Part –II

DIGITAL ELECTRONICS

Chapter - 4 : Number systems and codes

Electronic circuits and systems can be broadly classified into analog and digital. Analog circuits are those in which voltages and currents show continuous variations with respect to time and can take any arbitrary value of magnitude within a specified range. A digital circuit is one in which the voltage levels assume a finite number of distinct values. In all modern digital circuits, normally there are two discrete voltage levels, called logic '0' and logic '1'.

Learning Outcomes:

At the end of this module, students will be able to:

1. Describe different types of number system.
2. Represent data in the respective number system.
3. Convert the number from one system to another system.
4. Perform binary arithmetic using ones and two's complement

Module 1- Number systems:

We discuss binary, decimal, octal and hexadecimal number systems in this module. A radix or base is an important part of any number system. The total number of symbols in every number system is equal to its base or radix. In fact the name of the number system is derived from its base.

Decimal Number system:

This system has 10 symbols, namely 0,1,2,3,4,5,6,7,8 and 9. The decimal number system is also called the base '10' system as it has '10' digits. Example: $(781)_{10}$, $(82.901)_{10}$

Binary Number system:

A number system that uses only two symbols '0'and'1' is called binary number system or base 2 system or radix-2 system. The symbols are called bits. Example: $(100010)_2$, $(0.1011)_2$.

Octal Number System:

A number system that uses 8 symbols (0-7) is called an octal number system. The radix or base of octal number system is 8. Example: $(723)_8$, $(6.76)_8$.

Hexadecimal Number System:

The hexadecimal number system has base 16. It has 16 distinct symbols. It uses the digits 0- 9 in addition to alphabets A,B,C,D, E and F as 16 symbols to represent the numbers.

Self test:

1. *The hex numbering system has a base of _____, and the binary numbering system has a base of _____.*
2. *The value of a particular digit in a number is determined by its relative position in a sequence of digits. (T/F)*
3. *A single hexadecimal digit can represent how many binary bits: (a) two, (b) three, (c) four?*
4. *The bases of the binary and decimal numbering systems are multiples of 2. (T/F)*

4.1 Conversion of numbers:

The decimal system is a more familiar system than the other systems. So, it is essential to understand the conversion of a number from any base to decimal and vice versa. The computer systems accept the data in decimal form, whereas they store and process the data in form. Therefore, it becomes necessary to convert the numbers represented in one system into the numbers represented in another system.

4.1.1: Decimal Number System

Decimal to Binary: The given decimal number is repeatedly divided by 2, which is the base number of binary systems till quotient becomes '0' and the remainder is collected from bottom wto top. To convert the fractional part into binary, fraction part is multiplied by 2 repeatedly and any carry in integer place is recorded. The string of integer obtained from top to bottom gives the equivalent fraction in binary number system.

Ex1: $(37)_{10} = (100101)_2$

37	divided by	2	Q=18	R=1
18	divided by	2	Q=9	R=0
9	divided by	2	Q=4	R=1
4	divided by	2	Q=2	R=0
2	divided by	2	Q=1	R=0

Note the way the bits are read to form the binary number.

Decimal to Octal: The given decimal number is repeatedly divided by 8, which is the base number of octal system till quotient becomes '0' and the remainder is collected from bottom to top. To convert the fractional part into octal, fraction part is multiplied by 8 repeatedly and any carry in integer place is recorded. The string of integer obtained from top to bottom gives the equivalent fraction in the octal number system.

Example - 1: $(97)_{10} = (141)_8$

97	divided by	8	Q=12	R=1
12	divided by	8	Q=1	R=4

Decimal to Hexadecimal: The given decimal number is repeatedly divided by 16, which is the base number of hexadecimal system till quotient becomes '0' and the remainder is collected from bottom to top. To convert the fractional part into hexadecimal, fraction part is multiplied by 16 repeatedly and any carry in integer place is recorded. The string of integer obtained from top to bottom gives the equivalent fraction in the hexadecimal number system.

Ex1: $(546)_{10} = (222)_{16}$

546	divided by	16	Q=34	R=2
34	divided by	16	Q=2	R=2

4.1.2: Binary Number System

Binary to decimal: Multiply the number by its equivalent binary weights. Add the products to get the decimal number.

Ex1: $110_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 + 0 = 6$

Ex2: $0.101_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 0.5 + 0 + 0.125 = 0.625$

Ex3: $110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 4 + 2 + 0 + 0.5 + 0 + 0.125 = 6.625$

Binary to Octal: Each octal digit is represented by three bits of binary.

Ex1: $(110101010)_2 = (652)_8$ 110 101 010 6 5 2

Ex2: $(0.1011010)_2 = (550)_8$ 101 101 0 5 5 0

Ex3: $(110101010.\ 1011010)_2 = (652.550)_8$

Binary to Hexadecimal: Each group of 4 binary bits is equal to 1 hexadecimal digit.

Ex1: $(111110110010)_2 = (FB2)_{16}$ 1111 1011 0010 15 11 2 ; F B 2

Ex2: $(0.010101111011)_2 = (0.57B)_{16}$ 0101 0111 1011; 5 7 11 ; 5 7 B

4.1.3: Octal Number system:

Octal to Binary: Each octal digit is represented by three binary bits

Ex1: $(347)_8 = (011100111)_2$

Ex2: $(0.245)_8 = (0.010100101)_2$

Ex3: $347.245_8 = (011100111.010100101)_2$

Octal to decimal: Multiply the number by its equivalent octal weights. Add the products to get the decimal number.

Ex1: $(457)_8 = 4 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 = 4 \times 64 + 5 \times 8 + 7 \times 1 = 256+40+7 = (303)_{10}$

Ex2: $(0.246)_8 = 2 \times 8^{-1} + 4 \times 8^{-2} + 6 \times 8^{-3} = 2 \times 0.125 + 4 \times 0.015625 + 6 \times 0.001953125$

$= (0.267969)_{10}$

Ex3: $(457.246)_8 = (303.267969)_{10}$

Octal to Hexadecimal: To convert an octal number to hexadecimal number, the octal number is first converted into binary. The binary bits are grouped such that each group consists of 4 bits and a group starts from the LSB. The 4 bits group is represented by its equivalent hexadecimal number.

Ex1: $(235)_8 = (09D)_{16}$

$(235)_8 = (\underline{0}\ \underline{1001}\ \underline{1101})_2 = (09D)_{16}$

4.1.4 Hexadecimal Number Systems:

Hexadecimal to Binary: Each group of 4 binary bits is equal to one hexadecimal digit.

Ex1: $(A7D)_{16} = (\underline{1010}\ \underline{0111}\ \underline{1110})_2$

Hexadecimal to Octal: To convert an hexadecimal number to octal number, the hexadecimal number is first converted into binary. They are grouped into 3 bits each starting from the LSB. The 3 bits group is represented by its equivalent octal number.

Ex1: $(C4)_{16} = (11\ 000\ 100)_2 = (304)_8$

Ex2: $(0.26A)_{16} = (0.001\ 001\ 101\ 010)_2 = (0.1152)_8$

Ex3: $(C4.26A)_{16} = (11\ 000\ 100.001\ 001\ 101\ 010)_2 = (304.1152)_8$

Hexadecimal to Decimal: Multiply the number by its equivalent hexadecimal weights. Add the products to get the decimal number.

Ex1: $(B40)_{16} = 11 \times 16^2 + 4 \times 16^1 + 0 \times 16^0 = 11 \times 256 + 4 \times 16 + 0 \times 1 = (2880)_{10}$

Ex2: $(0.237)_{16} = 2 \times 16^{-1} + 3 \times 16^{-2} + 7 \times 16^{-3} = (0.138427)_{10}$

Ex3: $(B40.237)_{16} = (2880.138427)_{10}$

Self test:

1. *The binary equivalent of decimal number 255 is _____.*
2. *The binary equivalent of hexadecimal number 1C is _____.*
3. *The decimal equivalent of hexadecimal number 1B6 is _____.*
4. *The hexadecimal equivalent of decimal number 129 is _____.*
5. *The decimal equivalent of binary number 110101 is _____.*
6. *The hexadecimal equivalent of binary number 1001 is _____.*
7. *The binary equivalent of decimal number 28 is _____.*
8. *The binary equivalent of hexadecimal number 35 is _____.*
9. *The decimal equivalent of hexadecimal number 7 is _____.*
10. *The hexadecimal equivalent of decimal number 49 is _____.*
11. *The decimal equivalent of binary number 110110110 is _____.*
12. *The hexadecimal equivalent of binary number 1110 is _____.*

4.1.5 Number System - Arithmetic:

Addition in Binary system: It is a simple task to add two binary numbers and it is very similar to addition of decimal numbers.

Ex1: 1010 + 0111 = 10001

1010 = 10

+ 0111 = 7

10001 = 17

Augend	Addend	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Addition in Octal system: Add the digit in each column in decimal and convert this sum into octal. Write the sum in that column and carry the carry term to the next higher significant column

Ex1: Add $(7AB.67)_{16}$ to $(15C.71)_{16}$ = $(907.D8)_{16}$

Augend	Addend	Sum	Carry
7	1	8	0(since sum<16)
6	7	6+7 = 13 (D)	0(since sum <16)
B(11)	C(12)	11+12=23 – 16= 7	1(16 subtracted once)
A(10)	5	10+5+1 = 16 – 16=0	1(16 subtracted once)
7	1	7+1+1 = 9	0(since sum <16)

Self test:

1. *The result of $101_2 + 11_2$ is _____ (in binary).*
2. *The result of $A1_{16} + BC_{16} + 10_{16}$ is _____ (in hexadecimal).*
3. *The result of $60_{10} + F1_{16} - 1001001_2$ is _____ (in decimal).*
4. *The result of $11_2 + 27_8 + 93_{10} - B_{16}$ is _____ (in decimal).*

4.1.6 Complementation of numbers:

The subtraction operation and logical manipulations become easy in digital computers by using the concept of complements. For a given number 'N' in base-'r', two types of complements are defined, namely, r's complement and (r-1)'s complement. The main advantage of performing subtraction by complement is that it requires less hardware to implement.

Subtraction using complements in Binary system:

The following rules are used in performing complement subtraction in binary system.

1. Identify the minuend and the subtrahend.

2. Find the 2's complement or the 1's complement of the subtrahend.

3. Add the complemented subtrahend to the minuend.

4. Check if there is a carry. If there is no carry in either of the form (1's or 2's complement) then find the corresponding complement of the result.

5. In case of 2's complement if there is a carry then neglect it. In case of 1's complement if there is a carry then add it to the LSB of the result.

Note: To find 1's complement of a binary number we need to invert each bit.

To find 2's complement of a binary number find 1's complement and add 1 to the result.

Ex1: Subtract $(0011)_2$ from $(0101)_2$ using 1's complement.

Step1: 0101 is minuend and 0011 is subtrahend.

Step2: 1's complement of 0011 is 1100.

