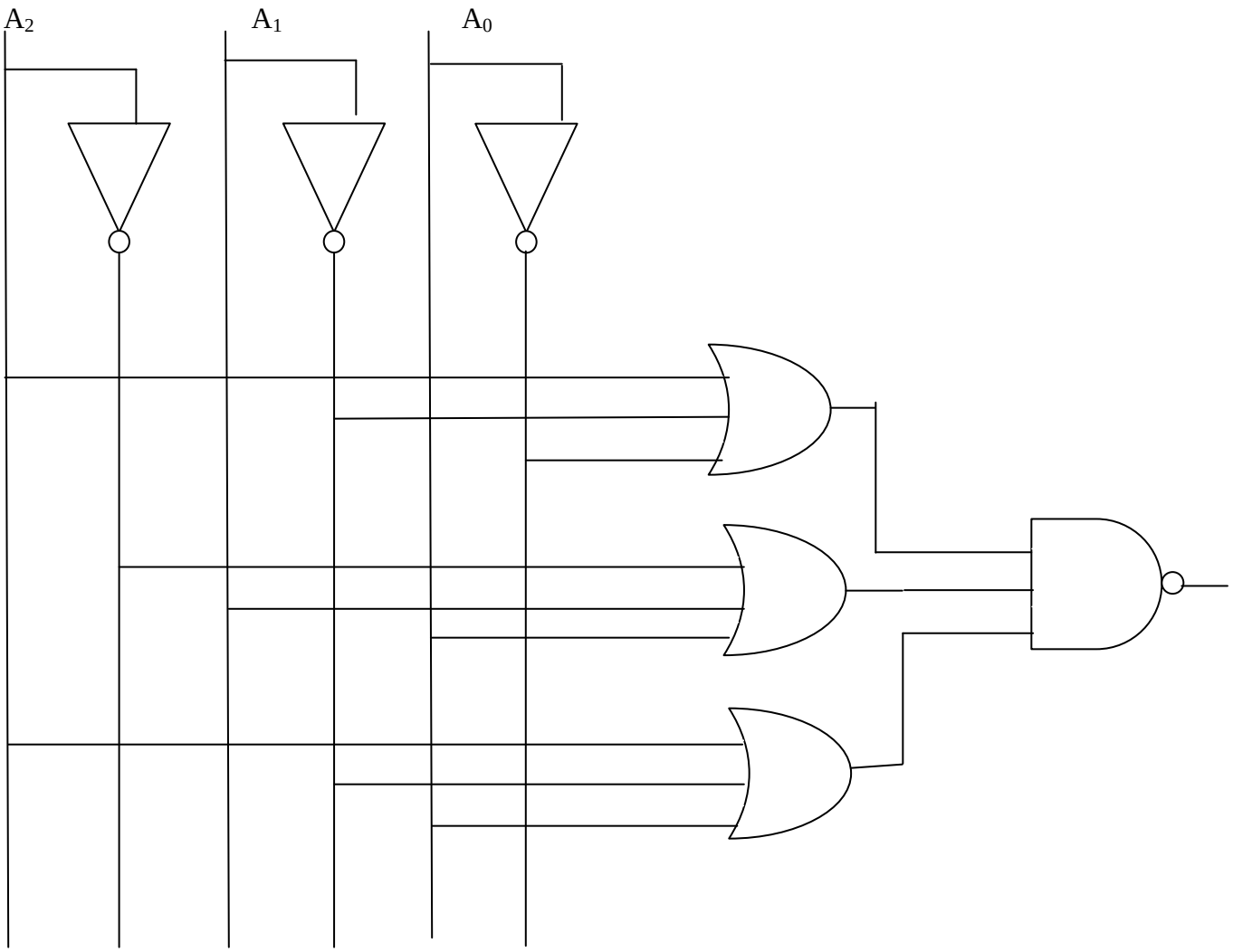
Numbers 2, 3 and 4

$$GO = \overline{(A_2+\bar{A}_1+\bar{A}_0)\ (\bar{A}_2+A_1+A_0)\ (A_2+\bar{A}_1+A_0)}$$
$$\qquad\qquad X \qquad\qquad Y \qquad\qquad Z$$



A_0

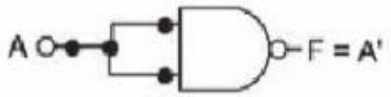




4. Write a logic expression that equals 1 only when two binary numbers A_1, A_0 and B_1, B_0 have the same value. Draw the logic diagram and construct the truth table to verify the logic.

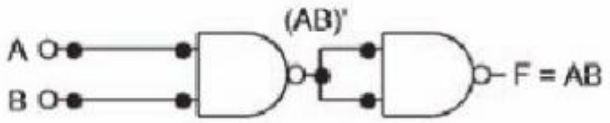
3.1 8 UNIVERSAL GATES

NAND gates and NOR gates are called universal gates or universal building blocks, as any type of gates or logic functions can be implemented by these gates. Figures 3.20(a)-(e) show how various logic functions can be realized by NAND gates and Figures 3.21(a)-(d) show the realization of various logic gates by NOR gates



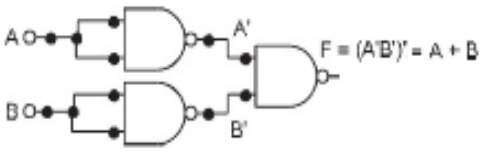
NOT function: F = A′

Figure 3.20(a)



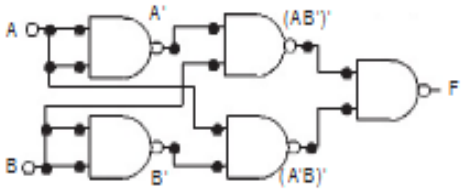
AND function: F = AB

Figure 3.20(b)



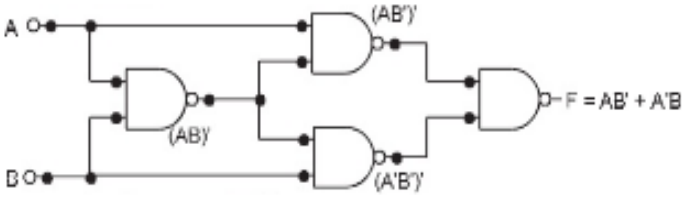
OR function: F = A + B

Figure 3.20(c)



Ex-OR function: F= ((AB′)′(A′B)′)′ =AB′ + A′B

Figure 3.20(d)

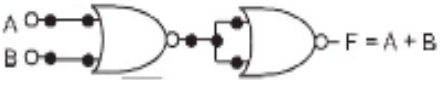


Ex-OR gate with reduced number of NAND gates

Figure 3.20(e)

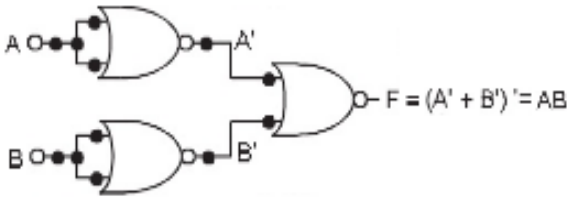
NOT function: F = A′

Figure 3.21(a)



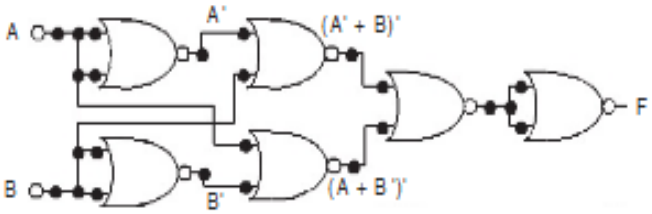
OR function: F = A + B

Figure 3.21(b)



AND function: F = AB

Figure 3.21(c)



Ex–OR function: F = [(A′ + B)′ + (A + B′)′] = AB′ + A′B

3.2 Two-level Implementation of Logic Networks

The maximum number of gates cascaded in series between an input and output is called the level of gates. For example, a sum of products (SOP) expression can be implemented using a two-level gate network, i.e., AND gates at the first-level and an OR gate at the second level. Similarly, a product of sums (POS) expression can be implemented by a two-level gate network, as OR gates at the first level and an AND gate at the second level. It is important to note that INVERTERS are not considered to decide the level of gate network.

Apart from the realization of Boolean functions using AND gates and OR gates, the

NAND gates and NOR gates are most often found in the implementation of logic circuits as they are universal type by nature. Some of the NAND and NOR gates allow the possibility of a wire connection between the outputs of two gates to provide a specific logic function.

This type of logic is called wired logic. (This will be discussed in detail in Chapter 11: Logic Family.) When two NAND gates are wired together as shown in Figure (a), they perform the wired-AND logic function. AND drawn with lines going through the center of the gate is symbolized as a wired-AND logic function. The wired-AND gate is not a physical gate, but only a symbol to designate the function obtained from the indicated wired connections.

The logic function implemented by the circuit of Figure (a) is

$F = (WX)'.(YZ)' = (WX + YZ)'$.

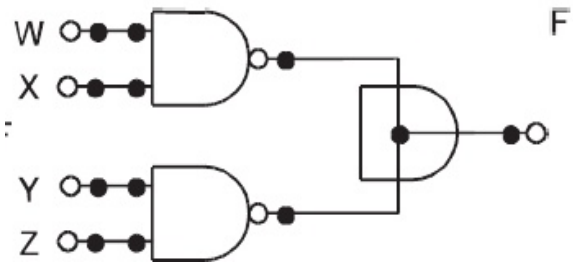


Figure (a)

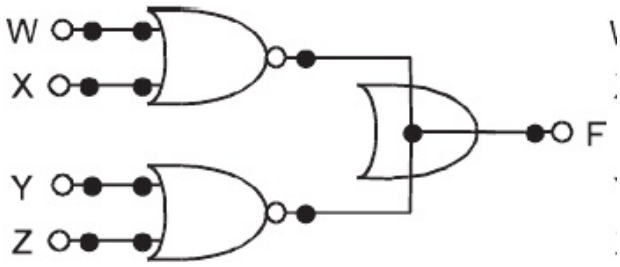


Figure (b

The above function is referred to as an AND-OR-INVERT function. Similarly, some specially constructed NOR gates outputs can be tied together to form the wired-OR function as shown in Figure (b). The logic function implemented by Figure (b) is

$F = (W + X)' + (Y + Z)' = [(W + X).(Y + Z)]'$.

This function is called an OR-AND-INVERT function.

The wired logic gate does not produce a physical second level gate since it is just the wire connection. However, according to the logic function concerned, wired logic is considered a two-level implementation.

3.3 Chapter Review Questions

1. What is the truth table and logic symbol of a three-input OR gate?
2. Write the expression for a 4-input AND gate. Construct the complete truth table showing the output for all possible cases.
3. Define NAND and NOR gates with their truth tables.
4. What is a logic gate? Explain logic designation.
5. Discuss the operation of Ex-OR and Ex-NOR gates with truth tables and logic diagram.
6. Explain the term 'universal gate.' Name the universal gates.
7. Explain how basic gates can be realized by NAND gates.
8. Explain how basic gates can be realized by NOR gates.
9. Construct a two-input XOR gate using NAND gates. Construct the same with NOR gates.
10. Realize an INVERTER with two-input XOR gate only.
11. Realize the logic expression for $A \oplus B \oplus C \oplus D$.
12. Draw a logic circuit for the function $F = (A + B)(B + C)(A + C)$, using NOR gates only.

CHAPTER FOUR

BOOLEAN ALGEBRA

Chapter objectives

1. *Explain Boolean algebra*
2. *Explain the various Boolean relations*
3. *Describe the de Morgan's theorem*
4. *Simplify Boolean expressions using Boolean algebra*
5. *Describe the standard forms*
6. *Describe the karnaugh map*
7. *Simplify Boolean expression using karnaugh map*
8. *Two level implementation of logic networks*
9. *Describe the tabulation method*

4.1 Introduction

George Boole is credited with the invention of what is now called Boolean logic. Digital electronics is based entirely on the fundamental principles of Boolean logic. Every digital circuit has inputs and outputs. Then, purpose of the digital circuit can be thought of as "matching" an output to a certain input (or a sequence of inputs). For example, a digital circuit can be designed to control the movement of an elevator. The inputs are the floor you are on and the floor you want to go to. The output is the motor being turned on in the right direction for the correct period of time. To design this and more complicated digital circuits, we use a field of mathematics called Boolean algebra.

Boolean algebra can be used to formalize the combinations of binary logic states.
It also helps to discover redundancies or contradictions that may arise. Axioms, Postulates of Boolean algebra are set of logical expressions that we accept without prove and with upon which we could build a set of useful theorems. Axioms are nothing but the definitions of the three basic functions that is AND, OR, NOT.

10 AXIOMS.

1.	0.0 = 0		0+0 = 0		1=0
2.	0.1 = 0	AND	1+0 = 1	OR	0=1
3.	1.0 = 0		0+1 = 1		
4.	1.1 = 1		1+1 = 1		

4.1 1 BOOLEAN RELATIONS.

1. **Commutative property.**
   a) AB = BA
      There is no difference which input of a AND Gate is connected to A.
   b) A + B = B+A
      There is no difference which input of the OR is connected to A or which is connected to B.

This two theorems can be applied to many variables e.g.
A+B+C+D=D+A+C+B

2. Associative property

a) A(BC)=(AB)C
b) (A+B)+C=A+(B+C)
The associative property can be extended to any number of variables.

3. **Idempotent property**

(a) A + A = A

If A=0, 0+0=0, =A

If A=1, 1+1=1, =A

(b) A A = A

If A=0, 0.0=0, =A

If A=1, 1.1=1, =A

4. Identity property

a) A.1=A
If A=0, 0.1=0, =A
If A=1, 1.1=1, =A

b) A+1=1
If A=0, 0+1=1,
If A=1, 1+1=1
Commutative property can be applied to identity property i.e
A.1=A 1.A=A

5. Null property

a) A.0=0
If A=0, 0.0=0
A=1, 1.0=0

N/B: anything ANDED with a zero produces a zero.

b) A+0=A
IF A=0, 0+0=0, =A
A=1, 1+0=1, =A

This property can also apply to a combination of properties

(AB+C) 0=0
ABC+DC+0=ABC+DC

6. **Distributive property**
   a) A (B + C) = A B + A C
   b) A + (B C) = (A + B) (A + C)

7. **Negation property**
   a) $\overline{A}\ A = 0$
   If A=0, 1.0=0
   If A=1, 0.1=0

   b) $\overline{A} + A = 1$
   If A=0, 1+0=1
   If A=1, 0+1=1

   Negation property can also apply to many variables that is ABC+ $\overline{ABC}$=1

8. **Double negation**
   a) $\bar{\bar{A}}$=A
   If A=0, $\bar{A}$=1, $\bar{\bar{A}}$=0, =A
   A=1, $\bar{A}$=0, $\bar{\bar{A}}$=1, =A

Any odd numbered inversion equals to the single inversion. Any even number of inversions equals to no inversion at all.