Step3: 0101 + 1100 = 1 0001(there is a carry)

Step4: 0001 + 1 = 0010 (5 – 3 =2)

Exercises:

1. Subtract $(AEF3.6D)_{16}$ from $(445.63)_8$ using 1's complement method.
2. Perform the following
   (i) $(257.75)_{10}$ - $(128.825)_{10}$ using binary 2's complement arithmetic
   (ii) $(ABCD)_{16} = (\ ?\ )_{10} = (\ ?\ )_2 = (\ ?\ )_8$
3. Perform the following addition in the binary system.
   1101 + 1001
   1101.01 + 101.11
   11.011 + 10.111
   1001 + 111
4. Determine the base b of the number system such that $225_{(b)} = 89_{(10)}$
5. Convert each of the following hexadecimal number into its equivalent in the binary number system
   i) 1C.3
   ii) F2.C
   iii) 450.B
   iv) 8EA.59
6. Perform the following conversions
   i) $(7305)_{10}$ to hexadecimal
   ii) $(7305)_8$ to decimal
7. Perform the following:
   i) $(F69.D3)_{16} - (A25.68)_{16} = (?)_{10}$
   ii) $(13.25)_{10} - (26.37)_{10}$ using Binary 1's complement arithmetic

Summary

1. A numeral system (or system of numeration) is a writing system for expressing numbers, that is, a mathematical notation for representing numbers of a given set, using digits or other symbols in a consistent manner.
2. The computer systems accept the data in decimal form, whereas they store and process the data in binary form. Therefore, it becomes necessary to convert the numbers represented in one system into the numbers represented in another system.
3. The advantage of performing subtraction by complement is reduction in the hardware. Instead of having separate digital circuits for addition and subtraction, only adding circuits are needed.

Module 2 – Codes

Learning Outcomes:

At the end of this module, students will be able to:

1. Discuss different types of binary codes.
2. Explain error detection using parity bit.
3. Describe error correction using hamming code.

When numbers, letters or words are represented as a specific group of symbols based on certain rules, it is said to be encoded. The group of symbols is called a code. Codes are represented, stored and transmitted in the form of binary bits. The codes may also use alpha numeric characters.

Advantages of Binary Code:

- Suitable for computer applications.
- Used in digital communications.
- Ease of circuit implementation.

Classification of binary codes

Generally codes are classified into following categories.

- Weighted Codes
- Non-Weighted Codes
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes

4.2.1 Weighted binary codes:

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the bit or digit in the code has a specific weight. Decimal digits 0-9 can be represented using 4 bit binary code. One such example is 8421 BCD (Binary coded Decimal), where each group of 4 bits is a weighted code as shown in Figure 4.2.1.

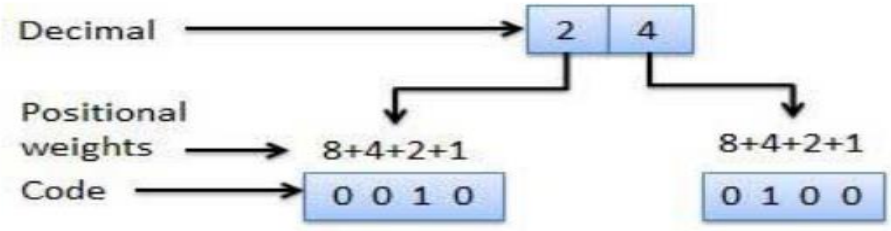


Figure 4.2.1: 8421 BCD code Illustration

Value of the number given by the code can be obtained using the following equation.

$$N = d_3w_3 + d_2w_2 + d_1w_1 + d_0w_0 \qquad (4.2.1)$$

w_3, w_2, w_1 and w_0 are the weights selected for a given BCD code:

w_3	w_2	w_1	w_0
8	4	2	1

Figure 4.2.2 weights of a BCD code.

For example the decimal number 9 would be represented by 1001 where 9 = 1x8 + 0x 4 +0x2 + 1x1. The most used binary coded decimal (BCD) is shown in Table 4.2.

Table 4.2.1 Decimal to Other BCD forms

Decimal	8421	7421	4221	$84\bar{2}\bar{1}$
0	0000	0000	0000	0000
1	0001	0001	0001	0111
2	0010	0010	0010	0110
3	0011	0011	0011	0101
4	0100	0100	1000	0100
5	0101	0101	0111	1011
6	0110	0110	1100	1010
7	0111	1000	1101	1001
8	1000	1001	1110	1000
9	1001	1010	1111	1111

4.2.2 Non-Weighted Codes:

Non-weighted code is one in which the positions in the code do not have a specific weight.

EXCESS-3 CODE

The Excess-3 code is also called XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words by adding (0011)2 or (3)10 to each code word in 8421.

GRAY CODE

The Gray code is neither a decimal code, nor is it an arithmetic code. The essential feature of a Gray code is that there is only a single bit difference between successive code words.

Table 4.2.2 BCD, Excess-3 and Gray code equivalent for decimal numbers

Decimal	BCD = 8421	Excess-3	Gray
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0011
3	0011	0110	0010
4	0100	0111	0110
5	0101	1000	0111
6	0110	1001	0101
7	0111	1010	0100
8	1000	1011	1100
9	1001	1100	1101

4.2.3 Self complementing code:

When arithmetic operations are performed, it is often required to take the “complement” of a given number. If the logical complement of a coded number is also its arithmetic complement, it will be convenient from hardware point of view. Self-complementing codes are applicable in such scenario. A code is said to be self-complementing, if the code word of the 9’s complement of N can be obtained from the code word of N by converting all the 0’s into 1’s and all 1’s into 0’s.

Example: Consider the 2421 code.

The 2421 code of $(4)_{10}$ is 0100.

Its complement is 1011 which is 2421 code for $(5)_{10} = (9 - 4)_{10}$.

4.2.4 Error detection and correction codes:

When data is transmitted in digital form from one place to another place through a transmission channel some data bits may be lost or modified. As there are needs to transmit

millions of bits per second, the data integrity should be very high. The error rate cannot be reduced to zero. A simple process of adding a special code bit to a data word can improve its integrity. This extra bit will allow detection of a single error in a given code word in which it is used, and is called the 'Parity Bit'.

a) Parity (Error detection code)

The simplest technique for error detection is that of adding an extra bit known as parity bit, to each word being transmitted. There are two types of parity – odd parity and even parity.

i. Odd parity: the parity bit is set to 0 or 1 at the transmitter such that the total number of 1 bit in the word including the parity bit is an odd number.

ii. Even parity: the parity bit is set to 0 or 1 at the transmitter such that the total number of 1 bit in the word including the parity bit is an even number.

Table 4.2.3 BCD with Odd and Even parity

Decimal	8 4 2 1 BCD	Odd Parity	Even Parity
0	0 0 0 0	0000**1**	0000**0**
1	0 0 0 1	0	1
2	0 0 1 0	0	1
3	0 0 1 1	1	0
4	0 1 0 0	0	1
5	0 1 0 1	1	0
6	0 1 1 0	1	0
7	0 1 1 1	0	1
8	1 0 0 0	0	1
9	1 0 0 1	1	0

Self test:

1. *What is the simplest technique for detecting error in the codes?*
2. *What is a parity bit?*
3. *What is a self -complementing code? Give an example.*

1. Example: In an even parity scheme, which of the following words contain an error?
   a. 10101010 Ans: No
   b. 1110110 Ans: Error
   c. 10111001 Ans: Error
2. Example: In an odd parity scheme, which of the following words contain an error?
   a) 10110111 Ans: Error
   b) 10011010 Ans: Error
   c) 11101010 Ans: No

b) Hamming Codes (Error correction code)

A code is said to be an error-correcting code, if there is a built in capability to correct errors occurred in the code word by some means, and thereby the correct code word can always be recovered from an erroneous word. For a code to be a single – bit error correcting code, the minimum distance of that code must be three. The minimum distance of a code is the smallest number of bits by which any two code words must differ. A code with minimum distance of three cannot only correct single bit errors, but also detect two bit errors. One type of error correcting code is hamming code.

In this code, to each group of m information, k parity checking bits denoted by P1.to Pk located at position 2(k-1) from left are added to form an (m+k) bit code word. To correct the error, k parity checks are performed on selected bits of each code word, and the position of the error bit is located by forming an error word and the error bit is then complemented. The k bit error word is generated by putting a 0 or a 1 in the $2(k-1)^{th}$ position depending upon whether the check for parity involving the parity bit Pk is satisfied or not.

7 Bit Hamming code

To transmit 4 data bits, 3 parity bits located at positions 2^0 2^1 and 2^2 from left are added to make a 7 bit code word which is then transmitted. The word format

P1 P2 D3 P4 D5 D6 D7

Where the D bits are data bits and P bits are parity bits.
P1 is to be set to 0 or 1 so that it establishes even parity over bits 1,3,5 and 7.
P2 is to be set to 0 or 1 so that it establishes even parity over bits 2,3,6 and 7.
P4 is to be set to 0 or 1 so that it establishes even parity over bits 4,5,6 and 7.

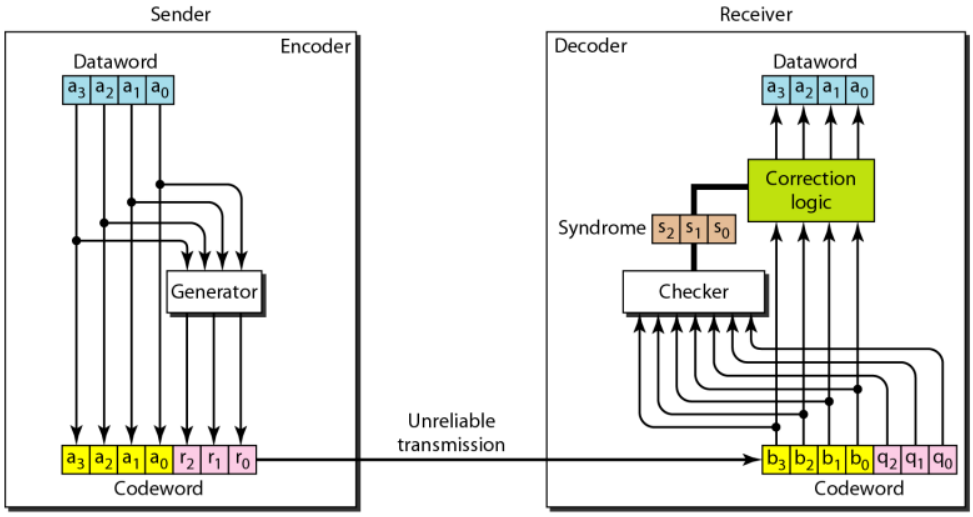


Figure 4.2.3 Coding and Decoding of Hamming

7 bit hamming code for the decimal digit coded in BCD and EXCESS 3 codes is shown inTable 4.2.4. The minimum distance of the 7 bit hamming code for the BCD code is 3 as observed Table 4.2.4.

Table 4.2.4 Hamming code for BCD and Excess-3

Decimal Digit	Hamming Code bits	
	For BCD	For EXCESS 3
	P1 P2 D3 P4 D5 D6 D7	P1 P2 D3 P4 D5 D6 D7
0	0 0 0 0 0 0 0	1 0 0 0 0 1 1
1	1 1 0 1 0 0 1	1 0 0 1 1 0 0
2	0 1 0 1 0 1 1	0 1 0 0 1 0 1
3	1 0 0 0 0 1 1	1 1 0 0 1 1 0
4	1 0 0 1 1 0 0	0 0 0 1 1 1 1
5	0 1 0 0 1 0 1	1 1 1 0 0 0 0
6	1 1 0 0 1 1 0	0 0 1 1 0 0 1
7	0 0 0 1 1 1 1	1 0 1 1 0 1 0
8	1 1 1 0 0 0 0	0 1 1 0 0 1 1
9	0 0 1 1 0 0 1	0 1 1 1 1 0 0

At the receiving end, the message received in the hamming code is decoded to see if any errors have occurred. Bits 1,3,5, 7 and bits 2,3,6,7 and 4,5,6,7 are all checked for even parity. If they all check out, there is no error. If there is an error, the error bit can be located by forming a 3 bit binary number C3 C2 C1

$$C3 = D4 \oplus D5 \oplus D6 \oplus D7$$

$$C2 = P2 \oplus D3 \oplus D6 \oplus D7$$

$$C1 = P1 \oplus D3 \oplus D5 \oplus D7$$

After 3 parity checks , the error bit is corrected by complementing it.

Example: Encode data bits 1101 into 7 bit even parity hamming code.

P1	P2	D-3	P4	D5	D6	D7
		1		1	0	1

Bits 1,3,5,7 (i.e. P1 1 1 1) must be even parity. So P1 must be 1.

Bits 2,3,6,7 (i.e. P2 1 0 1) must be even parity. So P2 must be 0.

Bits 4,5,6,7 (i.e. P4 1 0 1) must be even parity. So P4 must be 0.

Therefore , the final code is 1 0 1 0 1 0 1

Exercises:

1. Consider the sequence of digits 10001001010110000011. Determine the number being represented in each of the following BCD coding schemes:
   a. 8421 code
   b. Excess-3 code
   c. 2 4 2 1 code
2. The message coded in the 7 bit hamming code 1001001 is transmitted through a noisy channel. Decode the message assuming that at most a single error occurred.

Chapter -5: Boolean Algebra and Logic Gates

Module-1: Introduction to Boolean Algebra

Learning Outcomes:

At the end of this module, students will be able to:

1. State the laws of Boolean algebra.
2. Simplify Boolean expressions using Boolean algebraic theorems.

5.1.1 Boolean Algebraic Theorems

George Boole in 1854 invented a new kind of algebra known as Boolean algebra. It is sometimes called switching algebra. Boolean algebra is the mathematical frame work on which logic design is based. It is used in synthesis and analysis of binary logical function.

History

The algebraic system known as Boolean algebra named after the mathematician George Boole. George Boole Invented multi-valued discrete algebra (1854) and E. V. Huntington developed its postulates and theorems (1904). Historically, the theory of switching networks (or systems) is credited to Claude Shannon, who applied mathematical logic to describe relay circuits (1938). Relays are controlled electromechanical switches and they have been replaced by electronic controlled switches called logic gates. A special case of Boolean Algebra known as Switching Algebra is a useful mathematical model for describing the combinational circuits.

a. Boolean Algebraic theorems:

These have been derived by using Boolean postulates. These laws are used to design and analyze logic circuit mathematically. Table 5.1.1 summarizes the Boolean theorems.

Table 5.1.1: Boolean Theorems

Laws of union
Law1: A + 0 = A Law2: A + 1 = 1
Laws of intersection
Law3: A . 1 = A Law4: A . 0 = 0
Laws of tautology
Law5: A + A = A Law6: A.A = A
Laws of complements
Law7: A + A' = 1 Law8: A.A' = 0
Law of double complement
Law9: A'' = A
Laws of commutation
Law10: A + B = B + A Law11: A.B = B.A
Laws of association
Law12: A + (B + C) = (A + B) + C Law 13: A. (B . C) = (A . B) . C
Laws of distribution
Law14: A (B + C) = AB + AC Law15: (A+B) (C+D) = AC + AD + BC + BD
Laws of absorption
Law16: A (A+B) = A Law17: A + AB =A Law18: A (A'+B) = AB Law19: AB + B' = A+B' Law20: AB' + B = A+B

Self test:

1. *Prove laws of absorption using appropriate Boolean laws*

b) Principle of Duality: One more important property of Boolean algebra is called Duality principle. The Dual of any expression can be obtained easily by the following rules.

1. Change all 0's to 1's

2. Change all 1's to 0's

3. AND's (dot's) are replaced by OR's (plus)

4. OR's (plus) are replaced by AND's (dot's)

Example 1:

No.	Boolean Expression	Dual of the expression
1	X + 0 = X	X . 1 = X
2	X + Y = Y + X	X . Y = Y . X
3	X + 1 = 0	X . 0 = 1

c) De Morgon's Theorems: It is one of the important properties of Boolean algebra. It is extensively useful in simplifying complex Boolean expression.

De Morgon's First Theorem: It states that "the complement of product of variables is equal to sum of the complements of individual variable".

i.e. $\overline{AB} = \overline{A} + \overline{B}$. The proof is given below.

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

De Morgon's Second Theorem: It states that "the complement of sum of variables is equal to product of complement of two individual variables".

i.e. $\overline{A+B} = \overline{A}.\overline{B}$. The proof is shown using a truth table given below.

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

5.1.2 Simplification of Boolean algebraic expressions

It will be always desirable to simplify any Boolean Expression to its simplest form before converting to a logic circuit so that the circuit is constructed with minimum number of logic gates. It makes cost effective and more reliable due to lesser number of interconnections.

Example 2: Simplify

i. F = X'Y'Z + X'YZ ii) F = X (X' + Y) iii) F = B (A+C) + C

Solution:

i. F = X'Y'Z + X'YZ
= X'Z (Y' +Y) (Y'+Y=1)
= X'Z

ii. F = X (X' + Y)
= X.X' + XY (X.X' = 0)
= XY

iii. F = B (A +C) +C

= BA + BC +C

= BA + C (1+B) (1+B = 1)

= BA + C

Example 3:

i. Prove that $Y (WZ' + WZ) + XY = Y(W+X)$

ii. Prove that $ABC + A'BC + AB'C + ABC' = AB + BC + CA$

i) L.H.S. $= Y (WZ' + WZ) + XY$

$= Y (WZ' + WZ) + XY$

$= YW (Z+Z') + XY$ $(Z+Z'=1)$

$= YW + XY$

$= Y (X+W)$

ii) L.H.S $= ABC + A'BC + AB'C + ABC'$

$= BC (A+A') + AB'C + ABC'$

$= BC + AB'C + ABC'$

$= C (B + AB') + ABC'$ $(B+B'A = B+A)$

$= C (B+A) + ABC'$

$= BC + AC + ABC'$

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

$= BC + A (C+B)$

$= BC + AC + AB$

Exercise:

1. Simplify the following
   - i) $F = XY+XYZ+XYZ'+X'YZ$ (Ans : $XY+YZ$)
   - ii) $F = XYZ+X'Y+XYZ'$ (Ans: Y)
2. Prove that $(A+B) (A+C) = A+BC$

Summary

In this module we have learnt to:

1. List the Boolean algebraic theorems
2. Simplify the given Boolean expressions by applying Boolean Algebra theorems
3. The need for simplification of Boolean Expression.
4. The Dual of any expression can be obtained easily by the following rules - 1. Change all 0's to 1's, 2. Change all 1's to 0's, 3. AND's (dot's) are replaced by OR's (plus) and 4. OR's (plus) are replaced by AND's (dot's).
5. De Morgon's First Theorem: It states that "the complement of product of variables is equal to sum of the complements of individual variable".
6. De Morgon's Second Theorem: It states that "the complement of sum of variables is equal to product of complement of two individual variables".
7. Simplification of Boolean algebraic expressions makes cost effective and more reliable logic circuits due to lesser number of interconnections.

Module 2: Logic Gates

Logic gate is a physical device or hardware used to implement a Boolean function. It performs the Boolean operation as per the design. Logic gates are primarily implemented using diodes or transistors operating as electronic switches. In this module, we will discuss basic logic functions, corresponding Gates and implementation of combinational circuits using logic gates.

Learning Outcomes:

At the end of this module, students will be able to:

1. Describe basic logic gates and the concept of universal logic.
2. Build a logic circuit for the given Boolean expressions.
3. Write Boolean expressions for the given logic circuit.
4. Differentiate combinational and sequential circuits.

5.2.1 Logic gates and operation

Logic gate is an electronic circuit, which makes logic decisions. It is a digital circuit with one or more input signal and only one output signal. It produces one output level when some combinations of input levels are present and a different output level when other combinations of input levels are present. All input or output signals are either low voltage or high voltage. A digital circuit is referred to as logic gate for simple reason that, it can be analyzed based on Boolean algebra. To make logical decisions, three basic gates are used. They are OR, AND and NOT gate. These logic gates are building blocks, which are available in the form of an Integrated circuit (IC). The input and output of the binary variables for each gate can be represented in a tabular column called as truth table.

a) Basic gates

i) OR Gate: The OR gate performs logical additions commonly known as OR function. The OR gate has two or more inputs and only one output. The operation of OR gate is such that a HIGH (logic 1) on the output is produced when any of the input or both the inputs are HIGH. The output is LOW(logic 0) only when all the inputs are LOW. If A and B are the input variables of an OR gate and Y is its output, then Y=A+B. Similarly for more than two variables, the OR function can be expressed as Y=A+B+C.

Figure 5.2.1: Logical Symbol: Two Input OR gate

Table 5.2.1: Truth table for two input OR gate:

Input		Output
A	B	Y= A+B
0	0	0
0	1	1
1	0	1
1	1	1

Realization of OR gate using diodes: Two input OR gate using "diode-resistor" logic is shown in Figure 5.2.2, where X, Y are the inputs and F is the output.

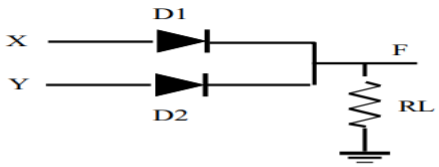


Figure 5.2.2: OR gate using Diode Resistor Logic

- If X = 0 and Y = 0, then both the diodes D1 and D2 are reverse biased and thus both the diodes are in cut-off and thus F is LOW.
- If X = 0 and Y = 1, D1 is reverse biased, thus does not conduct. But D2 is forward biased, thus conducts and thus pulling F to HIGH
- If X = 1 and Y = 0, D2 is reverse biased, thus does not conduct. But D1 is forward biased, thus conducts and thus pulling F to HIGH.
- If X = 1 and Y = 1, then both the diodes D1 and D2 are forward biased and thus both the diodes conduct and thus F is HIGH

ii) AND Gate: The AND gate performs logical multiplication, and commonly known as AND function. The AND gate has two or more inputs and a single output. The output of an AND gate is HIGH only when all the inputs are HIGH. Even if any one of the input is LOW, the

output will be LOW. If A and B are input variables of an AND gate and Y is its output, then Y=A.B.