9. **Absorption property**
   a) A + A B = A

   A(A+B)=A.1
   =A

   b) A (A + B) = A

   AA+AB=A+AB
   =A

   These variables can be replaced by a number of variables
   c) A+$\bar{A}$B=A+B

$$
\begin{aligned}
A + \overline{A}\ B &= A\ 1 + \overline{A}\ B \\
&= A\ (1 + B) + \overline{A}\ B \\
&= A + A\ B + \overline{A}\ B \\
&= A + B\ (A + \overline{A}) \\
&= A + B
\end{aligned}
$$

10. De Morgan's theorem

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

A NAND gate performs the same operation as an OR gate with all the inputs inverted

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

b) $(\overline{A+B}) = \overline{A}\ \overline{B}$

An NOR gate performs the same operation as an AND gate with the inputs inverted.

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

4.1 2 SIMPLIFYING EXPRESSIONS USING BOOLEAN ALGEBRA

Logic expression can be simplified for two reasons;

a) A simple expression can be implemented using fewer gates.
b) Simplifications can review redundant variables or contradictions.

Simplifications can be obtained by either using Boolean algebra or using k-map

Using Boolean Algebra

Example 3.1. Simplify the Boolean function F=AB+ BC + $\bar{B}$ C.
Solution. F = AB + BC + $\bar{B}$ C
= AB + C(B + $\bar{B}$)
= AB + C
Example 3.2. Simplify the Boolean function F= A + $\bar{A}$ B.
Solution. F = A+ $\bar{A}$ B
= (A + $\bar{A}$) (A + B)
= A + B
Example 3.3. Simplify the Boolean function F= $\bar{A}$ $\bar{B}$ C + $\bar{A}$ BC + A $\bar{B}$.
Solution. F = $\bar{A}$ $\bar{B}$ C + $\bar{A}$ BC + A $\bar{B}$
= $\bar{A}$ C ($\bar{B}$ +B) + A $\bar{B}$
= $\bar{A}$ C + A $\bar{B}$
Example 3.4.Simplify the Boolean function F = AB + ($\overline{AC}$) + A $\bar{B}$ C(AB + C).
Solution. F = AB + ($\overline{AC}$) + A $\bar{B}$ C(AB + C)
= AB + $\bar{A}$ + $\bar{C}$+ A$\bar{B}$ C.AB + A $\bar{B}$ C.C
= AB + $\bar{A}$ + $\bar{C}$ + 0 + A $\bar{B}$ C
(B.$\bar{B}$ = 0 and C.C = C)
= ABC + AB $\bar{C}$ + $\bar{A}$ + $\bar{C}$ + A $\bar{B}$ C (AB = AB(C + $\bar{C}$) = ABC + AB$\bar{C}$)

= AC(B + $\bar{B}$) + $\bar{C}$ (AB + 1) + $\bar{A}$
= AC + $\bar{C}$ +$\bar{A}$
(B + $\bar{B}$ = 1 and AB + 1 = 1)
= AC + ($\overline{AC}$)
= 1
Example 3.5.Simplify the Boolean function F = ((XY′ + XYZ)′ + X(Y + XY′))′.
Solution. F = ((XY′ + XYZ)′ + X(Y + XY′))′
= ((X(Y′ + YZ))′ + XY + XY′)′
= ((X(Y′Z + Y′ + YZ))′ + X(Y + Y′))′
(Y′ = Y′(Z + 1) = Y′Z + Y′)
= (X(Y′ + Z))′ + X)′
= (X′ + (Y′ + Z)′ + X)′
= (1+ YZ′)′
= 1 ′
= 0
Example 3.6.Simplify the Boolean function F = XYZ + XY′Z + XYZ′.
Solution. F = XYZ + XY′Z + XYZ′
= XZ (Y + Y′) + XY (Z + Z′)
= XZ + XY
= X (Y + Z

4.2 Standard Forms

The product of sum is any group of sum terms ANDED for example
(A+$\bar{B}$+$\bar{C}$)(A+B+$\bar{C}$)
A sum term is any group of variables that are Ored together.
A sum of products is a group of product term Ored together. For example
$\bar{A}\bar{B}$C+AB$\bar{C}$+$\bar{A}$BC
The sum of product is said to be standard or canonical if every term involves every variables or its complement. For example
F=$\bar{A}\bar{B}\bar{C}$+A$\bar{B}\bar{C}$+AB$\bar{C}$ where A, B and C are the only variables pertaining to the logic.
After F=$\bar{A}\bar{B}\bar{C}$+A$\bar{B}\bar{C}$+AB$\bar{C}$ is simplified what you obtain is not a standard form (standard form of product).
A product of sum (POS) is standard form if every sum term involves every variable or its complement.
W=(X_1+X_2+X_3) ($\bar{X}_1$+X_2+X_3) where X_1, X_2 and X_3 are th$_e$ only variables pertaining to that logic
However, Boolean functions are also sometimes expressed in nonstandard forms like
F = (AB + CD)($\overline{AB}$ + $\overline{CD}$), which is neither a sum of products form nor a product of sums form.
However, Boolean functions are also sometimes expressed in nonstandard forms like

F = (AB + CD)($\bar{A}$ $\bar{B}$ + $\bar{C}$ $\bar{D}$), which is neither a sum of products form nor a product of sums form.
However, the same expression can be converted to a standard form with help of various Boolean properties, as
F = (AB + CD)($\bar{A}$ $\bar{B}$ + $\bar{C}$ $\bar{D}$) = $\bar{A}$ $\bar{B}$CD + AB$\bar{C}$ $\bar{D}$
.

Minterm

A product term containing all n variables of the function in either true or complemented form is called the minterm. Each minterm is obtained by an AND operation of the variables in their true form or complemented form. For a two-variable function, four different combinations are possible, such as, $\bar{A}\ \bar{B}$, $\bar{A}$ B, A$\bar{B}$, and AB. These product terms are called the fundamental products or standard products or minterms. In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form. For three variables function, eight minterms are possible as listed in the following table below.

A B C Minterm

0 0 0 $\bar{A}\ \bar{B}\ \bar{C}$

0 0 1 $\bar{A}\ \bar{B}$ C

0 1 0 $\bar{A}$ B $\bar{C}$

0 1 1 $\bar{A}$ BC

1 0 0 A$\bar{B}\ \bar{C}$

1 0 1 A$\bar{B}$ C

1 1 0 AB $\bar{C}$

1 1 1 ABC

Maxterm

Maxterm

A sum term containing alln variables of the function in either true or complemented form is called the maxterm. Each maxterm is obtained by an OR operation of the variables in their true form or complemented form. Four different combinations are possible for a two-variable function, such as, $\bar{A}$ + $\bar{B}$, $\bar{A}$ + B, A + $\bar{B}$, and A + B. These sum terms are called the standard sums or maxterms. Note that, in the maxterm, a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1, if it is in complemented form. Like minterms, for a three-variable function, eight maxterms are also possible as listed in the following table in Figure 3.13.

A B C Maxterm

0 0 0 A + B + C

0 0 1 A + B + $\bar{C}$

0 1 0 A + $\bar{B}$ + C

0 1 1 A + $\bar{B}$ + $\bar{C}$

1 0 0 $\bar{A}$ + B + C

1 0 1 $\bar{A}$ + B + $\bar{C}$

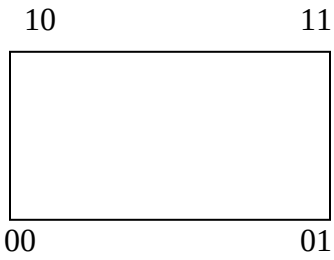
1 1 0 $\bar{A}$ + $\bar{B}$ + C

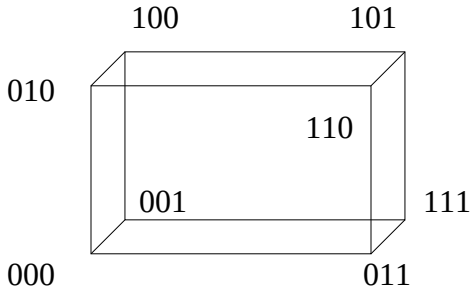
1 1 1 $\bar{A}$ + $\bar{B}$ + $\bar{C}$

4.3 Karnaugh Map

It's a simple straight forward procedure for minimizing the number of operators in the standard form expression. To generate such a standard form, we begin a convenient form of truth table called

karnaugh which was proposed by veitch and later modified by karnaugh. The map is designed to help us identify the smallest possible sub cubes that define a given Boolean function. Each sub cube represents a product in a standard sum of products.





Convenient way to assign product terms to the karnaugh map is to label each row and column of the map with the variable, with this complement or with a combination variable or complement. The product term corresponding to a given cell is the product of all the variables of the rows and columns of the cell where the cell is located.
To plot logic expression of the karnaugh we write 1 each cell that corresponds to the product term in the expression.
e.g

F=A$\bar{B}$+AB

	$\bar{A}$	A
$\bar{B}$	0	1
B	*0*	1

Thus the logic expression that is to be simplified using a karnaugh is to be written or rewritten in the sum of product form. This form can be obtained by Boolean manipulation where necessary or by construction a truth table. To simplify a logic expression plotted on the map we first circle group of adjacent cells containing ones. Adjacent cells are those that adjoin each other along the rows or columns but not as diagonal. Each group of adjacent cells that we circle must contain a group of ones that equal to the power of 2 i.e. $2^{0,}$ 2^1 , 2^n

A group can consist a single cell containing a one that 2^0 where there is no other adjacent cells.
The circled group must be as large as possible subject to the requirement that the one must contain 1,2,4,8,16

5.1 1 Two Variable K-Map

The karnaugh map is the visual display of the fundamental products needed for a solved product/solution. It's a chart/grid containing boxes which are called shelves which represents one of the 2^n of the possible products that can be formed from n variables. Thus a 2 variable map contains 4 shelves.

2^2=4

	$\bar{A}$	A
$\bar{A}$	$\bar{A}\bar{B}$	A$\bar{B}$
B	$\bar{A}$B	AB

5.1 2 Three-Variable Karnaugh Map

Since, there are eight minterms for three variables; the map consists of eight cells or squares, which is shown in Figure (a). It may be noticed that the minterms are arranged, not according to the binary sequence, but according to the sequence similar to the reflected code, which means, between two consecutive rows or columns, only one single variable changes its logic value from 0 to 1 or from 1 to 0. Figure (b) shows the relationship between the squares and the variables. Two rows are assigned to $\bar{C}$ and C, and four columns to $\bar{A}\ \bar{B}$, $\bar{A}$B, AB and A $\bar{B}$. The mintermm3, for example, in the row C and column $\bar{A}$B,, asm3 = $\bar{A}$BC. Note that, each of the variables has four squares where its logic value is 0 and four squares with logic value 1.

2^3=8

	$\bar{A}\ \bar{B}$	$\bar{A}$B	AB	A $\bar{B}$
$\bar{C}$	m_0	m_2	m_6	m_4
C	m_1	m_3	m_7	m_5

Figure (a).

	$\bar{A}\ \bar{B}$	$\bar{A}$B	AB	A $\bar{B}$
$\bar{C}$	$\bar{A}\ \bar{B}\ \bar{C}$	$\bar{A}$B$\bar{C}$	ABC	A $\bar{B}\ \bar{C}$
C	$\bar{A}\ \bar{B}$ C	$\bar{A}$BC	ABC	A $\bar{B}\ C$

Figure (b)

2^3=8

	$\bar{A}\ \bar{B}$	$\bar{A}$B	AB	A $\bar{B}$
$\bar{C}$	$\bar{A}\bar{B}\ \bar{C}$	$\bar{A}$B$\bar{C}$	ABC	A$\bar{B}\ \bar{C}$
C	$\bar{A}\bar{B}$ C	$\bar{A}$BC	ABC	A$\bar{B}$ C

Figure (c)

5.1 3 Four-Variable Karnaugh Maps

Similar to the method used for two-variable and three-variable Karnaugh maps, four-variable Karnaugh maps may be constructed with 16 squares consisting of 16 minterms as shown in Figure below. The same is redrawn in Figure below to show the relationship with the four binary variables. The rows and columns are numbered in a reflected code sequence, where only one variable is changing its form between two adjacent squares. The minterm of a particular square can be

obtained by combining the row and column. As an example, the minterm of the second row and third column is $\bar{A}$BCDi.e.,m7.