Figure 5.2.3. Logical Symbol: Two input AND gate

Table 5.2.2: Truth table for two input AND gate

Input		Output
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Realization of AND gate using diodes: Two input AND gate using "diode-resistor" logic is shown in Figure 5.2.4, where X, Y are inputs and F is the output.

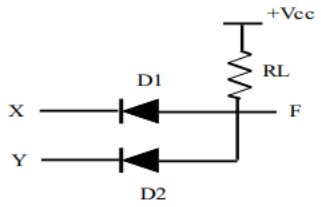


Figure 5.2.4 AND gate using Diode Resistor Logic

- If X = 0 and Y = 0, then both the diodes D1 and D2 are forward biased and thus both the diodes conduct and pulls F to LOW.
- If X = 0 and Y = 1, D1 is reverse biased, thus does not conduct. But D2 is forward biased, thus conducts and pulls F to LOW.
- If X = 1 and Y = 0, D2 is reverse biased, thus does not conduct. But D1 is forward biased, thus conducts and pulls F to LOW.

- If X = 1 and Y = 1, then both the diodes D1 and D2 are reverse biased and thus both the diodes are in cut-off and there is no drop in voltage at F. Thus F is HIGH.

iii) NOT Gate (Inverter): The NOT gate performs the basic logical function called inversion or complementation. The purpose of his gate is to convert one logic level into the opposite logic level. It has one input and one output. When a HIGH level is applied to an inverter, a LOW level appears at the output and vice-versa.

Figure 5.2.5: Logic symbol of NOT gate

Table 5.2.3: Truth table for NOT gate

Input	Output
A	$Y=\overline{A}$
0	1
1	0

Realization of NOT gate using Transistors: A NOT gate using a transistor is shown in figure 5.2.6. 'A' represents the input and 'F' represents the output.

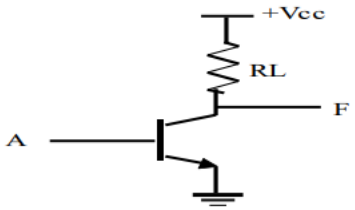


Figure 5.2.6: Realization of NOT gate using transistor.

- If A = 0, then the transistor is OFF thus pulling F to HIGH.
- If A = 1, then the transistor is ON thus driving F to HIGH.

b) Other type of gates

i) NAND Gate: The output of a NAND gate is LOW only when all inputs are HIGH and output of the NAND is HIGH if one or both inputs are LOW.

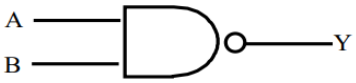


Figure 5.2.7: Logical Symbol: Two input NAND gate

Table 5.2.4 : Truth table of NAND gate

Input		Output
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

ii) NOR Gate: The output of the NOR gate is HIGH only when all the inputs are LOW.

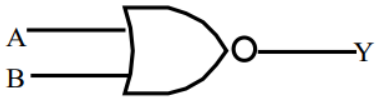


Figure 5.2.8: Logical Symbol: Two input NOR Gate

Table 5.2.5: Truth table for NOR gate

Input		Output
A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

iii) XOR Gate or Exclusive OR gate: In this gate output is HIGH only when any one of the input is HIGH. The circuit is also called as inequality comparator, because it produces output when two inputs are different.

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

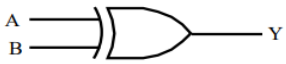


Figure 5.2.9: Logical Symbol: Two input XOR Gate

Table 5.2.6: Truth table for an XOR gate

Input		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

iv) XNOR Gate or Exclusive NOR Gate: XNOR gate is a gate with two or more inputs and one output. XNOR operation is complementary of XOR operation. i.e. The output of XNOR gate is High, when all the inputs are identical; otherwise it is low.

Figure 5.2.10: Logical Symbol of Two input XNOR Gat

Table 5.2.7 Truth table for XNOR gate

Input		Output
A	B	$Y = \overline{A}\ \overline{B} + AB$
0	0	1
0	1	0
1	0	0
1	1	1

Self test:

How many two input AND and OR gates are required to realize the following expressions in addition to inverters?

a) *F1=ABC+AB'CD+EF'+AD*
b) *F2=A(B+C+D')(B'+C+E')(A+B'+C+E)*
c) *What is the uniqueness of XOR logic? Explain briefly.*
d) *List two typical applications of XOR logic.*

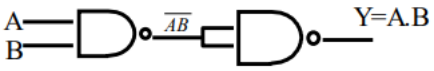
5.2.2. Concept of Universal Logic

NAND and NOR gates are called Universal gates or Universal building blocks, because both can be used to implement any gate like AND, OR and NOT gates or any combination of these basic gates.

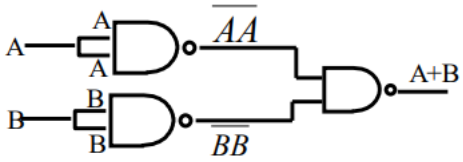
a) NAND gate as Universal gate:

1. NOT operation:

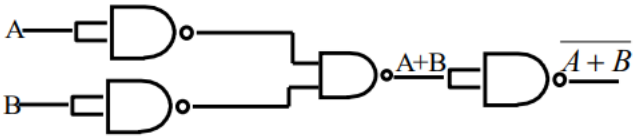
2. AND operation:



3. OR operation:



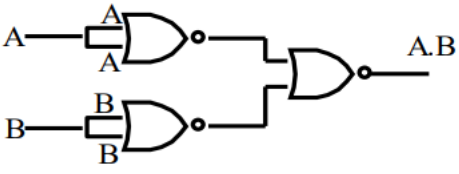
4. NOR operation:



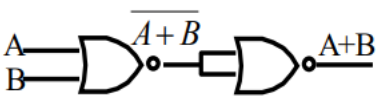
b)NOR gate as Universal gate:

1. NOT operation:

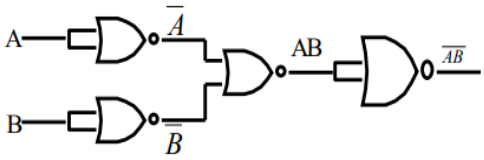
2. AND operation:



3. OR operation:



4. NAND operation:



Exercise:

1. Draw the logic circuit for the Boolean expression. Y= BC+A' C+AB'C.
2. Show that AB+(A+B) is equivalent to A Θ B. Also construct the corresponding logic diagrams.
3. The most suitable gate to check whether the number of 1's in a digital word is even or odd is

   a) X-OR b) NAND c) NOR d) AND, OR and NOT
4. Realize NOR and NAND gate using discrete components.
5. Realize NOR and NAND gate using Basic gates.

5.2.3 Classification of digital circuits

Digital circuits are circuits constructed using digital gates and operate as per digital logic. Basically digital circuits can be classified into two types.

- Combinational Circuits
- Sequential Circuits

a) Combinational Digital Circuits:

Logic circuits whose output at any instant of time is entirely dependent upon the input signals present at that time are known as combinational digital circuits as shown in Figure 5.2.11. In particular, the output of the combinational circuit doesn't depend upon any past input or output. The circuit doesn't possess any memory. The output signals of combinational circuits are not fed back to any other part of the circuit.

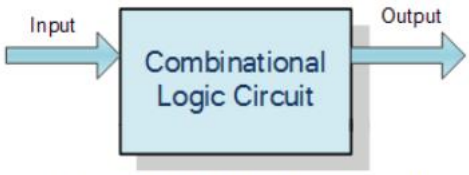


Figure 5.2.11 Combinational Circuit

Combinational circuits are generally faster, since the operation need not have to be performed in sequences. Combinational circuits can be constructed using Sum of Products (SOP) or Product of Sums(POS).

1. Sum of products (SOP): Sum of product is the logical expression in which OR of multiple product terms are present. Each product term is the logical AND of literals. Y+XY'+XYZ is an example of SOP.
2. Product of Sums (POS): Product of sums is the logical expression in which AND of multiple sum terms are present. Each sum term is the logical OR of literals. (X+Y')((X'Y+Z)(X+Y+Z) is an example of POS.
3. Minterm: It is a special type of product (AND) term. It is a product term which contains all the input variables that make up a Boolean expression.
4. Maxterm: It is a special type of sum (OR) term. A maxterm is a sum term that contains all the input variables that make up a Boolean expression.
5. Canonical form: Canonical is defined as "conforming to a general rule". The rule for Boolean logic is that each term used in a boolean expression must contain all the variables.
6. Canonical Sum of Products: A canonical SOP is a complete set of minterms that defines when an output variable is a logical '1'. Each minterm corresponds to the row in the truth table when the output function is 1.
7. Canonical Product of Sums: A canonical POS is a complete set of maxterms that defines when an output variable is a logical '0'. Each maxterm corresponds to the row in the truth table when the output function is 0.

Boolean algebra is used to simplify the Boolean expressions, thus reducing redundancy and designing low cost logic circuits. Any logic expression can be implemented by logic gates.

b) Examples of combinational circuits:

i) Half Adder: An electronic combinational circuit which performs the arithmetic addition of two binary bits is called Half Adder. In the half adder circuit, there are two inputs, addend, and augend and two outputs are Sum and Carry. The logic symbol of half adder is shown in Figure 5.2.13 and the truth table of half adder is given in table 5.2.8.

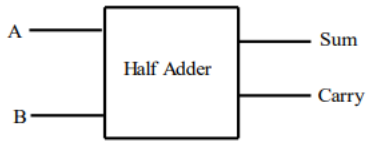


Figure 5.2.13: Half Adder Logic Symbol.

Table 5.2.8: Truth table of half adder.

Input		Output	
Augend	Addend	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The expression for sum and carry is given in expressions 5.2.1 and 5.2.2 respectively.

Sum= A B+A B =A ⊕ B …….(5.2.1)

Carry= A.B …….. (5.2.2)

The circuit for Half Adder using Basic Gates is shown in Figure 5.2.1

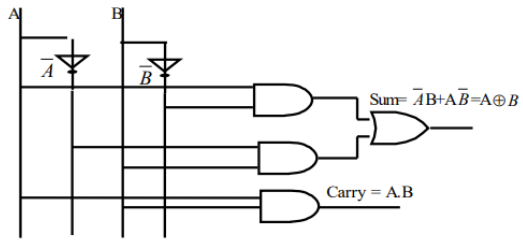


Figure 5.2.14 Half adder using basic gates.