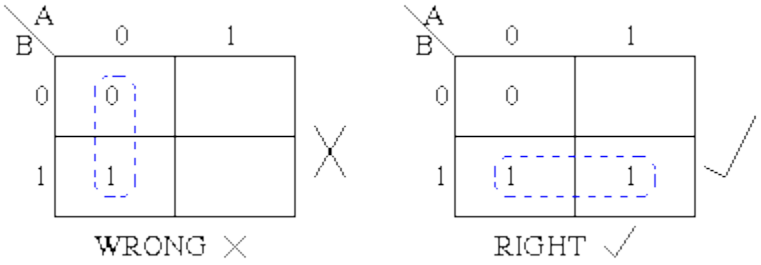
	$\bar{C}\bar{D}$	$\bar{C}$D	CD	C$\bar{D}$
$\bar{A}\bar{B}$	m0	m1	m3	m2
$\bar{A}$B	m4	m5	m7	m6
AB	m12	m13	m15	m14
A$\bar{B}$	m8	m9	m11	m10

	$\bar{C}\bar{D}$	$\bar{C}$D	CD	C$\bar{D}$
$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}$D	$\bar{A}\bar{B}$CD	$\bar{A}\bar{B}$ C$\bar{D}$
$\bar{A}$B	$\bar{A}$B$\bar{C}\bar{D}$	$\bar{A}$B$\bar{C}$D	$\bar{A}$BCD	$\bar{A}$B C$\bar{D}$
AB	AB$\bar{C}\bar{D}$	AB$\bar{C}$D	ABCD	AB C$\bar{D}$
A$\bar{B}$	A$\bar{B}\bar{C}\bar{D}$	A$\bar{B}\bar{C}$D	A$\bar{B}$CD	A$\bar{B}$ C$\bar{D}$

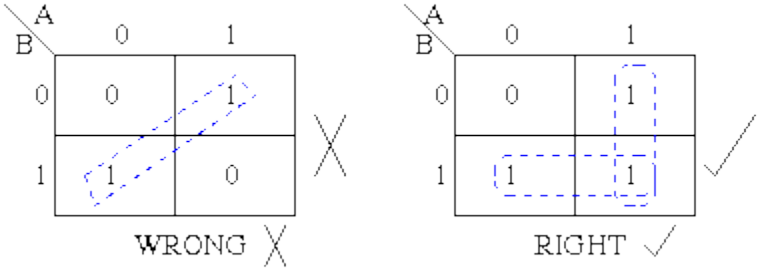
5.1 4Karnaugh map rules

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**

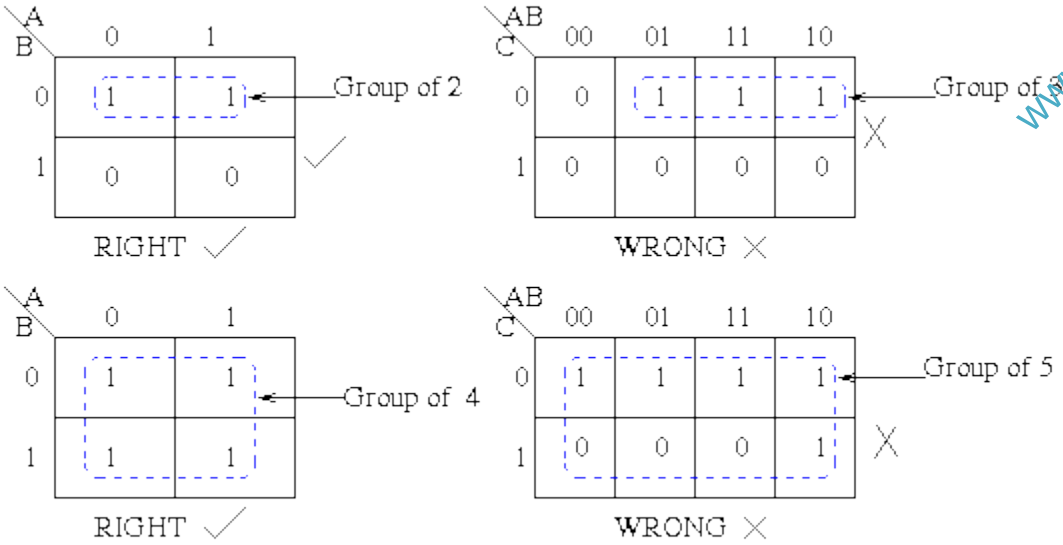


- **Groups may be horizontal or vertical, but not diagonal.**

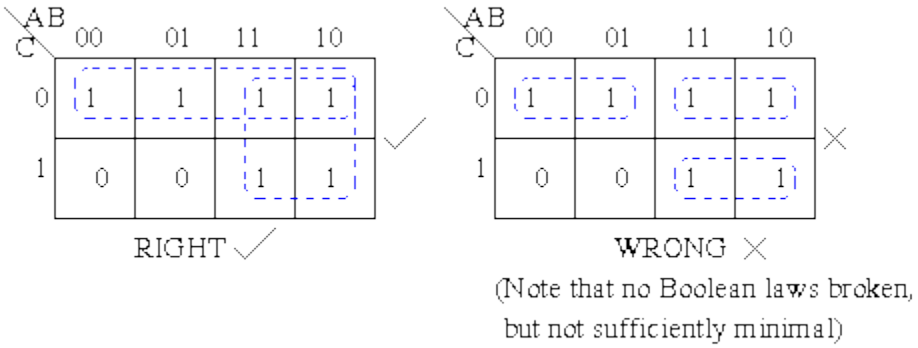


- **Groups must contain 1, 2, 4, 8, or in general 2^n cells. That is if n = 1, a group will contain two 1's since 2^1 = 2.**

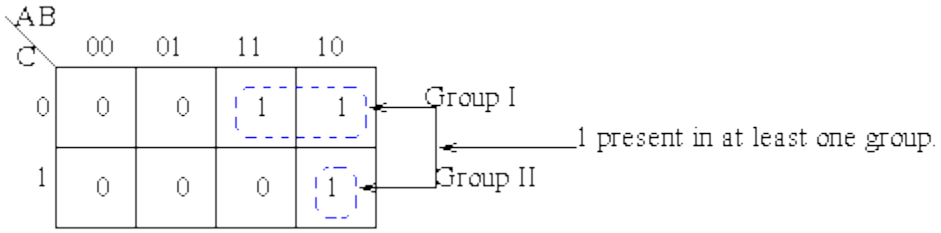
If n = 2, a group will contain four 1's since $2^2 = 4$.



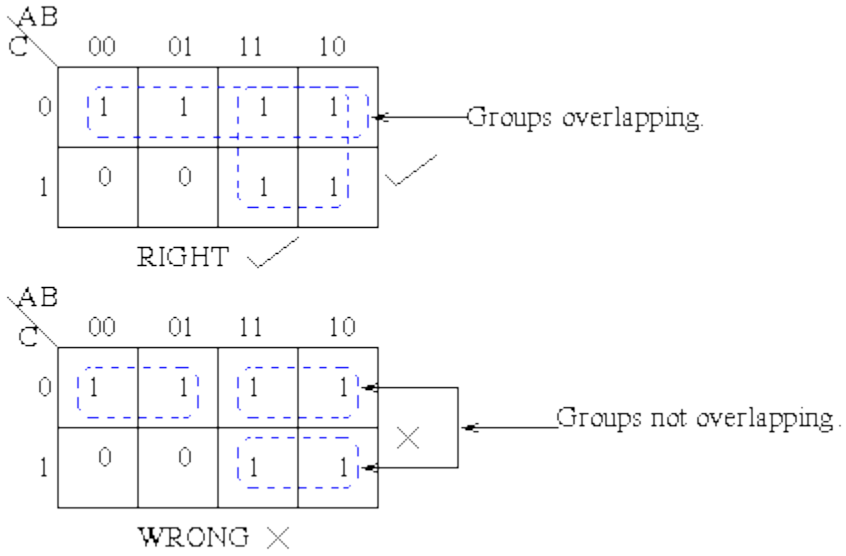
- **Each group should be as large as possible.**



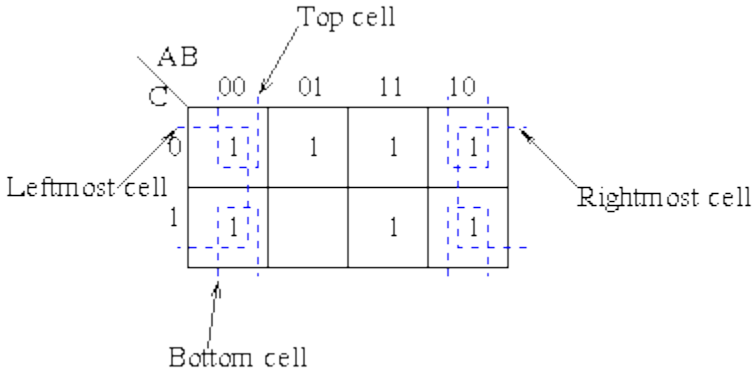
- **Each cell containing a *one* must be in at least one group.**



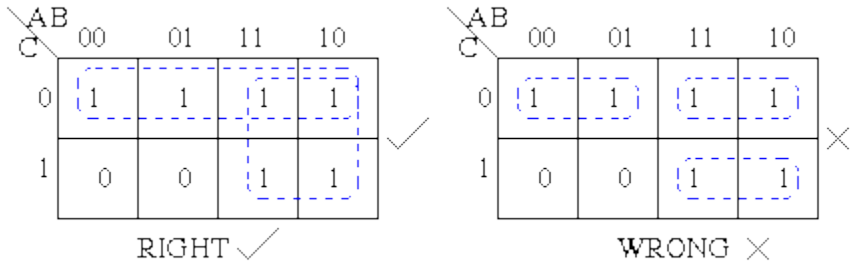
- **Groups may overlap.**



- **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.**



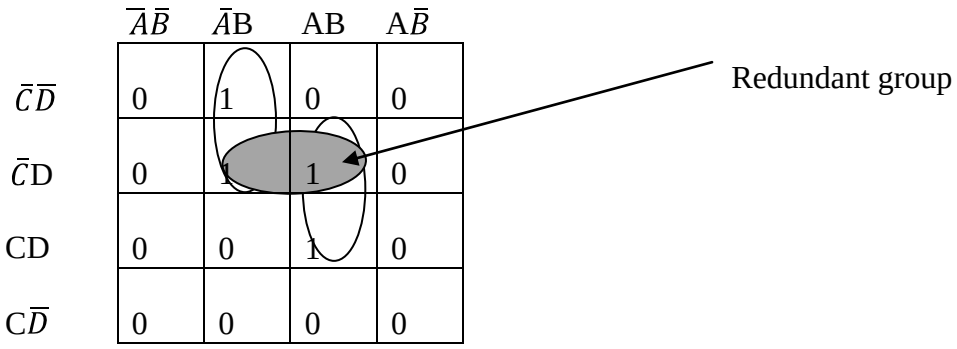
- **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**



5.1 5 Redundant group

After encircling groups before coming up with the simplified Boolean expression eliminate any group whose 1's are completely overlapped by other groups. Such a group is known as redundant group.

Example



5.1 6 Don't care condition.

At times it does not matter what the output is for a given input word, to indicate this condition an x is used in the truth table instead of a zero or one. This x is known as don't care because they can either be zeros or ones. During simplification of the logic expression, the x's are circled with ones In certain cases some of the minterms may never occur or it may not matter what happens if they do. In such cases we fill in the Karnaugh map with and X meaning don't care

– When minimizing an X is like a "joker"

• X can be 0 or 1 - whatever helps best with the minimization

– Eg:

A\BC	*00*	*01*	11	10
0	0	0	1	X
1	0	0	1	1

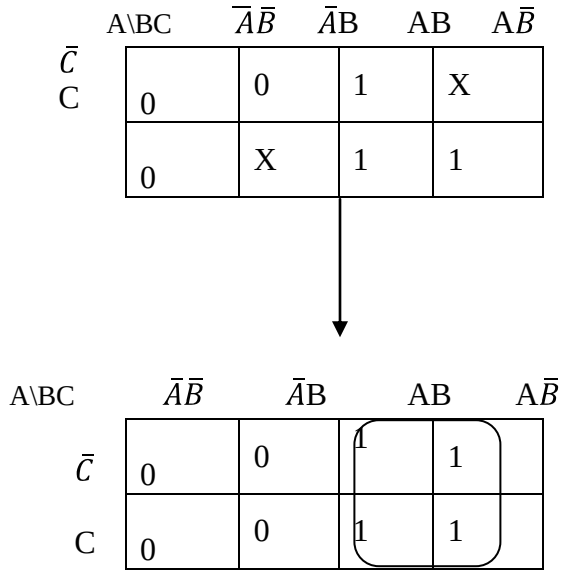
– simplifies to *B* if X is assumed 1

"Don't Care" examples

"Don't care" conditions should be changed to either 0 or 1 to produce K-map looping that yields the simplest. Expression.

Truth table

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	X
1	0	0	X
1	0	1	1
1	1	0	1
1	1	1	1



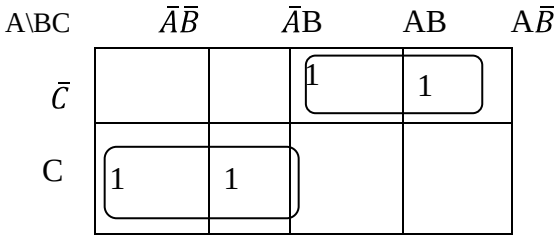
Z=A

Note: a group involving the x must have a one.

Simplify the Boolean function using K-map

1. F=$\bar{A}\bar{B}$C+$\bar{A}$BC+AB$\bar{C}$+A$\bar{B}\bar{C}$

Solution.
First, a three-variable Karnaugh map is drawn and 1s are placed at the squares according to the minterms of the function as shown in Figure below.



Now two 1s of adjacent squares are grouped together. As in the figure above. $AB\bar{C}$ and $A\bar{B}\bar{C}$. are grouped together at the first row, and $\overline{AB}C$ and $\bar{A}BC$ are grouped together In the second row. From the first row, the reduced term of $AB\bar{C} + A\bar{B}\bar{C}$ is $A\bar{C}$, as B is the variable which changes its form. Similarly from the second row, $\overline{AB}C + \bar{A}BC$ can be simplified to $\bar{A}C$. Now, as further simplification is not possible for this particular Boolean function, the simplified sum of the product of the function can be written as,

$F = A\bar{C}+\bar{A}C$.

5.2 The Tabulation Method

The Karnaugh map method is a very useful and convenient tool for simplification of Boolean functions as long as the number of variables does not exceed four (at the most six). But if the number of variables increases, the visualization and selection of patterns of adjacent cells in the Karnaugh map becomes complicated and difficult. The tabular method, also known as the Quine-McCluskey method, overcomes this difficulty. It is a specific step-by-step procedure to achieve guaranteed, simplified standard form of expression for a function. The following steps are followed for simplification by the tabular or Quine-McCluskey method.

1. An exhaustive search is done to find the terms that may be included in the simplified functions. These terms are called prime implicants.
2. Form the set of prime implicants, essential prime implicants are determined by preparing a prime implicants chart.
3. The minterms that are not covered by the essential prime implicants, are taken into consideration by selecting some more prime implications to obtain an optimized Boolean expression.