The circuit for Half Adder using XOR gate is shown in Figure 5.2.15.

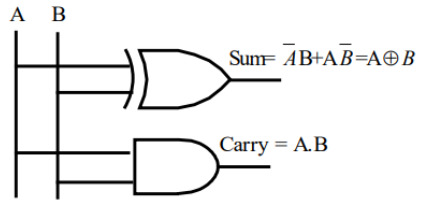


Figure 5.2.15: Half adder using XOR gate.

ii) Full Adder: Full adder is a combinational circuit that performs the arithmetic sum of three input its. It consists of three inputs and two outputs. Two of the inputs are variables, denoted by A and B, which represent the two significant bit to be added. The third input 'Cin' represents

carry from the previous lower significant position. The logical symbol of full adder is shown in Figure 5.2.16 and the truth table is given in table 5.2.9.

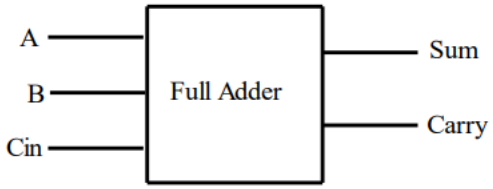


Figure 5.2.16: Full Adder logic symbol.

Table 5.2.9: Truth table for Full Adder

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The expression for sum and carry of full adder is given in expressions 5.2.3 and 5.2.4 respectively.

$$\text{Sum} = \overline{A}\,\overline{B}\,\text{Cin} + \overline{A}\,\text{B}\,\overline{Cin} + \text{A}\,\overline{B}\,\overline{Cin} + \text{ABCin}$$

$$= \overline{A}\,[\overline{B}\,\text{Cin} + \text{B}\,\overline{Cin}] + \text{A}[\overline{B}\,\overline{Cin} + \text{BCin}]$$

$$= \overline{A}\,[\text{B} \oplus \text{Cin}] + \text{A}[\overline{B \oplus Cin}]$$

$$= \text{A} \oplus \text{B} \oplus \text{Cin} \qquad (5.2.3)$$

$$\text{Carry} = \overline{A}\,\text{BCin} + \text{A}\,\overline{B}\,\text{Cin} + \text{AB}\,\overline{Cin} + \text{ABCin} = \overline{A}\,\text{BCin} + \text{A}\,\overline{B}\,\text{Cin} + \text{AB}(\overline{Cin} + \text{Cin})$$

$$= \overline{A}\,\text{BCin} + \text{A}\,\overline{B}\,\text{Cin} + \text{AB} = \overline{A}\,\text{BCin} + \text{A}(\overline{B}\,\text{Cin} + \text{B}) = \overline{A}\,\text{BCin} + \text{AB} + \text{ACin}$$

$$= \text{B}(\overline{A}\,\text{Cin} + \text{A}) + \text{Acin} \quad = \text{B}(\text{A} + \text{Cin}) + \text{Acin}$$

$$= \text{AB} + \text{BCin} + \text{ACin} \qquad (5.2.4)$$

The logic circuit of full adder is shown in Figure 5.2.17.

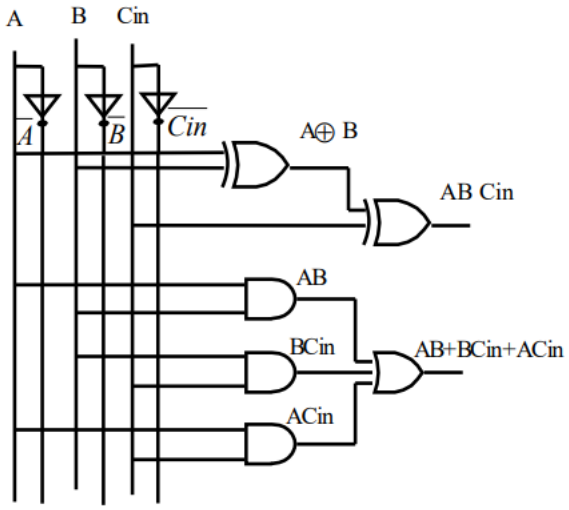
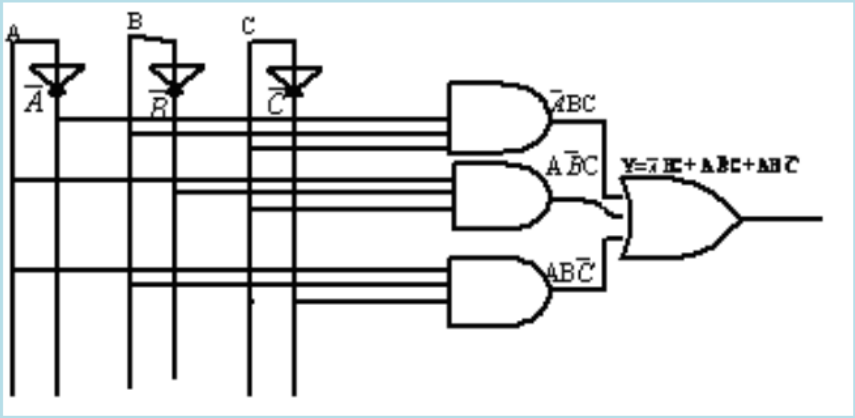


Figure 5.2.17: Full Adder circuit

Ex1: Draw the logic circuit for the Boolean expression. Y= $\overline{A}$BC+A $\overline{B}$ C+ABC'



Summary

1. Logic gates are fundamental building blocks of digital systems
2. The basic set of logic gates are AND, OR and NOT and this set is called Universal set.
3. NAND and NOR are called Universal gates.
4. Inputs and outputs of logic gates can occur in two levels. These two levels are termed as HIGH and LOW, or TRUE and FALSE, or ON and OFF, or simply 1 or 0.
5. Logic circuits whose output at any instant of time is entirely dependent upon the input signals present at that time are known as combinational digital circuits.
6. Logic circuits whose output at any instant of time depend, not only on the present input but also on the past outputs are called Sequential Circuits.

Exercise:

1. Implement Full Subtractor using Basic gates.
2. Implement full adder using two half adders and an OR gate.
3. Simplify and realize using only NAND gates
4. XYZ+XYZ+YZ+$\overline{Z}$.
5. (A+ $\overline{B}$ +C) ($\overline{A}$ +B+$\overline{C}$)(A+$\overline{B}$)
6. Simplify and realize using only NOR gates
   a. Y=A $\overline{B}$ $\overline{C}$+$\overline{A}$ $\overline{B}$ $\overline{C}$+$\overline{B}$ $\overline{C}$+A$\overline{C}$

Module 3: Simplification of Boolean functions using K-map

Maurice Karnaugh, a telecommunications engineer, developed the Karnaugh map at Bell Labs in 1953 while designing digital logic based telephone switching circuits. The Karnaugh map, also known as the K-map, is a graphical method to simplify Boolean algebraic expressions. Karnaugh maps reduce a logic function more quickly and easily compared to Boolean algebra and thus reduces cost of the circuit in terms of inputs and gates.

The required Boolean results are transferred from a truth table onto a two-dimensional grid. The cells are addressed by Gray code, and each cell represents one combination of input conditions (Minterm or Maxterm), while each cell value represents the corresponding output of the function. Optimal groups of 1s or 0s are identified, which represent the terms of a canonical form of the logic in the original truth table. These terms can be used to write a minimal Boolean expression representing the required logic.

Learning Outcomes:

At the end of this module, students will be able to:

1. Describe standard form of Boolean expressions.
2. Apply K-map method for simplification of Boolean expressions.

5.3.1 Standard forms of Boolean expressions

Sum of product (SOP) is the logical expression in which OR of multiple product terms are present. Each product term is the logical AND of literals. Y+XY'+XYZ is an example of SOP. In SOP all the individual terms do not involve all literals. If each term in SOP expression contains all the literals then that SOP is known as standard or canonical SOP.

Every individual term in the standard SOP is a Minterm. Product of sums (POS) is the logical expression in which AND of multiple sum terms are present. Each sum term is the logical OR of literals. (X+Y')((X'Y+Z)(X+Y+Z) is an example of POS. In POS all the individual terms do not involve all literals. If each term in POS expression contains all the literals then that POS is known as standard or canonical POS. Every individual term in the standard POS is a Maxterm.

a) Steps to convert SOP to canonical SOP:

1. Find the missing literal in each product term.

2. AND each product term having missing literals with terms by OR ing the literal and its complement.

3. Expand the terms and reduce the expression by removing repeated terms.

Ex1: F(A,B,C) = AC+AB+BC

In first term B is missing literal, in second term C is missing literal and in third term A is missing literal.

F = AC (B+B') + AB (C+C') + BC (A+A')

F = ACB + ACB' + ABC + ABC' + ABC + A'BC

F = ABC + A'BC + AB'C + ABC'

b) Steps to convert POS to canonical POS:

1. Find the missing literal in each sum term.

2. OR each sum term having missing literals with terms by AND ing the literal and its complement.

3. Expand the terms and reduce the expression by removing repeated terms.

Ex2: F (A, B, C) = (A+B)(B+C)(C+A)

In first term C is missing literal, second term A is missing literal and third term B is missing literal.

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

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

[A+BC=(A+B)(A+C)….distributive property & X.X =X]

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

The canonical SOP and Canonical POS representation of a Boolean function are complementary. The variables which exists in SOP representation do not appear in POS representation. Similarly all the variables which exist in POS representation do not appear in SOP representation. To convert from one canonical form to other canonical form, the symbols Σ and π will be interchanged and the list of variables will be present in new form which is actually missing from original form.

Ex3: Determine the Boolean function from the truth table in terms of minterms. Also give canonical POS form of the expression.

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

5.3.2 Introduction to Karnaugh Map (K-map)

K- map is a matrix of squares and each square represents a minterm or maxterm of a Boolean expression. It is used to simplify the Boolean expression. Each map lists 2n product terms that can be formed from ‘n’ variables, each in a different square. A product term in ‘n’ variables is called a minterm.

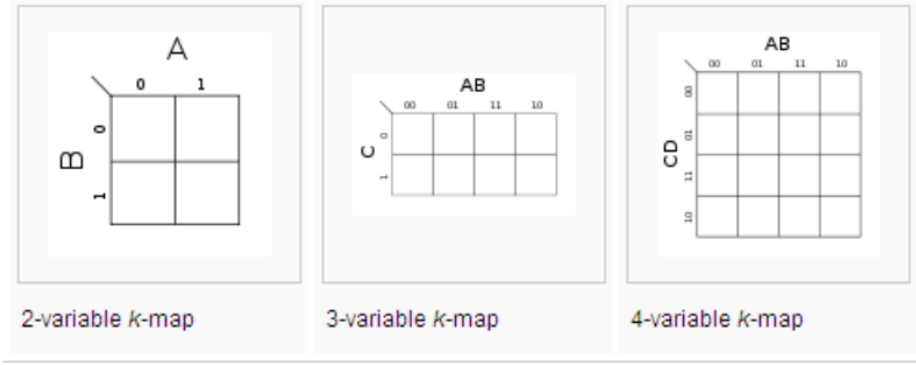


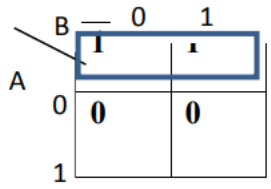
Figure 5.3.1: Variable sized K Maps.