Determination of Prime Implicants

The prime implicants are obtained by the following procedure:

1. Each minterm of the function is expressed by its binary representation.
2. The minterms are arranged according to increasing index (index is defined as the number of 1s in a minterm). Each set of minterms possessing the same index are separated by lines.
3. Now each of the minterms is compared with the minterms of a higher index. For each pair of terms that can combine, the new terms are formed. If two minterms are differed by only one variable, that variable is replaced by a '-' (dash) to form the new term with one less number of literals. A line is drawn in when all the minterms of one set is compared with all the minterms of a higher index.
4. The same process is repeated for all the groups of minterms. A new list of terms is obtained after the first stage of elimination is completed.
5. At the next stage of elimination two terms from the new list with the '-' of the same position differing by only one variable are compared and again another new term is formed with a less number of literals.
6. The process is to be continued until no new match is possible.
7. All the terms that remain uncheckedi.e., where no match is found during the process, are considered to be the prime implicants.

4.8.2 Prime Implicant Chart

1. After obtaining the prime implicants, a chart or table is prepared where rows are represented by the prime implicants and the columns are represented by the minterms of the function.

2. Crosses are placed in each row to show the composition of the minterms that makes the prime implicants.
3. A completed prime implicant table is to be inspected for the columns containing only a single cross. Prime implicants that cover the minterms with a single cross are called the essential prime implicants. The above process to find the prime implicants and preparation of the chart can be illustrated by the following examples. Example; Obtain the minimal sum of the products for the function F (A, B, C, D) =Σ (1, 4, 6, 7, 8, 9, 10, 11, 15).

Solution. The table in Figure 4.28 shows the step-by-step procedure the Quine-McCluskey method uses to obtain the simplified expression of the above function.

Column I consists of the decimal equivalent of the function or the minterms and column II is the corresponding binary representation. They are grouped according to their indexi.e, number of 1s in the binary equivalents. In column III, two minterms are grouped if they are differed by only a single variable and equivalent terms are written with a '-' in the place where the variable changes its logic value. As an example, minterms 1 (0001) and 9 (1001) are grouped and written as 1,9 (–001) and so on for the others. Also, the terms of column II, which are considered to form the group in column III, are marked with '√'

I	*II*					*III*			*IV*	
Decimal equivalent	*Binary equivalent* *A*	*B*	*C*	*D*		*ABCD*			*ABCD*	
1	0	0	0	1	√	1,9	–001		8,9,10,11	10– –
4	0	1	0	0	√	4,6	01–0		8,10,9,11	10– –
8	1	0	0	0	√	8,9	100–	√		
						8,10	10–0	√		
6	0	1	1	0	√	6,7	011–			
9	1	0	0	1	√	9,11	10–1	√		
10	1	0	1	0	√	10,11	101–	√		
7	0	1	1	1	√	7,15	–111			
11	1	0	1	1	√	11,15	1–11			
15	1	1	1	1	√					

The terms which are not marked with '√' are the Prime implicants. To express the prime implicants algebraically, variables are to be considered as true form in place of 1s, as complemented form in place of 0s, and no variable if '-' appears. Here the prime implicants are B′C′D, A′BD′, A′BC, BCD, ACD (from column III), and AB′ (from column IV). So the Boolean expression of the given function can be written as

F = AB′ + B′C′D + A′BD′ + A′BC + BCD + ACD.

But the above expression may not be of minimized form, as all the prime implicants may not be necessary. To find out the essential prime implicants, the following steps are carried out. A table or chart consisting of prime implicants and the decimal equivalent of minterms as given in the expression, as in Figure below is prepared

Prime Implicants	*1*	*4*	*6*	*7*	*8*	*9*	*10*	*11*	*15*
√ AB′					X	X	X	X	
√ B′C′D	X					X			
√ A′BD′		X	X						
A′BC			X	X					
BCD				X					X
ACD								X	X
	√	√	√		√	√	√	√	

In the table, the prime implicants are listed in the 1st column and Xs are placed against the corresponding minterms. The completed prime implicant table is now inspected for the columns containing only a single X. As in Figure above, the minterm 1 is represented by only a single prime implicant B′C′D, and only a single X in that column, it should be marked as well as the corresponding column should be marked. Similarly, the prime implicants AB′ and
AB′D′ are marked. These are the essential prime implicants as they are absolutely necessary to form the minimized Boolen expression. Now all the other minterms corresponding to these prime implicants are marked at the end of the columnsi.e., the minterms 1, 4, 6, 8, 9, 10, and 11 are marked. Note that the terms A′BC, BCD, and ACD are not marked. So they are not the essential prime implicants. However, the minterms 7 and 15 are still unmarked and both of them are covered by the term BCD and are included in the Boolean expression.
Therefore, the simplified Boolen expression of the given function can be written as
F = AB′ + B′C′D + A′BD′ + BCD.

The simplified expressions derived in the preceeding example are in the sum of products form. The Quine-McClusky method can also be adopted to derive the simplified expression in product of sums form. In the Karnaugh map method the complement of the function was considered by taking 0s from the initial list of the minterns. Similarly the tabulation method or Quine-McClusky method may be carried out by considering the 0s of the function to derive the sum of products form. Finally, by making the complement again, we obtain the simplified expression in the form of product of sums.

A function with don’t-care conditions can be simplified by the tabulation method with slight modification. The don’t-care conditions are to be included in the list of minterms while determining the prime implicants. This allows the derivation of prime implicants with the least number of literals. But the don’t-care conditions are excluded in the list of minterms when the prime implicants table is prepared, because these terms do not have to be covered by the selected prime implicants.

5.3 Chapter Review Questions

1. **Simplify the following expressions:**
   a. A $\bar{B}$ $\bar{C}$+ $\bar{A}$ $\bar{B}$ $\bar{C}$+ $\bar{A}$ B $\bar{C}$ + $\bar{A}$ $\bar{B}$ C
   b. ABC + $\bar{A}$ BC + A $\bar{B}$ C + AB $\bar{C}$ + A $\bar{B}$ $\bar{C}$ + $\bar{A}$B $\bar{C}$+ $\bar{A}$ $\bar{B}$ $\bar{C}$
   c. A(A + B + C) ($\bar{A}$ + B + C) (A + $\bar{B}$ + C) (A + B + $\bar{C}$)
   d. (A + B + C) (A + $\bar{B}$ + $\bar{C}$) (A + B + $\bar{C}$) (A + $\bar{B}$ + C)
2. Simplify the following Boolean expressions using Boolean technique:
   a. AB + A (B + C) + B (B + C)
   b. AB(C + B $\bar{D}$) ($\overline{AB}$)
   c. A + AB + A$\bar{B}$ C
   d. ($\bar{A}$ + B)C + ABC
   e. A$\bar{B}$ C (BD + CDE) + A$\bar{C}$
   f. BD + B (D + E) + $\bar{D}$ (D + F)

3. Simplify the expression the following expressions using K-MAP
   a. F = $\bar{A}$BC + A$\bar{B}$$\bar{C}$ + ABC + AB$\bar{C}$.
   b. F = $\bar{A}$$\bar{B}$C + $\bar{A}$BC + $\bar{A}$B$\bar{C}$ + A$\bar{B}$C + ABC.
   c. Z = f(A,B,C) = $\bar{A}$ $\bar{B}$ $\bar{C}$+ $\bar{A}$B + AB $\bar{C}$ + AC
   d. Z = f(A,B,C) = $\bar{A}$B + B $\bar{C}$ + BC + A $\bar{B}$ $\bar{C}$
   e. Z = f(A,B,C) = $\bar{A}$ $\bar{B}$ $\bar{C}$+ $\bar{A}$B + AB $\bar{C}$ + AC
4. What are the don't-care conditions?
5. What are the advantages of the tabulation method?
6. Draw a Karnaugh map for a four-variable Ex-OR function and derive its expression.
7. How does a Karnaugh map differ from a truth table?
8. What kind of network is developed by sum of the products?
9. A combinational switching network has four inputs A, B, C, and D, and one output Z. The output is to be 0, if the input combination is a valid Excess-3 coded decimal digit. If any other combinations of inputs appear, the output is to be 1. Implement the network using basic gates

CHAPTER SIX

DIGITAL CIRCUITS

Chapter objectives

1. *differentiate between sequential and combination circuits*
2. *explain the design of various combinational circuits*
3. *describe Combinational Logic with MSI AND LSI*
4. *explain the design of various sequential circuits*

6.1 Digital Circuits

Digital circuits can be subdivided into two main categories;

i. Combination circuits
ii. Sequential circuits

Combination circuits are types of circuits where by the outputs depends on the inputs at that instance of time. This type of circuit has no memory devices.
Sequential circuits have outputs which depend on inputs at that instance of time as whereas outputs from other circuits which serve as input to that circuit. These circuits therefore serve as inputs. This means that sequential circuits require memory elements. Both Sequential and Combination circuits make micro architecture of the standard micro processors as well as custom specific integrated circuits (ICs).

6.1 1 Combinational circuit

Combination circuits perform data transforms for example logic operations and arithmetic operations (XOR, AND, complement operations etc). These circuits' devices include adders, multiplexers, de-multiplexors, parity encoders, comparators etc.

DESIGN PROCEDURE
Any combinational circuit can be designed by the following steps of design procedure.

1. The problem is stated.
2. Identify the input variables and output functions.
3. The input and output variables are assigned letter symbols.
4. The truth table is prepared that completely defines the relationship between the input variables and output functions.
5. The simplified Boolean expression is obtained by any method of minimization—algebraic method, Karnaugh map method, or tabulation method.
6. A logic diagram is realized from the simplified expression using logic gates

It is very important that the design problem or the verbal specifications be interpreted correctly to prepare the truth table. Sometimes the designer must use his intuition and experience to arrive at the correct interpretation. Word specification are very seldom exact and complete. Any wrong interpretation results in incorrect truth table and combinational circuit.
Varieties of simplification methods are available to derive the output Boolean functions from the truth table, such as the algebraic method, the Karnaugh map, and the tabulation method. However, one must consider different aspects, limitations, restrictions, and criteria for a particular design application to arrive at suitable algebraic expression. A practical design approach should consider constraints like—(1) minimum number of gates, (2) minimum number of outputs, (3) minimum propagation time of the signal through a circuit, (4) minimum number of interconnections, and (5) limitations of the driving capabilities of each logic gate. Since the importance of each constraint is dictated by the particular application, it is difficult to make all these criteria satisfied simultaneously and also difficult to make a general statement on the process of achieving an acceptable simplifi cation. However, in most cases, first the simplified Boolean expression at standard form is derived and then other constraints are taken care of as far as possible for a particular application.

6.1 2 ADDERS

Various information-processing jobs are carried out by digital computers. Arithmetic operations are among the basic functions of a digital computer. Addition of two binary digits is the most basic arithmetic operation. The simple addition consists of four possible elementary operations, which are 0+0 = 0, 0+1 = 1, 1+0 = 1, and 1+1 = 10. The first three operations produce a sum of one digit, but the fourth operation produces a sum consisting of two digits. The higher significant bit of this result is called the carry. A combinational circuit that performs the addition of two bits as described above is called a half-adder. When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher-order pair of significant bits. Here the addition operation involves three bits—the augend bit, addend bit, and the carry bit and produce a sum result as well as carry. The combinational circuit performing this type of addition operation is called a full-adder. In circuit development two half-adders can be employed to form a full-adder.

Design of Half adder

A **half adder** adds two one-bit binary numbers *A* and *B*. It has two outputs, *S* and *C* (the value theoretically carried on to the next addition); the final sum is 2*C* + *S*. The simplest half-adder design, pictured on the right, incorporates an XOR gate for *S* and an AND gate for *C*. Half adders cannot be used compositely, given their incapacity for a carry-in bit.

Truth Table

A	B	SUM(s)	CARRY(c)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table in Figure 5.2, it can be seen that the outputs S and C functions are similar to Exclusive-OR and AND functions respectively.
The Boolean expressions are

S = A′B+AB′ and
C = AB.

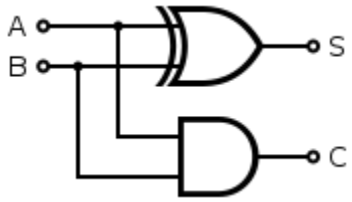


Fig. shows the logic diagram to implement the half-adder circuit

Full adder

Design of Full adder

The half adder has two inputs but it has no provision to add a carry coming from the lower order bits i.e. when the multi-bit addition is being performed. A third input is required for the carry generated. This modified circuit is known as the full adder circuit. The full adder circuit can be designed using either the AND, OR and XOR or using NAND, and OR gates. A combinational circuit of full-adder performs the operation of addition of three bits—the augend, addend, and previous carry, and produces the outputs sum and carry

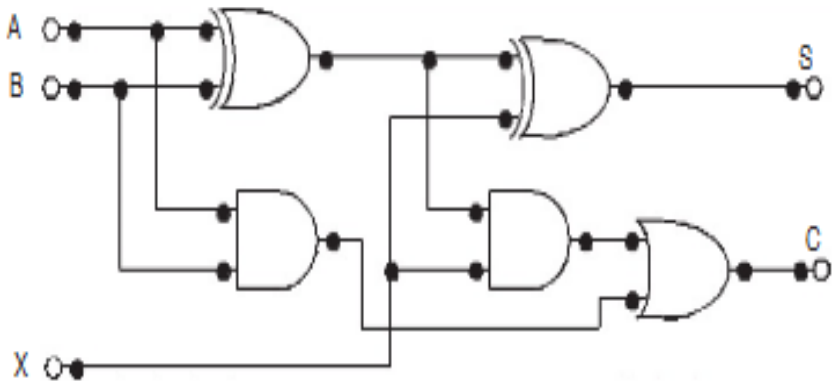
Let us designate the input variables augend as A, addend as B, and previous carry as X, and outputs sum as S and carry as C. As there are three input variables, eight different input combinations are possible. The truth table is shown in below according to its functions

Inputs			**Outputs**	
A	***B***	***X***	***C***	***S***
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

The Boolean expressions of S and C are modified as follows
S = X′A′B + X′AB′ + XA′B′ + XAB
= X′ (A′B + AB′) + X (A′B′ + AB)

= X′ (A B) + X (A B)′
= X A B
C = AB + BX + AX = AB + X (A + B)
= AB + X (AB + AB′ + AB + A′B)
= AB + X (AB + AB′ + A’B)
= AB + XAB + X (AB′ + A’B)
= AB + X (A B)

Logic diagram according to the modified expression is shown below



Notice that the full-adder developed in Figure 5.7 consists of two 2-input AND gates, two 2-input XOR (Exclusive-OR) gates and one 2-input OR gate. This contains a reduced number
of gates as well as type of gates

6.2 Combinational Logic with MSI AND LSI

The purpose of simplification of Boolean functions is to obtain an algebraic expression with less number of literals and less numbers of logic gates. This results in low-cost circuit implementation. The design procedure for combinational circuits as described in the preceding sections is intended to minimize the number of logic gates to implement a given function.