Size of K-map: The size of K- map with n Boolean variables is determined by 2n . The size of the group within a K-map with n Boolean variables and k number of terms in the resulting Boolean expression is determined by 2nk. Common sized maps are of 2 variables which is a 2×2 map, 3 variables which is a 2×4 map and 4 variables which is a 4×4 map. These maps are shown in Figure 5.3.1.

5.3.3 Simplification of Boolean expressions using K-map

The K-Map method may theoretically be applied for the simplification of any Boolean expression regardless of its number of variables, but is most often used when there are fewer than six variables because K-Maps of expressions with more than six variables are complex and tedious to simplify. Each variable contributes two possibilities: the complemented and un-complemented forms. It therefore organizes all possibilities of the system. The variables are arranged in Gray code in which only one variable changes between two adjacent grid boxes.

a) Two variable K-map



f(A,B)=A'

b) Three variable K-map

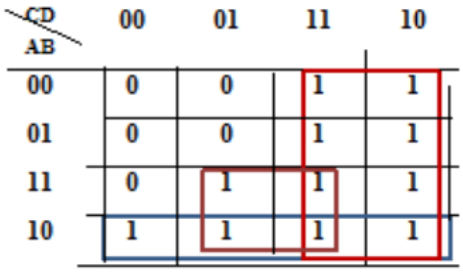
Simplify the following Boolean expression using K Map.

F (A,B,C) = Σm(0,1,3,7)

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

F = A'B' + BC

c) ***Four variable K-map***



f(A,B,C,D)=AB'+AD+C

Ex: For the following truth table construct the K Map

Sl.No.	A	B	C	D	f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

K-map generally becomes more cluttered and hard to interpret when the number of variables increase. For expressions with larger numbers of variables, we have other algorithms. One such algorithm is called Quine McCluskey algorithm and is suitable for automation.

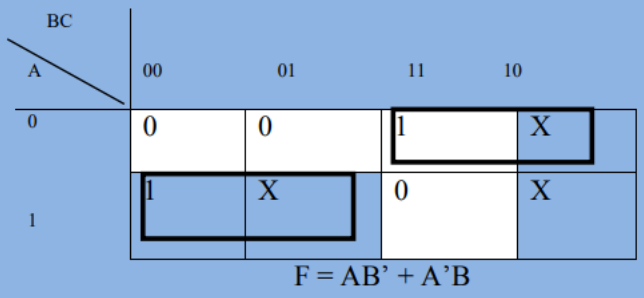
Note:

- *When moving horizontally or vertically, only one variable changes between adjacent squares. This property of K-map is unique and accounts for its unusual numbering system.*
- *In all K-maps, left and right edges are a common edge, while top and bottom edges are also common. Thus top and bottom rows are adjacent, as are the left and right columns.*

So far, the expressions considered have been completely specified for every combination of the input variables, that is, each minterm (maxterm) has been specified as a 1 or a 0. For certain input combinations, the value of the output is unspecified either because the input combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the values of the expression are not specified are called don't care combinations. The don't care terms are denoted by 'd' or 'X'. During the process of design using SOP map, each don't care is treated as a 1 if it is helpful in map reduction; otherwise it is treated as a 0 and left out. During the process of design using a POS map, each don't care is treated as 0 if it is useful in reduction, otherwise it is treated as a 1 and left out.

Ex1: Simplify the following Boolean expression using K Map.

$F(A,B,C) = \Sigma m(3, 4) + d(2,5,6)$



$F = AB' + A'B$

Summary

1. Any Boolean expression can be expressed in a standard or canonical or expanded sum (OR) of products (AND) form –SOP form or in a standard or canonical or expanded product (AND) of sums (OR) form – POS form.
2. The K-map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum of product form.
3. The K-map is useful in Boolean expression simplification and hence further reduces the cost for logic realization.
4. It is a means of showing the relationship between the logic inputs and desired output.
5. K-map is limited to 6 variables.

Exercise:

1. Consider the truth table of a function. Transfer the outputs to the K map and write the Boolean expression.

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

2. Simplify the following Boolean expressions using K maps.

   (a) $F = \Sigma m(0,2,4,6)$

   (b) $F = \Sigma m(0,2,4,6) + d(5,7)$.

Chapter -6: SEQUENTIAL CIRCUITS

Sequential circuits are digital circuits whose output at a given time is dependent on the input at that time and also on the earlier inputs. Thus sequential circuits always have a memory associated with them. The operation of sequential circuits is expressed in terms of their states. The basic memory element in sequential circuits is the flip-flop. A flip-flop can store a 1 or a 0 and hence is called one bit memory.

Module – 1 : Flip-Flops and its Applications

Learning Outcomes:

At the end of this module, students will be able to:

1. Differentiate latches and Flip flops.
2. Draw the circuit of SR, D, JK and D flip-flop using NAND gates and explain its working principle with its truth table.
3. Design ripple up/down counter using flip flop
4. Explain the working principle of SISO, SIPO shift registers.

6.1.1 Introduction to Flip-Flops

It is an electronic circuit or device which is used to store data in binary form. A flip-flop is a sequential circuit used as a one bit memory element. A flip-flop circuit can be constructed from NAND gates or NOR gates. It has two stable outputs Q and Q' and hence it is also known as Bi-stable circuit.

History of Flip-Flop

The first electronic flip-flop was invented in 1918 by William Eccles and F. W. Jordan. It was initially called the *Eccles–Jordan trigger circuit* and consisted of two active elements (vacuum tubes) now it is realized using transistors.

A "clock" is derived using a special circuit that sends electrical pulses, called clock in the context of digital applications as shown in Figure 6.1.1. Each pulse has a precise width and there is a precise interval between pulses – known as clock cycle time. Figure 6.1.2 shows various parts of the clock signal for which the circuit will respond.

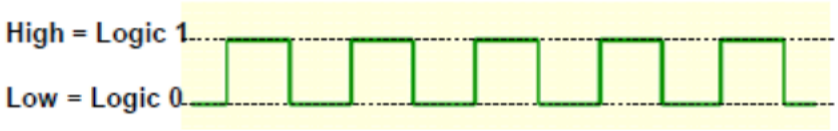


Figure 6.1.1 Clock Signal

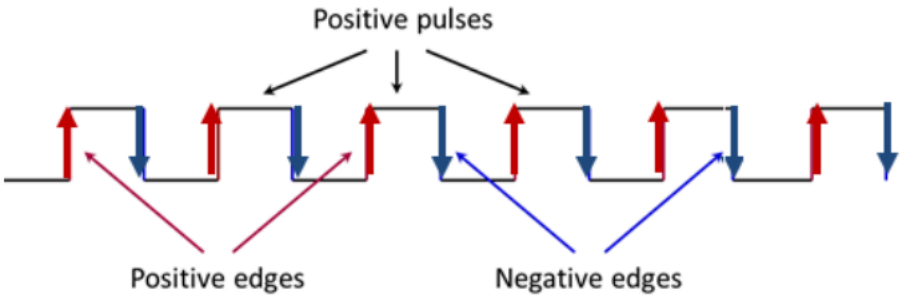


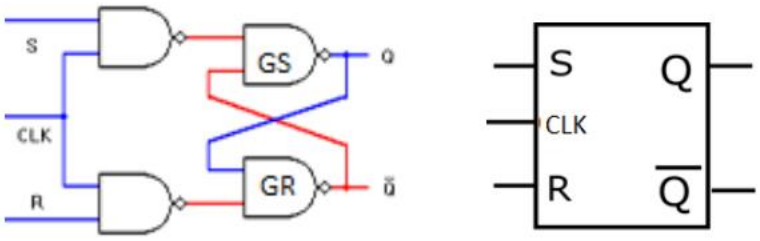
Figure 6.1.2 Edges and levels of a Clock Signal

b) Flip-Flop and Latch

The word latch is mainly used for storage elements, while clocked devices are described as flip-flops. A latch is level-sensitive, whereas a flip-flop is edge-sensitive. Based on the triggering condition, Flip flops are divided into positive edge-triggered (active when transition of clock is 0 to 1) and negative edge-triggered (active when transition of clock is 1 to 0).

c) SR Flip-Flop:

The SR flip-flop can be considered as one of the most basic sequential logic circuit possible.This simple flip-flop is basically a one-bit memory bi-stable device that has two inputs, one which will "SET" the device (output Q = "1"), and is labeled S and another which will "RESET" the device (output Q = "0"), labeled R.



a) b)

Figure 6.1.3 (a): Logic diagram of Clocked SR Flip flop (b) Logical Symbol

A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit as shown in Figure 6.1.3 (a). The SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it's current state or history. The term "Flip-flop" relates to the actual operation of the device, as it can be "flipped" into one logic state (SET) or "flopped" back into the opposing logic state (RESET). Figure 6.1.3 (b) shows symbolic representation of SR Flip-Flop.

The operation of S-R Flip Flop can be described as follows::

When Clock is High:

- When S input is made '1' and R input is made '0', the Q output takes the state '1'. This is known as SET condition.
- When S input is made '0' and R input is made '1', the Q output takes the state '0'. This is known as RESET condition.
- When S = R = '0', the SR flip flop exhibits the memory. i.e. Qn+1 = Qn ; Holds the state.
- S = R = 1 is the Not allowed in S-R Flip flop.

When Clock is Low:

- Irrespective of S and R inputs, the SR flip flop exhibits the memory. i.e. Qn+1 = Qn : No change in the state.

The function of the circuit is described by the table 6.1.1. This table is called Truth Table. Q(n+1) represents the next state of the output and Qn corresponds to the previous state.