This classical procedure realizes the logic circuit with fewer gates with the assumption that the circuit with fewer gates will cost less. However, in practical design, with the arrival of a variety of integrated circuits (IC), this concept is always true.

Since one single IC package contains several number of logic gates, it is economical to use as many of the gates from an already used package, even if the total number of gates is increased by doing so. Moreover, some of the interconnections among the gates in many ICs are internal to the chip and it is more economical to use such types of ICs to minimize the external interconnections or wirings among the IC pins as much as possible. For design with integrated circuits, it is not the count of logic gates that reduces the cost, but the number and type of IC packages used and the number of interconnections required to implement certain functions.

Though the classical method constitutes a general procedure, is very easy to understand, and certain to produce a result, on numerous occasions it does not achieve the best possible combinational

circuit for a given function. Moreover, the truth table and simplification procedure in this method become too cumbersome if the number of input variables is excessively large and the final circuit obtained may require a relatively large number of ICs and interconnecting wires. In many cases the alternative design approach can lead to a far better combinational circuit for a given function with comparison to the classical method.

The alternate design approach depends on the particular application and the ingenuity as well as experience of the designer. To handle a practical design problem, it should always be investigated which method is more suitable and efficient.
Design approach of a combinational circuit is first to analysis and to find out whether the function is already available as an IC package. Numerous ICs are commercially available, some of which perform specific functions and are commonly employed in the design of digital computer system. If the required function is not exactly matched with any of the commercially available devices, a designer will formulate a method to incorporate the ICs that are nearly suitable to the function.
A large number of integrated circuit packages are commercially available nowadays.
They can be widely categorized into three groups;

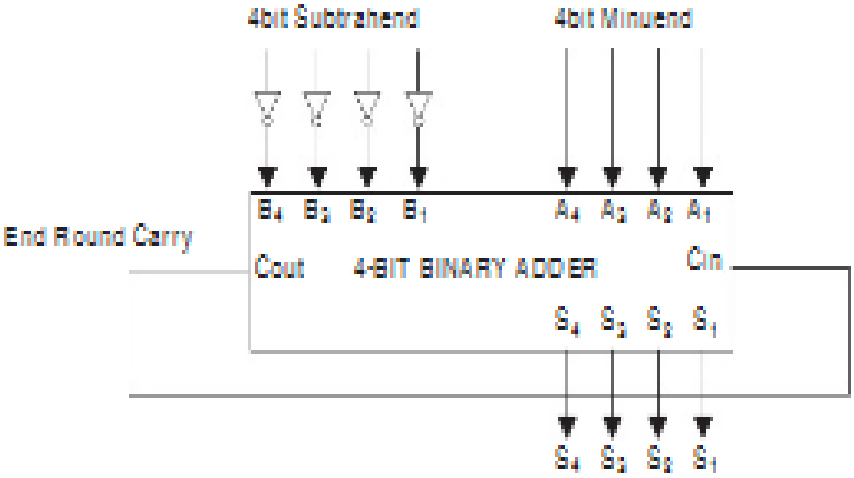
1. SSI or small scale integration where the number of logic gates is limited to ten in one IC package,
2. MSI or medium scale integration where the number of logic gates is eleven to one hundred in one IC package.
3. LSI or large-scale integration containing more than one hundred gates in one package. Some of them are fabricated for specific functions. VLSI or very large scale integration IC packages are also introduced, which perform dedicated functions achieving high circuit space reduction and interconnection reduction

6.2 1 Magnitude Comparator

A magnitude comparator is one of the useful combinational logic networks and has wide applications. It compares two binary numbers and determines if one number is greater than, less than, or equal to the other number. It is a multiple output combinational logic circuit. If two binary numbers are considered as A and B, the magnitude comparator gives three outputs for
A > B, A < B, and A = B.
For comparison of two n-bit numbers, the classical method to achieve the Boolean expressions requires a truth table of 22n entries and becomes too lengthy and cumbersome.
It is also desired to have a digital circuit possessing with a certain amount of regularity, so that similar circuits can be applied for the comparison of any number of bits. Digital functions that follow an inherent well-defined regularity can usually be developed by means of algorithmic procedure if it exists. An algorithm is a process that follows a finite set of steps to arrive at the solution to a problem.

Four-bit Binary Parallel Subtractor

It is interesting to note that a 4-bit binary adder can be employed to obtain the 4-bit binary subtraction. Previously we saw how binary subtraction can be achieved using
1's complement or 2's complement. By 1's complement method, the bits of subtrahend are complemented and added to the minuend. If any carry is generated it is added to the sum output. Figure below demonstrates the subtraction of B_4, B_3, B_2, B_1 from A_4, A_3, A_2, A_1. Each bit of
B_4, B_3, B_2, B_1 is first complemented by using INVERTER gates and added to A_4, A_3, A_2, A_1 by a 4-bit binary adder. End round carry is again added using the C in pin of the IC.

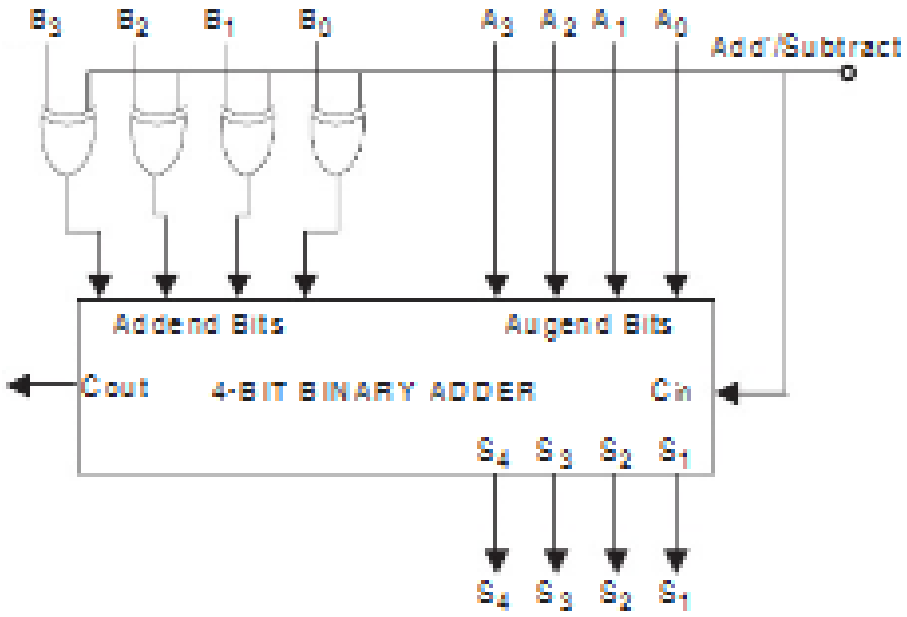


Four-bit Binary Parallel Adder/Subtractor

Due to the property of the 4-bit binary adder that it can perform the subtraction operation with external inverter gates; a single combinational circuit may be developed that can perform addition as well as the subtraction introducing a control bit. A little modification helps to obtain this dual operation. Figure below demonstrates this dual-purpose combinational logic circuit.

XOR gates are used at addend or subtrahend bits when one of the inputs of the XOR gate is connected to the ADD/SUBTRACT terminal, which is acting as control terminal. The same terminal is connected to Cin. When this terminal is connected to logic 0 the combinational circuit behaves like a 4-bit full adder, as at this instant Cin is logic low and XOR gates are acting as buffers whose outputs are an uncomplemented form of inputs. If logic 1 is applied to the ADD/SUBTRACT terminal, the XOR gates behave like INVERTER gates and data bits are complemented. The 4-bit adder now performs the addition operation of data A3A2A1A0 with complemented form of data B3B2B1B0 as well as with a single bit 1, as Cin is now logic

1. This operation is identical to a subtraction operation using 2’s complment



6.2 2 Multiplexers or Data Selectors

A multiplexer is one of the important combinational circuits and has a wide range of applications. The term multiplex means "many into one." Multiplexers transmit large numbers of information channels to a smaller number of channels. A digital multiplexer is a combinational circuit that selects binary information from one of the many input channels and transmits to a single output line. That is why the multiplxers are also called data selectors. The selection of the particular input channel is controlled by a set of select inputs.

A digital multiplexer of 2n input channels can be controlled by n numbers of select lines and an input line is selected according to the bit combinations of select lines

A 4-to-1 line multiplexer is defined as the multiplexer consisting of four input channels and information of one of the channels can be selected and transmitted to an output line according to the select inputs combinations. Selection of one of the four input channels is possible by two selection inputs. The truth table is given below. Input channels I_0, I_1, I_2, and I_3 are selected by the combinations of select inputs S_1 and S_0. The circuit diagram is shown in figure (a) below. To demonstrate the operation, let us consider that select input combination S_1S_0 is 01. The AND gate associated with I_1 will have two of inputs equal to logic 1 and a third input is connected to I_1. Therefore, output of this AND gate is according to the information provided by channel I_1. The other three AND gates have logic 0 to at least one of their inputs which makes their outputs to logic 0. Hence, OR output (Y) is equal to the data provided by the channel I_1. Thus, information from I1 is available at Y. Normally a multiplexer has an ENABLE input to also control its operation. The ENABLE input (also called STROBE) is useful to expand two or more multiplexer ICs to a digital multiplexer with a larger number of inputs, which will be demonstrated in a later part of this section.

A multiplexer is often abbreviated as MUX. Its block diagram is shown in Figure (b) below

Truth table

S_1	S_0	I_0	I_1	I_2	I_3	Y
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

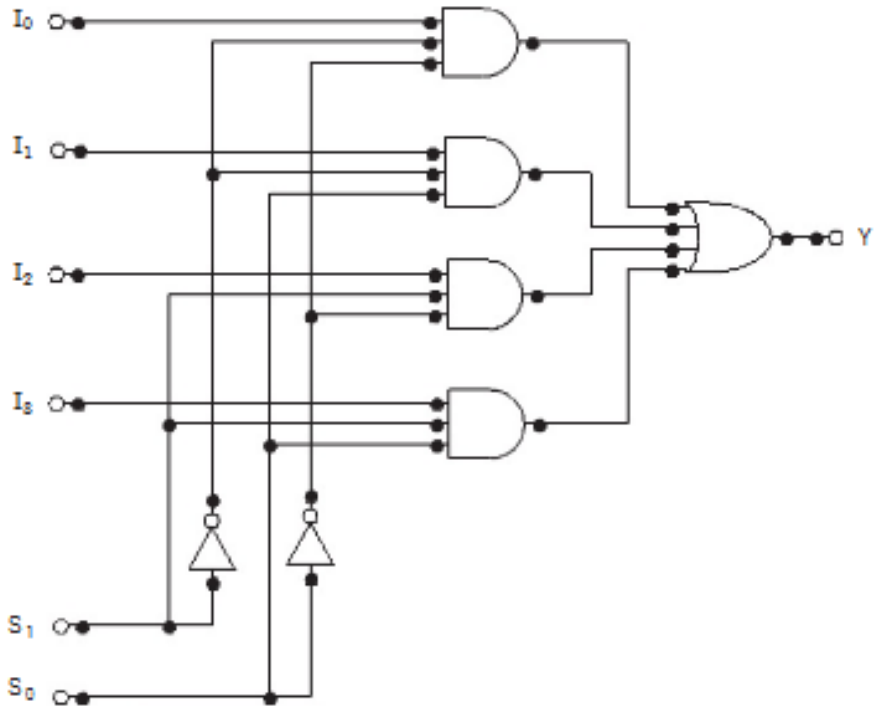


Figure (a) multiplexer circuit diagram

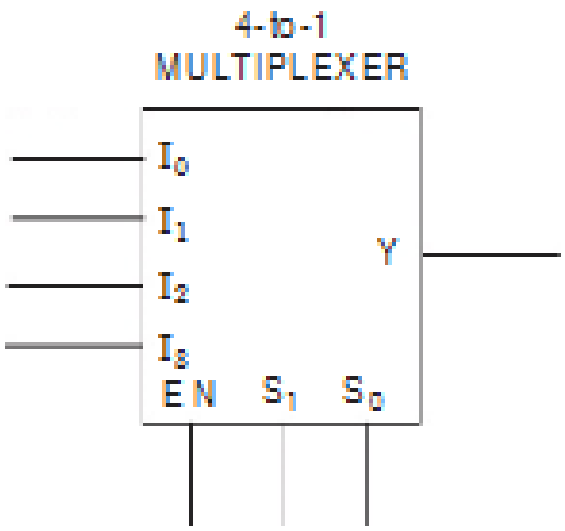


Figure (b) multiplexer block diagram

6.2 3 Encoders

An encoder is a combinational network that performs the reverse operation of the decoder.

An encoder has 2^n or less numbers of inputs and n output lines. The output lines of an encoder generate the binary code for the 2n input variables. Figure 5.70 illustrates an eight inputs/three outputs encoder. It may also be referred to as an octal-to-binary encoder where binary codes are generated at outputs according to the input conditions. The truth table is given in Figure 5.71.

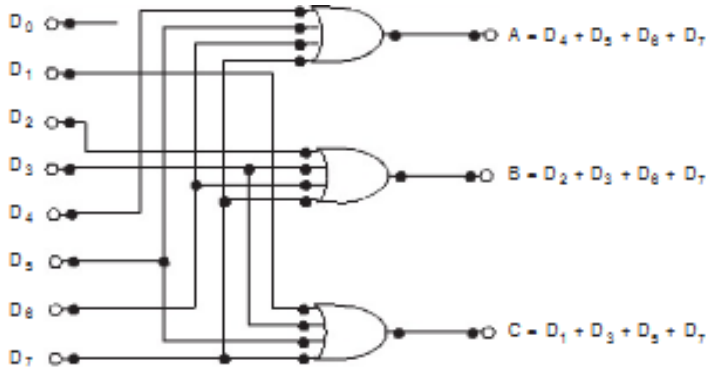


Figure 5.70

Inputs								*Outputs*		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	*A*	*B*	*C*
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

Figure 5.71

6.3 Sequential Logic Circuits

Though Combinational digital circuits are very important, they constitute only a part of digital systems. The other major aspect of a digital system is the analysis and design of sequential digital circuits. However, sequential circuit design depends, greatly, on the combinational circuit design. logic circuits whose outputs at any instant of time depend on the present inputs as well as on the past outputs are called sequential circuits. In sequential circuits, the output signals are fed back to the input side. A block diagram of a sequential circuit is shown below

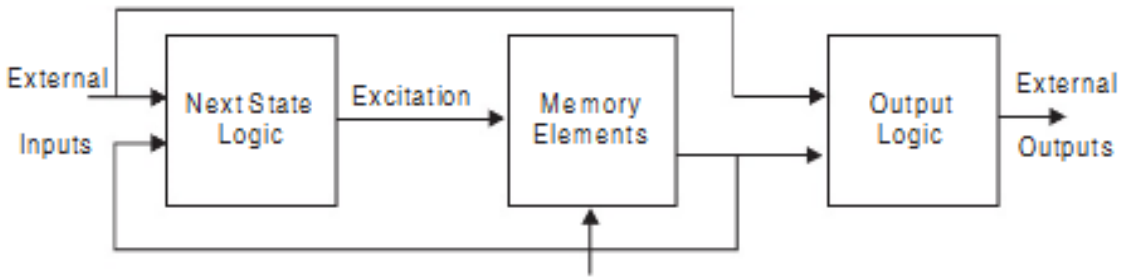


Fig. Block diagram of a sequential circuit

The Figure above consists of combinational circuits, which accept digital signals from external inputs and from outputs of memory elements and generates signals for external outputs and for inputs to memory elements, referred to as excitation

A memory element is a medium in which one bit of information (0 or 1) can be stored or retained until necessary, and thereafter its contents can be replaced by a new value.

The contents of memory elements can be changed by the outputs of combinational circuits that are connected to its input.

Combinational circuits are often faster than sequential circuits since the combinational circuits do not require memory elements whereas the sequential circuit needs memory elements to perform its operations in sequence.

Sequential circuits are broadly classified into two main categories;

1. Synchronous or clocked
2. Asynchronous or unclocked sequential circuits, depending on the timing of their signals.

A sequential circuit whose behavior can be defined from the knowledge of its signal at discrete instants of time is referred to as a *synchronous sequential circuit*. In these systems, the memory elements are affected only at discrete instants of time. The synchronization is achieved by a timing device known as a system clock, which generates a periodic train of clock pulses as shown in Figure (a). The outputs are affected only with the application of a clock pulse. The rate at which the master clock generates pulses must be slow enough to permit the slowest circuit to respond. This limits the speed of all circuits. Synchronous circuits have gained considerable domination and wide popularity.

A sequential circuit whose behavior depends upon the sequence in which the input signals change is referred to as an *asynchronous sequential circuit*. The output will be affected whenever the input changes. The commonly used memory elements in these circuits are time-delay devices. There is no need to wait for a clock pulse. Therefore, in general, asynchronous circuits are faster than synchronous sequential circuits. However, in an asynchronous circuit, events are allowed to occur without any synchronization. And in such a case, the system becomes unstable. Since the designs of asynchronous circuits are more tedious and difficult, their uses are rather limited. The memory elements used in sequential circuits are flip-flops which are capable of storing binary information

6.3 1 Flip-Flops

The basic 1-bit digital memory circuit is known as a flip-flop. It can have only two states, either the 1 state or the 0 state. A flip-flop is also known as a bistable multivibrator. Flip-flops can be obtained by using NAND or NOR gates. The general block diagram representation of a

flip-fl op is shown below. It has one or more inputs and two outputs. The two outputs are complementary to each other. If Q is 1 i.e., Set, then Q' is 0; if Q is 0i.e. Reset, then

Q' is 1. That means Q and Q' cannot be at the same state simultaneously. If it happens by any chance, it violates the definition of a flip-flop and hence is called an undefined condition.

Normally, the state of Q is called the state of the flip-flop, whereas the state of Q' is called

The complementary state of the flip-flop. When the output Q is either 1 or 0, it remains in that state unless one or more inputs are excited to effect a change in the output. Since the output of the flip-flop remains in the same state until the trigger pulse is applied to change the state, it can be regarded as a memory device to store one binary bit.

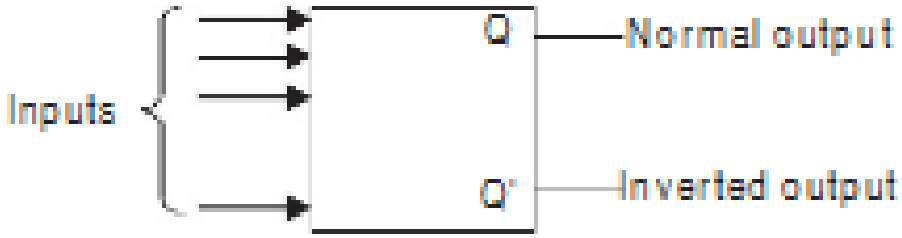


Fig. Block diagram of a flip-flop.

Types of Flip-Flops

There are different types of flip-flops depending on how their inputs and clock pulses cause transition between two states. Here we will discuss four different types of flip-flops:

- S-R flip-flops
- J-K, flip-flops

S-R (Set-Reset) Flip-flop

An S-R flip-flop has two inputs named Set (S) and Reset (R), and two outputs Q and Q'. The outputs are complement of each other,i.e., if one of the outputs is 0 then the other should be 1. This can be implemented using NAND or NOR gates. The block diagram of an S-R flip-flop is shown in Figure below.

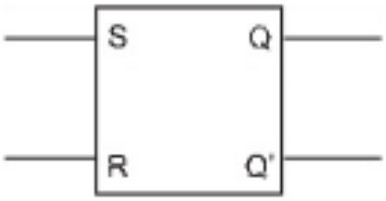
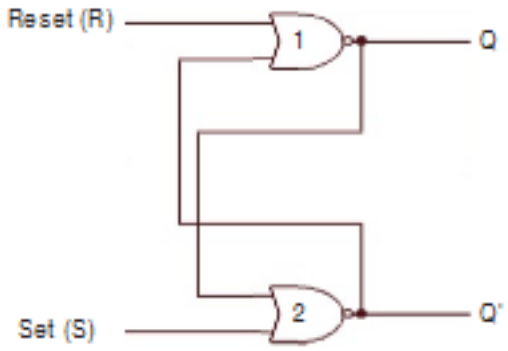


Fig. block diagram of an S-R flip-flop

S-R Flip-flop Based on NOR Gates

An S-R flip-flop can be constructed with NOR gates at ease by connecting the NOR gates back to back as shown in Figure below. The cross-coupled connections from the output of gate 1 to the input of gate 2 constitute a feedback path. This circuit is not clocked and is classified as an asynchronous sequential circuit.

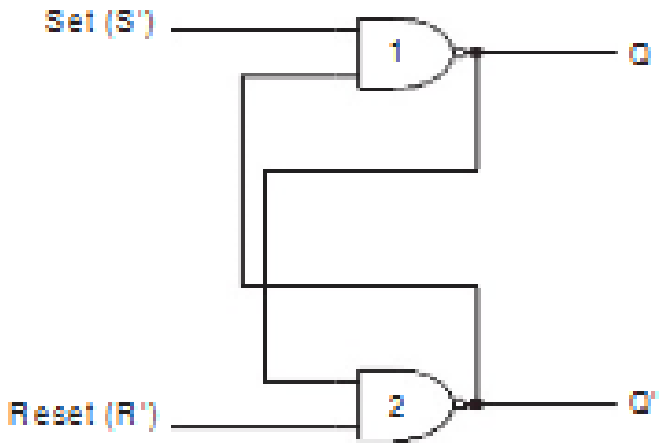


To analyze the circuit shown in above, we have to consider the fact that the output of a NOR gate is 0 if any of the inputs are 1, irrespective of the other input. The output is 1 only if all of the inputs are 0. The outputs for all the possible conditions as shown in the table in below

Inputs		*Outputs*		*Action*
S	*R*	Q_{n+1}	Q'_{n+1}	
0	0	Q_n	Q'_n	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Forbidden (Undefi ned)
0	0	–	–	Indeterminate

S'-R' Flip-flop Based on NAND Gates

An S'-R' flip-flop can be constructed with NAND gates by connecting the NAND gates back to back as shown in Figure below. The operation of the S'-R' flip-fl op can be analyzed in a similar manner as that employed for the NOR-based S-R flip-flop. This circuit is also not clocked and is classified as an asynchronous sequential circuit.



To analyze the circuit shown above we have to remember that a LOW at any input of a NAND gate forces the output to be HIGH, irrespective of the other input. The output of a NAND gate is 0 only if all of the inputs of the NAND gate are 1. The outputs for all the possible conditions as shown in the table below

Inputs		Outputs		Action
S'	R'	Q_{n+1}	Q'_{n+1}	
1	1	Q_n	Q_n	No change
1	0	0	1	Reset
0	1	1	0	Set
0	0	1	1	Forbidden (Undefined)
1	1	–	–	Indeterminate

CLOCKED S-R FLIP-FLOP

Generally, synchronous circuits change their states only when clock pulses are present. The operation of the basic flip-flop can be modified by including an additional input to control the behavior of the circuit. Such a circuit is shown in Figure (a) below. This circuit consists of two AND gates. The clock input is connected to both of the AND gates, resulting in LOW outputs when the clock input is LOW. In this situation the changes in S and R inputs will not affect the state (Q) of the flip-flop. On the other hand, if the clock input is HIGH, the changes in S and R will be passed over by the AND gates and they will cause changes in the output (Q) of the fl ip-fl op. This way, any information, either 1 or 0, can be stored in the flip-flop by applying a HIGH clock input and be retained for any desired period of time by applying a LOW at the clock input. This type of flip-flop is called a clocked S-R fl ip-flop. Such a clocked S-R flip-flop made up of two AND gates and two NOR gates is shown in Figure (b)

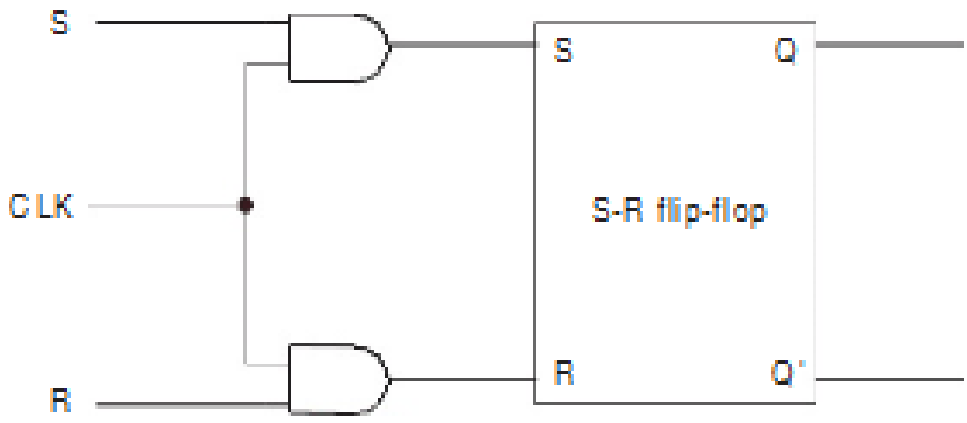


Figure (a) Block diagram of a clocked S-R flip-flop

The logic symbol of the S-R flip-flop is shown in in theFigure below. It has three inputs: S, R, and CLK. The CLK input is marked with a small triangle. The triangle is a symbol that denotes the fact that the circuit responds to an edge or transition at CLK input

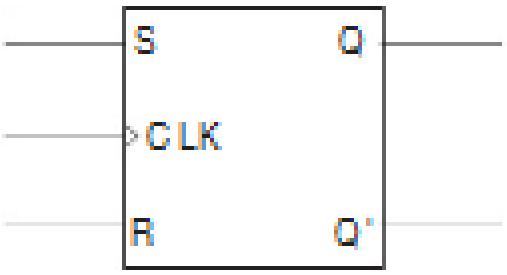


Fig. Logic symbol of a clocked S-R flip-fl op

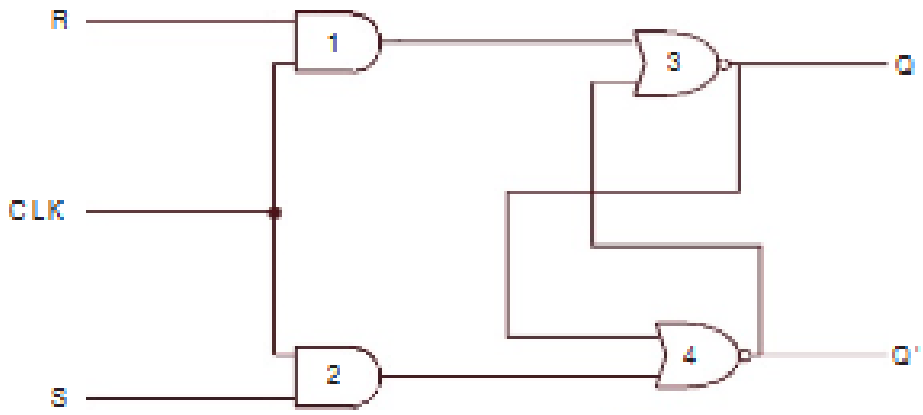


Figure (b) A clocked NOR-based S-R flip-flop

The same S-R flip-flop can be constructed using the basic NAND latch and two other NAND gates as shown in Figure below. The S and R inputs control the states of the
flip-flop in the same way as described earlier for the unclocked S-R fl ip-fl op. However, the
flip-flop only responds when the clock signal occurs. The clock pulse input acts as an enable signal for the other two inputs. As long as the clock input remains 0 the outputs of NAND gates 1 and 2 stay at logic 1. This 1 level at the inputs of the basic NAND-based S-R fl ip-
flop retains the present state

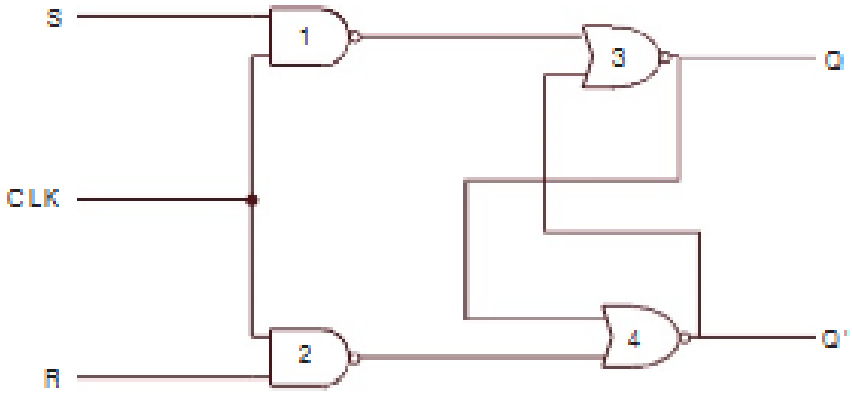


Fig. A clocked NAND-based S-R fl ip-flop

Assuming that the inputs do not change during the presence of the clock pulse, we can express the working of the S-R flip-flop in the form of the truth table below. Here, Sn and Rn denote the inputs and Qn the output during the bit time n
Qn+1 denotes the output after the pulse passes,i.e., in the bit timen + 1

Inputs		Output
S_n	R_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	–

J-K FLIP-FLOP

A J-K flip-flop has very similar characteristics to an S-R flip-flop. The only difference is that the undefined condition for an S-R flip-flop, i.e., Sn= R n = 1 condition, is also included in this case. Inputs J and K behave like inputs S and R to set and reset the flip-flop respectively. When J = K = 1, the flip-flop is said to be in a toggle state, which means the output switches to its complementary state every time a clock passes.