Table 6.1.1: Truth Table of SR Flip-Flop

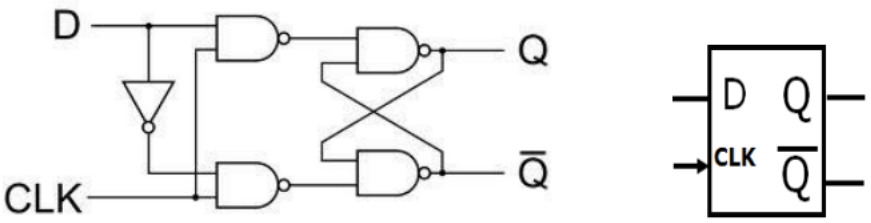
CLK	S	R	Q(n+1)	Mode
1	0	0	Qn	Previous Output
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	Invalid	Invalid
0	X	X	Qn	Previous Output

Note: Practical available Flip flops are edge triggered, that means flip flop responds for clock edges either positive edge or negative edge. Following are the symbolic representation of positive edge and negative edge triggered flip flops. Observe the 'not' (bubble) symbol at the clock of negative edge triggered flip flop.

a) Positive edge triggered Flip flop b) Negative edge triggered Flip flop

d) D Flip-Flop:

The D flip-flop shown in Figure 6.1.4 (a) is a modification of the clocked SR flip-flop. The D input goes directly into the S input and the complement of the D input goes to the R input. The D input is sampled during the occurrence of a clock pulse. If it is 1, the flip-flop is witched to the set state (unless it was already set). If it is 0, the flip-flop switches to the clear state. The R and S inputs will be always in complimentary form, thus preventing R = S = 1 condition from occurring. When the clock (CLK) is low, previous data is stored, i.e, it exhibits memory as shown in table 6.1.2.



a) b)

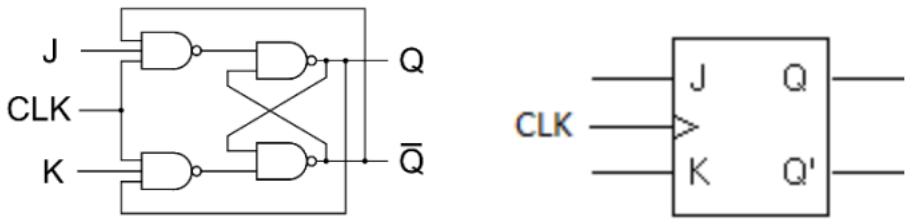
Figure 6.1.4: (a) Logic diagram of D Flip flop (b) Logic Symbol

Table 6.1.2: Truth Table

En	D	Q(n+1	Mode
1	0	0	Reset
1	1	1	Set
0	X	Qn	Previous Output

e) JK Flip-Flop:

A JK flip-flop is a refinement of the SR flip-flop, in which the indeterminate state of the SR type is defined in the JK type as shown in figure 6.1.5 (a). Inputs J and K behave like inputs S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and the letter K is for clear). When logic 1 inputs are applied to both J and K simultaneously, the flip flop switches to its complement state, ie., if Q=1, it switches to Q=0 and vice versa.



a) b)

Figure 6.1.5: (a) Logic Diagram of JK Flip-flop (b) Logic symbol

JK flip flop is one of the most versatile flip flop. The operation of JK flip flop can be explained as follows:

When Clock is High:

- J = K = 0 causes memory condition.
- J = 0 & K = 1 causes reset condition.
- J = 1 & K = 0 causes set condition.
- J = K = 1 causes output to toggle. i.e. Qn+1 = Qn'

When Clock is Low:

- Irrespective of J & K inputs, the J-K flip flop exhibits the memory condition.

All these characteristics are summarized in Table 6.1.3

Table 6.1.3 Truth Table of JK flip-flop

CLK	J	K	Q(n+1)	Mode
1	0	0	Qn	Previous Output
1	0	1	1	Reset
1	1	0	0	Set
1	1	1	Qn'	Toggle
0	X	X	Qn	Previous Output

f) T Flip-flop:

Figure 6.1.6 shows the logic circuit of T flip-flop in which J and K inputs of a JK flip flop are combined and taken as a single input T. When T = 0, output of T flip-flop will remain as it was previously. When T = 1, output of T flip-flop will be complement of its previous output and hence this circuit is known as toggle circuit. The excitation table is as shown in Table 6.1.4.

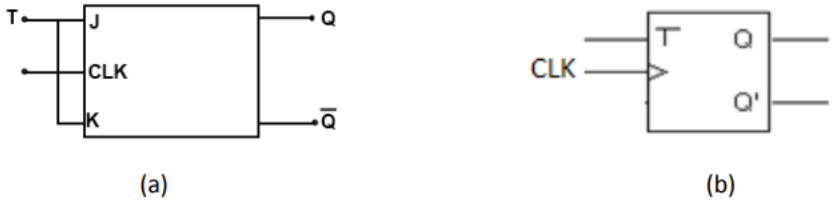


Figure 6.1.6: (a) Logic Diagram of T Flip-flop (b) Logic symbol of T Flip-flop

Table 6.1.4: Truth table of T flip-flop

CLK	T	Q(n+1)	Mode
1	0	Qn	No change
1	1	Qn'	Toggle
0	X	Qn	Previous Output

Self test:

1. *Implement clocked SR flip flop and JK flip flop using only NOR gates and write the truth table.*
2. *Convert SR flip flop to D flip and T flip flop. Show using appropriate logic diagram.*
3. *Compare a) SR and JK flip flop b) T and D flip-flop.*
4. *Draw the timing diagram for a) SR Flip flop and b) JK Flip flop*

6.1.2 Counters:

A digital counter is a set of flip flops whose states change in response to pulses applied at the input to the counter. In binary counter, the flip flops are interconnected such that their combined state at any time is the binary equivalent of the total number of pulses that have occurred up to that time. Thus a counter is used to count pulses. Each of the counts of the counter is called state of the counter. The number of states through which the counter passes before returning to the starting state is called modulus of the counter. In general an n-bit counter will have 'n' flip flops and 2n states and divides the input frequency by 2n. Hence it is a divide by 2n counter.

a) Classification of counters

i) Synchronous and Asynchronous counters :

Counters may be synchronous or asynchronous counters as shown in Figure 6.1.7. In synchronous counters, flip flops are not triggered simultaneously. Synchronous counters are clocked such that each flip flop in the counter is triggered at the same time. This is accomplished by connecting the same clock line to each stage of the counter. Synchronous counters are faster than asynchronous counters because the propagation delay involved is less.

ii) Up counter or Down Counter:

A counter may be an up counter or a down counter. An up counter is a counter which counts in the upward direction i.e 0, 1, 2, 3,…, N. A down counter is a counter which counts in the downward direction i.e N, N-1, N-2,…,1, 0.

Note:

Practical available Flip flops are edge triggered, so counters can be realized using positive edge triggered or negative edge triggered flip flops.

We can realize other than 2^n counter by appropriately setting the asynchronous inputs of flip flop using logic gates. For example Decade counter counts 10 states of a counter which is not a 2^n

b) Applications of counter:

Counters are used

1. As Frequency divider,
2. To perform the timing function as in digital watches
3. To create time delays
4. To produce non-sequential binary counts
5. To generate pulse trains
6. To act as frequency counters.

c) Ripple Counters:

Asynchronous counters are also called as ripple counters. In ripple counters the flip flops within the counter are not made to change the states at exactly the same time. In this counter, JK flip flops or T flip flops are used with J and K inputs or T input of the flip flops connected to 1. This makes the flip flop output to toggle when a clock pulse is applied. It is also called as serial or series counters.

Example 3:
Design Two bit ripple up counter using negative edge triggered JK flip flops:
Steps:

1. Select 2 JK flip flops (number of flip flops depends upon number of bits to count).
2. Connect JK inputs to high
3. Apply the –ve edge clock pulse to first JK flip flop
4. Write the truth table of 2 bit up counter as shown where Q1 and Q2 are the outputs of 2 JK flip flops.

Clk pulse	Q2	Q1(LSB)
1	0	0
2	0	1
3	1	0
4	1	1

5. Observe when higher bit (Q2) is changing. In this case, Q2 is changing when Q1 is changing from logic 1 to logic 0 (At clock pulse 3). Since we require –ve edge counter which responses when clock is changing from 1 to 0 is required, connect Q1 as the clock for the next flip flop as shown in Figure 6.1.7 (a)

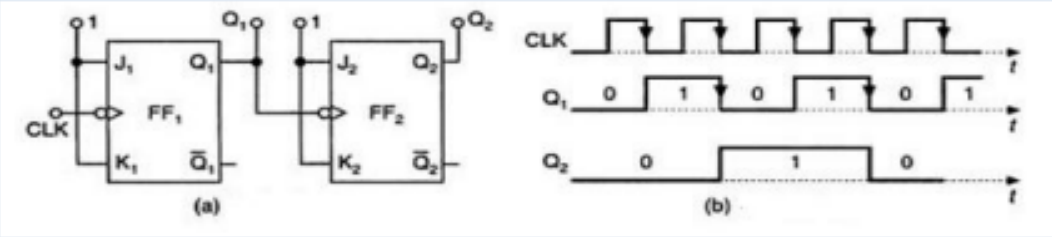


Figure 6.1.7 (a) Asynchronous 2 bit up counter using -ve edge triggered flip flops (b) Timing diagram

Working principle: For the clock pulse applied to FF1, the output of FF1 toggles. For the Q_1 pulse applied to FF2, the output of FF2 toggles. Thus the 2 bit up counter counts in the order 0, 1, 2, 3, 0, 1, ….Figure 6.1.7 shows a 2 bit ripple up counter using negative edge triggered JK flip flops and its timing diagram.

Example 4:

Design Two bit ripple up counter using positive edge triggered JK flip flops:

Steps:

Steps 1and 2 are same as example 3

3 Apply the +ve edge clock pulse to first JK flip flop (Observe the Clk signal in Figure 6.1.7

4 Write the truth table of 2 bit up counter as in example 6.3

5 Observe when higher bit (Q2) is changing. In this case, Q2 is changing when Q1 is changing from logic 1 to logic 0. Since we require +ve edge counter which responses when clock is changing from 0 to 1 is required, connect Q1' (Q1 complement) as the clock for the

next flip flop (because when Q1 is changing 1 to 0 Q1' will change 0 to 1 which is the positive edge of the clock) as shown in Figure 6.1.8

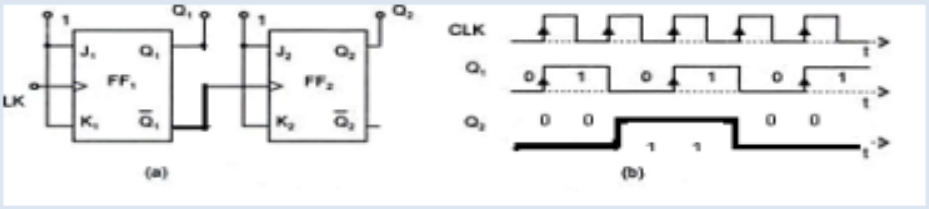


Figure 6.1.8 (a) Asynchronous 2 bit up counter using +ve edge triggered flip flops. (b) Timing diagram.