The data inputs are J and K, which are ANDed with Q' and Q respectively to obtain the inputs for S and R respectively. A J-K fl ip-fl op thus obtained is shown in Figure

Below

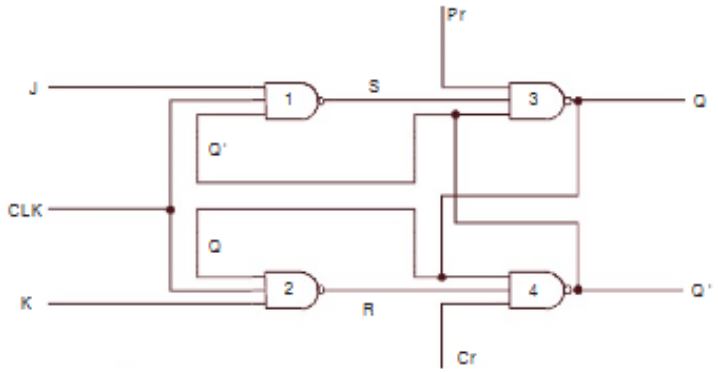


Figure 7.30 A J-K flip-fl op using NAND gates.

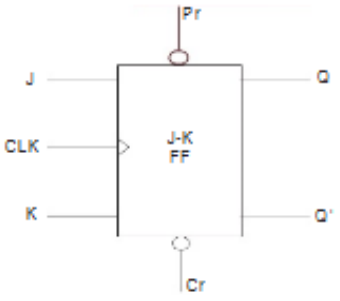


Figure 7.31 Logic symbol of a J-K flip-fl op.

The truth table of such a flip-flop is as follows

Inputs		*Output*
J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Q'_n

Race-around Condition of a J-K Flip-fl op

The inherent difficulty of an S-R flip-flop (i.e., S = R = 1) is eliminated by using the feedback connections from the outputs to the inputs of gate 1 and gate 2 as shown in J-K Flip-flop Figures above were formed with the assumption that the inputs do not change during the clock pulse (CLK = 1). But the consideration is not true because of the feedback connections

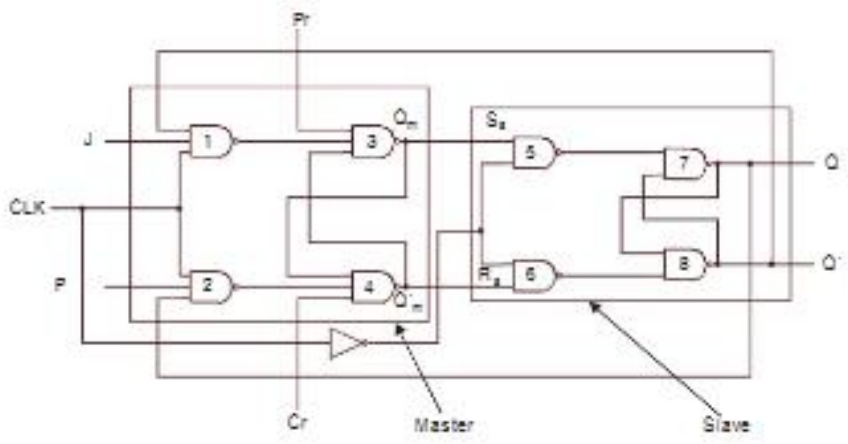
Consider, for example, that the inputs are J = K = 1 and Q = 1, and a pulse is applied at the clock input. After a time interval Δt equal to the propagation delay through two NAND gates in series, the outputs will change to Q = 0. So now we have J = K = 1 and Q = 0. After another time interval of Δt the output will change back to Q = 1. Hence, we conclude that for the time duration of t_P of the clock pulse, the output will oscillate between 0 and 1. Hence, at the end of the clock pulse, the value of the output is not certain. This situation is referred to as a *race-around condition*.

Generally, the propagation delay of TTL gates is of the order of nanoseconds. So if the clock pulse is of the order of microseconds, then the output will change thousands of times within the clock pulse. This race-around condition can be avoided if $t_p<\Delta t < T$. Due to the small propagation delay of the ICs it may be difficult to satisfy the above condition. A more practical way to avoid the problem is to use the master-slave (M-S) configuration.

Master-Slave J-K Flip-fl op

A master-slave (M-S) flip-fl op is shown in Figure below. Basically, a master-slave flip-flop is a system of two flip-flops—one being designated as master and the other is the slave.

From the figure we see that a clock pulse is applied to the master and the inverted form of the same clock pulse is applied to the slave



When CLK = 1, the first flip-flop (i.e., the master) is enabled and the outputs Qm and Q'm respond to the inputs J and K according to the table shown in Figure above. At this time the second flip-flop (i.e., the slave) is disabled because the CLK is LOW to the second flip-flop. Similarly, when CLK becomes LOW, the master becomes disabled and the slave becomes active, since now the CLK to it is HIGH. Therefore, the outputs Q and Q' follow the outputs Qm and Q'm respectively. Since the second fl ip-fl op just follows the fi rst one, it is referred to as a slave and the fi rst one is called the master. Hence, the configuration is referred to as a master-slave (M-S) flip-flop

6.3 2 Triggering Of Flip-Flops

Flip-flops are synchronous sequential circuits. This type of circuit works with the application of a synchronization mechanism, which is termed as a clock. Based on the specific interval or point in the clock during or at which triggering of the flip-fl op takes place, it can be classified into two different types—level triggering and edge triggering.

A clock pulse starts from an initial value of 0, goes momentarily to 1, and after a short interval, returns to the initial value.

Level Triggering of Flip-fl ops

If a flip-flop gets enabled when a clock pulse goes HIGH and remains enabled throughout the duration of the clock pulse remaining HIGH, the flip-flop is said to be a level triggered flip-flop. If the flip-flop changes its state when the clock pulse is positive, it is termed as a positive level triggered flip-flop. On the other hand, if a NOT gate is introduced in the clock input terminal of the flip-flop, and then the fl ip-flop changes its state when the clock pulse is negative, it is termed as a negative level triggered flip-fl op.

The main drawback of level triggering is that, as long as the clock pulse is active, the fl ip-fl op changes its state more than once or many times for the change in inputs. If the inputs do not change during one clock pulse, then the output remains stable. On the other hand, if the frequency of the input change is higher than the input clock frequency, the output of the flip-flop undergoes multiple changes as long as the clock remains active. This can be overcome by using either master-slave fl ip-fl ops or the edge-triggered fl ip-fl op.

Edge-triggering of Flip-fl ops

A clock pulse goes from 0 to 1 and then returns from 1 to 0. Figure 7.46 shows the two transitions and they are defined as the positive edge (0 to 1 transition) and the negative edge (1 to 0 transition). The term edge-triggered means that the flip-flop changes its state only at either the positive or negative edge of the clock pulse

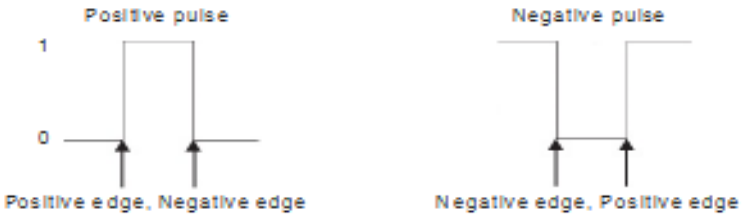


Figure 7.46 Clock pulse transition.

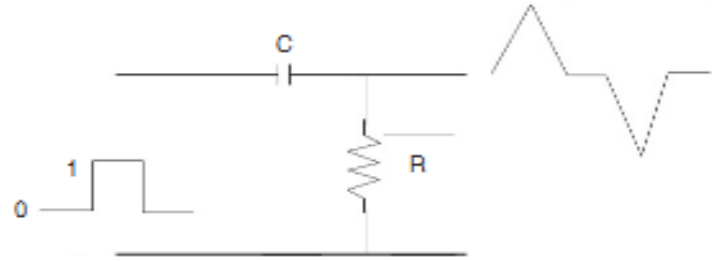
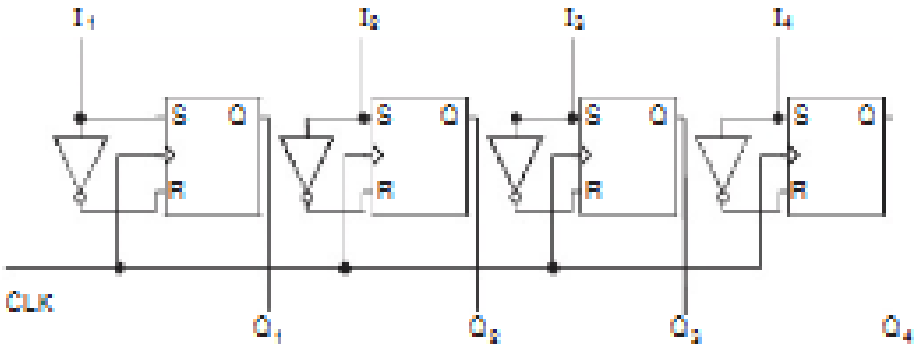


Figure 7.47 RC differentiator circuit for edge triggering.

6.3 3 Registers

Register is a group of binary storage cells capable of holding binary information. A group of fl ip-fl ops constitutes a register, since each fl ip-fl op can work as a binary cell. An n-bit register has n fl ip-flops and is capable of holding n-bits of information. In addition to flip-flops a register can have a combinational part that performs data-processing tasks.

Various types of registers are available in MSI circuits. The simplest possible register is one that contains no external gates, and is constructed of only flip-flops. The figure below shows such a type of register constructed of four S-R fl ip-fl ops, with a common clock pulse input. The clock pulse enables all the flip-flops at the same instant so that the information available at the four inputs can be transferred into the 4-bit register. All the flip-fl ops in a register should respond to the clock pulse transition. Hence they should be either of the edge-triggered type or the master-slave type. A group of fl ip-fl ops sensitive to the pulse duration is commonly called a gated latch. Latches are suitable to temporarily store binary information that is to be transferred to an external destination. They should not be used in the design of sequential circuits that have feedback connections



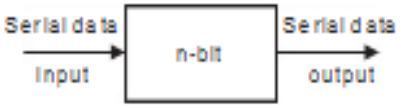
Shift Register

A register capable of shifting its binary contents either to the left or to the right is called a shift register. The shift register permits the stored data to move from a particular location to some other location within the register. Registers can be designed using discrete flip-flops (S-R, J-K, and D-type).

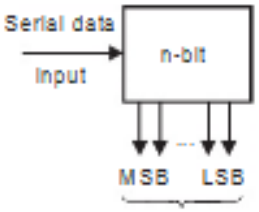
The data in a shift register can be shifted in two possible ways: (a) serial shifting and (b) parallel shifting. The serial shifting method shifts one bit at a time for each clock pulse in a serial manner, beginning with either LSB or MSB. On the other hand, in parallel shifting operation, all the data (input or output) gets shifted simultaneously during a single clock pulse. Hence, we may say that parallel shifting operation is much faster than serial shifting operation.

There are two ways to shift data into a register (serial or parallel) and similarly two ways to shift the data out of the register. This leads to the construction of four basic types of registers. All of the four configurations are commercially available as TTL MSI/LSI circuits. They are:

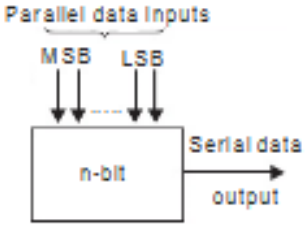
1. Serial in/Serial out (SISO) – 54/74L91, 8 bits
2. Serial in/Parallel out (SIPO) – 54/74164, 8 bits
3. Parallel in/Serial out (PISO) – 54/74265, 8 bits
4. Parallel in/Parallel out (PIPO) – 54/74198, 8 bits.

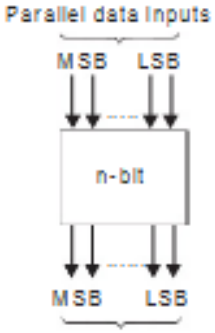


(a) Serial in/Serial out.



(b) Serial in/Parallel out.