Working principle: The output Q_1' of FF1 is connected to the clock input of FF2. For the clock pulse applied to FF1, the output of FF1 toggles. For the Q_1' pulse applied to FF2, the output of FF2 toggles. Thus the 2 bit up counter counts in the order 0, 1, 2, 3, 0, 1,

Example 5:

Design Two bit ripple down counter using negative edge triggered JK flip flops:

Steps: Steps 1, 2 and 3 are same as example 3

4 Write the truth table of 2 bit down counter as shown

Clk pulse	Q2	Q1(LSB)
1	1	1
2	1	0
3	0	1
4	0	0

5 Observe when higher bit (Q2) is changing. In this case, Q2 is changing when Q1 is changing from logic 0 to logic 1. Since we require –ve edge counter which responses when clock is changing from 1 to 0, connect Q1' (which changes 1 to 0 at clock pulse 3) as the clock for the next flip flop as shown in Figure 6.1.9

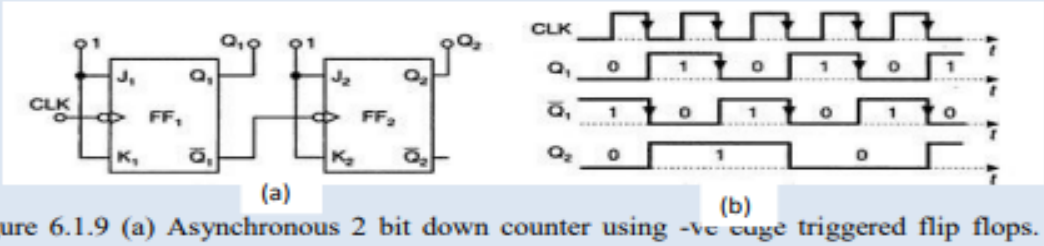


Figure 6.1.9 (a) Asynchronous 2 bit down counter using -ve edge triggered flip flops. (b) Timing diagram.

Working principle: The output Q_1' of FF1 is connected to the clock input of FF2. For the clock pulse applied to FF1, the output of FF1 toggles. For the Q_1' pulse applied to FF2, the output of FF2 toggles. Thus the 2 bit down counter counts in the order 0, 3, 2, 1, 0, 3 … Figure 6.1.8 shows a 2 bit ripple down counter using negative edge triggered JK flip flops and its timing diagram.

Self test:

1. *What is the modulus of 2 bit counter?*
2. *Why ripple counters are known as divide by n counter (In case of 3 bit counter, it is divide by 3 counter) ? (* Hint: analyze the frequency at the output of n^{th} flip flop with respect to clock signal)*
3. *Realize 2 bit +ve edge triggered up counter using T flip flop.*

6.1.3 Shift Register

Registers are digital circuits which are used to store 'n' bits information in the same time. Generally registers are built with D flip flops. Each flip flop can store a one bit, so a register composed of 'n' flip flops can store 'n' bit number. All these flip flops are driven by a common clock and they are set or reset simultaneously. Therefore, the data processing happens sequentially. The output of one flip flop is fed as input to the next flip flop.

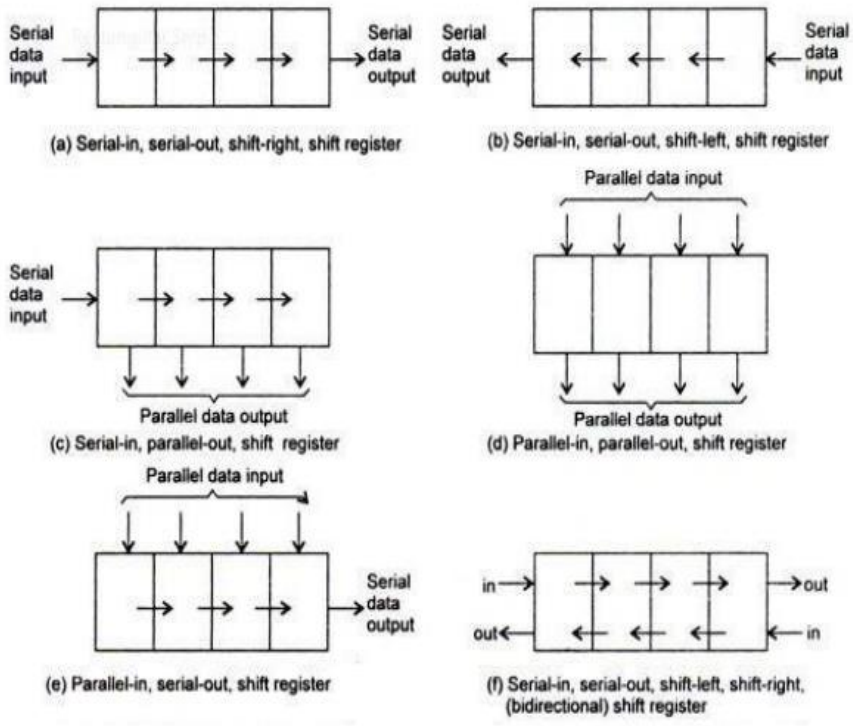


Figure 6.1.10 Data transmission in shift register

In a register data can be entered in serial form or data can also be output in serial form. Then this register is called as shift register since data bits are shifted in the flip flops with each clock pulse. Data can be shifted either towards right or left or even in both the directions. When the data is shifted from left to right, it is known as right shift register. If data is shifted right to left, it is called as left shift register. In bidirectional shift register, data can be shifted either left to

right or right to left, depending upon the mode control signal. Data transmission in shift register is shown in Figure 6.1.10.

There are four basic types of shift registers namely, Serial in Serial out (SISO), Serial in Parallel out (SIPO), Parallel in Parallel out (PIPO) and Parallel in Serial out (PISO) shift registers.

a) Serial In Serial Out Shift Register (SISO) :

In SISO shift register, data input is in serial form and clock pulses are applied to each flipflop. After each clock pulse, data moves by one position. The output can be obtained in serial form. In this type of shift register data moves either in left or right direction. The logic diagram of a 4 bit SISO shift right shift register is shown in Figure 6.1.11 with four flip flops, the register can store up to four bits of data. Serial data is applied at the D input of the FF1. The Q output of FF1 is connected to the D input of FF2, the Q output of FF2 is connected to the D input of FF3 and the Q output of FF3 is connected to D input of FF4. When serial data is transferred into a register, each new bit is clocked into the first flip flop FF1 at the positive going of each clock pulse. The bit that was previously stored by FF1 is transferred to FF2. The bit that was stored by FF2 is transferred to FF3 and so on. The bit that was stored by the last FF4 is shifted out.

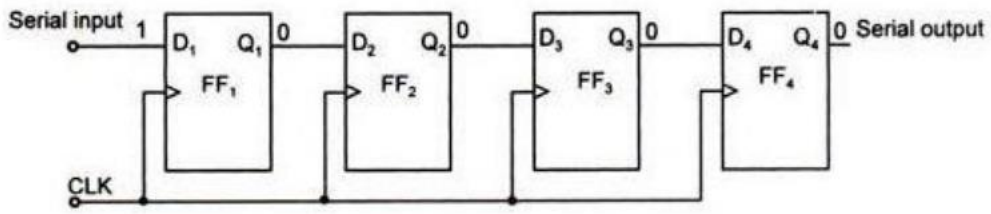


Figure 6.1.11 SISO Shift Register

Table 6.1.5 shows the operation of 4 bit SISO shift register to store 1010(2). The LSB bit is entered first in the register D1 and 4 clock pulse required to store 4 bit data and 5th clock pulse Data present in D4 will be taken out. So SISO requires (2n-1) clock pulses to take out n bit data i.e in above example MSB bit 1 is taken out from Q4 at 7th clock pulse

Table 6.1.5 Data shifting in 4 bit SISO shift register for data 1010

Clk Pulse	Q1	Q2	Q3	Q4
Before the CLK	#	#	#	#
Clk 1	0	#	#	#
Clk 2	1	0	#	#
Clk 3	0	1	0	#
Clk 4	1	0	1	0
Clk 5	#	1	0	1
Clk 6	#	#	1	0
Clk 7	#	#	#	1
Clk 8	#	#	#	#

Data is loaded

MSB is available

Note: # indicates any random data

b) Serial In Parallel Out Shift Register (SIPO):

SIPO shift register is shown in Figure 6.1.12. In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form. Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output. The serial-in parallel-out shift register can be used as a serial-in serial-out shift register if the output is taken from the Q terminal of the last FF.

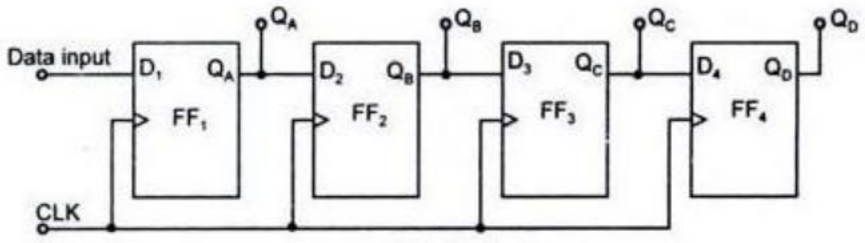


Figure 6.1.12: SIPO Shift Register

Table 6.1.6 Data shifting in 4 bit SIPO shift register for data 1010

Clk Pulse	Q1	Q2	Q3	Q4
Before the CLK	#	#	#	#
Clk 1	0	#	#	#
Clk 2	1	0	#	#
Clk 3	0	1	0	#
Clk 4	1	0	1	0

Data is loaded and can be taken out

c) Applications of shift Registers:

Shift registers are commonly used

1. To store digital data during arithmetic and logical operations.

2. To build Shift register counters.

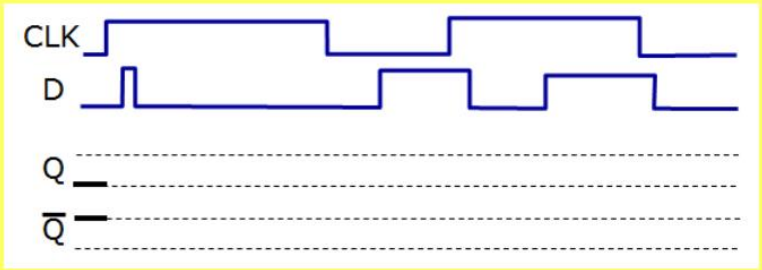
3. To generate timing and control waveforms.

Summary

1. latch is a level triggered bi-stable multi-vibrator circuit while a flip flop is an edge triggered.
2. Flipflop is an electronic circuit or device which is used to store data in binary form. . Latch is a class of flip-flops whose output responds immediately to appropriate changes in the input.
3. We have studied the working principle of SR, D, JK and T flip flops using NAND gates.
4. In binary counter, the flip flops are interconnected such that their combined state at any time is the binary equivalent of the total number of pulses that have occurred up to that time.
5. Asynchronous counters are also called as ripple counters
6. Registers are digital circuits which are used to store 'n' bits information in the same time
7. Shift registers are commonly used to store digital data during arithmetic and logical operations.

Exercises:

1. ***Complete the timing diagram of D flip-flop***



2. ***Design mod 4 down counter using +ve edge triggered T flip flop.***
3. ***Design 4 bit up counter using –ve edge triggered JK flip flop with the neat timing diagram***
4. ***Compare SISO and SIPO***
5. ***Consider data 101110 is given to SISO,SIPO. Data is entered from LSB. After how many clock pulse, MSB is retrieved?***