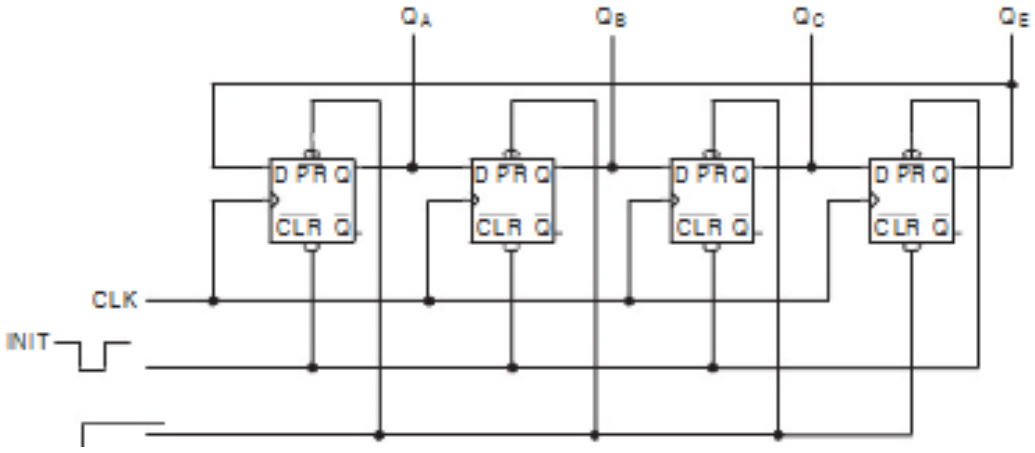
Shift Register Counters

Shift registers may be arranged to form different types of counters. These shift registers Use feedback, where the output of the last fl ip-flop in the shift register is fed back to the first flip-flop. Based on the type of this feedback connection, the shift register counters are classified as (i) ring counter and (ii) twisted ring or Johnson or Shift counter.

Ring Counter

It is possible to devise a counter-like circuit in which each flip-flop reaches the state Q = 1 for exactly one count, while for all other counts Q = 0. Then Q indicates directly an occurrence of the corresponding count. Actually, since this does not represent binary numbers, it is better to say that the outputs of the flip-flops represent a code. Such a circuit is shown in Figure below, which is known as a ring counter. The Q output of the last stage in the shift register is fed back as the input to the first stage, which creates a ring-like structure.

Hence a ring counter is a circular shift register with only one flip-flop being set at any particular time and all others being cleared. The single bit is shifted from one flip-flop to the other to produce the sequence of timing signals. Such encoding where there is a single 1 and the rest of the code variables are 0, is called a one-hot code



6.4 State Machine

A state machine is a digital device that traverses through a predetermined sequence of states in an orderly fashion. A state is a set of values measured at different parts of the circuit. A simple state machine can consist of PALdevice based combinatorial logic, output registers, and buried (state) registers. The state in such a sequencer is determined by the values stored in the buried and/or output registers.

State machines are required in a variety of applications covering a broad range of performance and complexity; low-level controls of microprocessor-to- VLSI-peripheral interfaces, bus arbitration and timing generation in conventional microprocessors, custom bit-slice microprocessors, data encryption and decryption, and transmission protocols are but a few examples

A general form of a state machine can be depicted as a device shown in Figure 1. In addition to the device inputs and outputs, a state machine consists of two essential elements: combinatorial logic and memory (registers).

This is similar to the registered counter designs, which are essentially simple state machines. The memory is used to store the state of the machine.

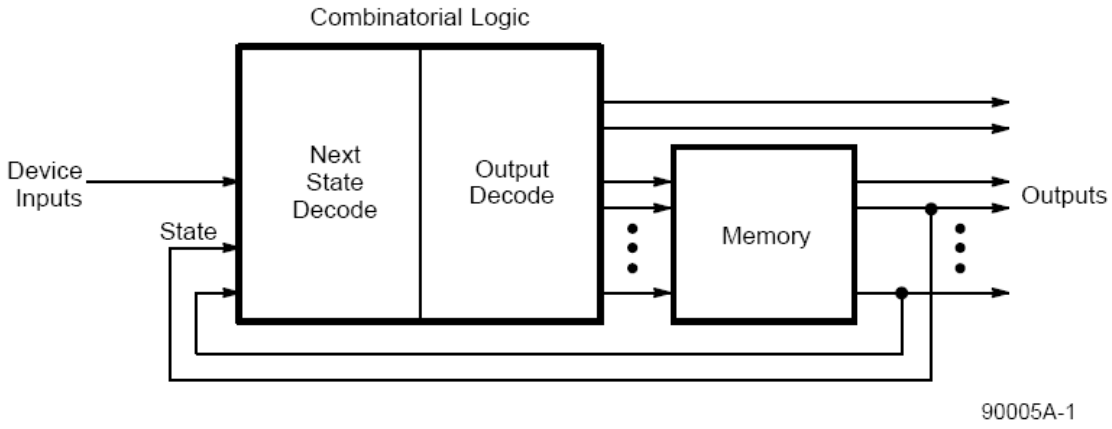


Figure 1. Block Diagram of a Simple State Machine

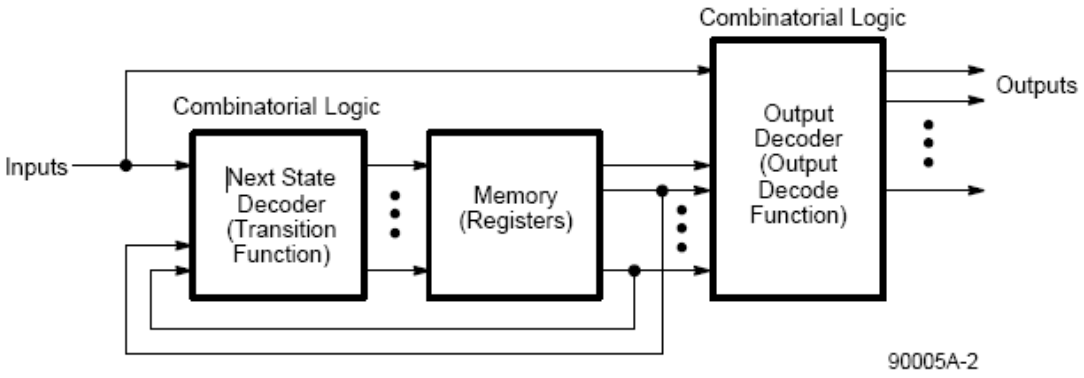


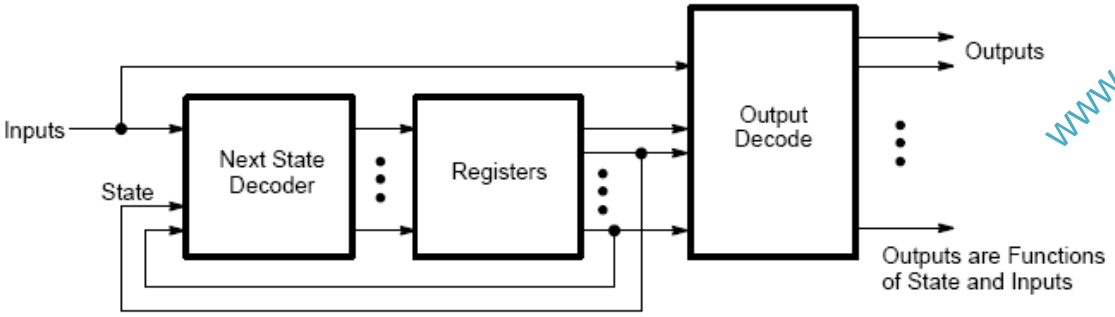
Figure 2. State Machine, with Separate Output and Next State Decoders

The basic operation of a state machine is twofold:

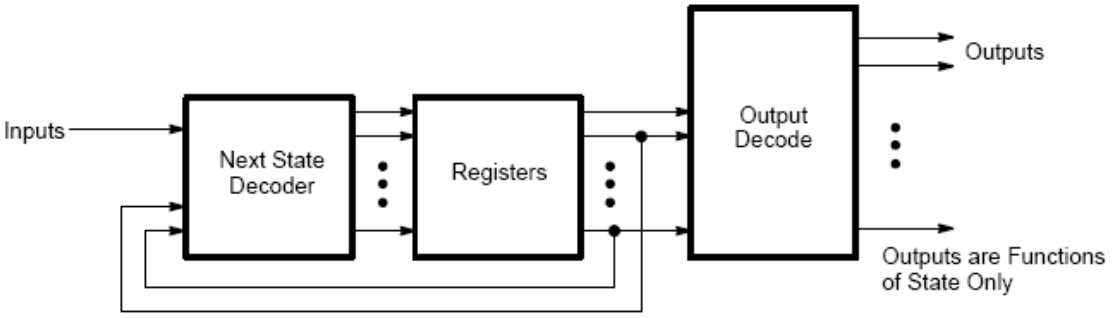
1. It traverses through a sequence of states, where the next state is determined by next state decoder, depending upon the present state and input conditions.
2. It provides sequences of output signals based upon state transitions. The outputs are generated by the output decoder based upon present state and input conditions.

The use of input signals in the decision-making process for output generation determines the type of a state machine.There are two widely known types of state machines: Mealy and Moore (Figure 2). Moore state machine outputs are a function of the present state only. In the more general Mealy-type state machines, the outputs are functions of both the state and the input signals.The logic required is known as the output function. For either type, the control sequencing depends upon both states and input signals.

Most practical state machines are synchronous sequential circuits that rely on clock signals to trigger the state transitions. A single clock is connected to all of the state and output edge-triggered flip-flops, which allows a state change to occur on the rising edge of the clock. Asynchronous state machines are also possible, which utilize the propagation delay in combinatorial logic for the memory function of the state machine. Such machines are highly susceptible to hazards, hard to design and are seldom used. In our discussion we will focus solely on sequential state machines.



a. Mealy State Machines



b. Moore State Machines

90005A-3

Figure 3. The Two Standard State Machine Models

State Machine Theory

The underlying theory for all sequential logic systems, the finite state machine (FSM), or simply state machine.

Those parts of digital systems whose outputs depend on their past inputs as well as their current ones can be modeled as finite state machines. The "history" of the machine is summed up in the value of its internal state.

When a new input is presented to the FSM, an output is generated which depends on this input and the present state of the FSM, and the machine is caused to move into new state, referred to as the next state. This new state also depends on both the input and present state.

The structure of an FSM is shown pictorially in Figure 2. The internal state is stored in a block labeled "memory."

two combinatorial functions are required: the transition function, which generates the value of the next state, and the output function, which generates the state machine output.

6.5 Review Questions

1. What is a half-adder? Write its truth table.
2. Design a half-adder using NOR gates only.
3. What is a full-adder? Draw its logic diagram with basic gates.
4. Implement a full-adder circuit using NAND gates only.
5. Implement a full-adder circuit using NOR gates only.
6. What is a decoder? Explain a 3-to-8 decoder with logic diagram.
7. Can more than one output be activated for a decoder? Justify the answer.
8. Design a 4-bit binary subtractor using a 4-bit adder and INVERTERs.
9. What is a magnitude comparator?
10. What is a multiplexer? How is it different from a decoder?
11. How are multiplexers are useful in developing combinational circuits?
12. What is the function of enable input(s) for a decoder?

6.6 Sample past Question Papers

MT KENYA UNIVERSITY
UNIVERSITY EXAMINATION
SCHOOL OF APPLIED SOCIAL SCIENCES
BIT 1202 BASIC DIGITAL ELECTRONICS AND LOGIC

INSTRUCTIONS: ANSWER QUESTION ONE AND ANY OTHER TWO
TIME: 2HOURS

QUESTION ONE

a) Explain the following number systems giving their characteristics
   i. Binary number system
   ii. Octal number system
   iii. Hexadecimal number system
   iv. Decimal dumber systems
   8marks

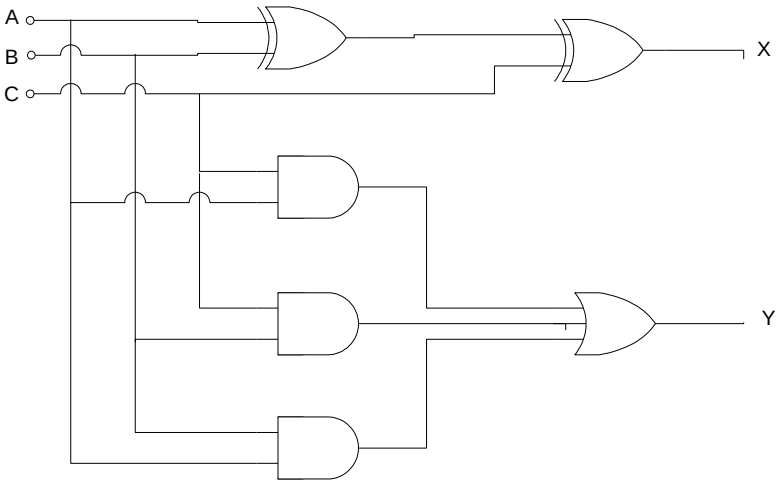
b) Discuss alphanumeric codes and outline the 8-4-2-1 BCD code
   10marks

c) Convert the following
   i. 11010111_2 to hexadecimal
   ii. 91_{10} to binary
   iii. AC9.E1 to binary
   iv. 1010111100 to otal
   v. 1110.11 to decimal 10marks

d) Using 9's complement subtract the 762-142 2marks

QUESTION TWO

a) Give a truth table for



6 marks

b) Perform the following calculations in 8-4-2-1 BCD
25+38 4marks

c) Explain sequential logic circuits 4marks

d) Differentiate between the following
   i. Asynchronous Sequential logic circuits and synchronous Sequential logic circuits
   ii. Combinational and sequential circuits 6marks

QUESTION THREE

a) Using Boolean algebra simplify the following logic expression using Boolean expression
   i. W=ABC+CAB+BAC 2marks

   ii. X=AB($\bar{A}$C+B) 3marks

   iii. Z=A$\overline{BC}$+A$\bar{B}$C+$\bar{A}$BC+$\overline{AB}$C 5marks

b) Simplify the following logic expression using k-map

F= $\overline{XYZ}$+X$\overline{YZ}$+ XYZ+ $\bar{X}$Y$\bar{Z}$ 4marks

c) When adding two binary number we you start with the least significant bit. When the two bits are added there might be a possibility of a carry. demonstrate this statement by use of half adder by giving the logic diagram and the truth table 6marks

QUESTION FOUR

a) Explain the advantage of edge triggering in flip flops
4marks

b) Give the logic symbol and truth table for JK Flip flop
6marks

c) Give important practical functions of registers
6marks

d) Explain the applications of digital electronics and logic
4marks